

# Web Service 안내서

JEUS 9

**TMAXSOFT**

## 저작권 공지

Copyright 2024. TmaxSoft Co., Ltd. All Rights Reserved.

## 제한된 권리

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

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

## 상표 공지

JEUS®는 TmaxSoft Co., Ltd.의 등록 상표입니다.

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

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

HP-UX는 Hewlett Packard Enterprise Company의 등록 상표입니다.

AIX는 International Business Machines Corporation의 등록 상표입니다.

UNIX는 X/Open Company, Ltd.의 등록 상표입니다.

Linux는 Linus Torvalds의 등록 상표입니다.

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

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

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

## 오픈 소스 소프트웨어 공지

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

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

`${INSTALL_PATH}/license/oss_licenses`

## 기술 지원

홈페이지: <https://www.tmaxsoft.com>

기술 서비스 센터: 1544-8629

## 안내서 이력

제품 버전	안내서 버전	발행일	비고
JEUS 9	3.1.1	2024-09-27	-

# 목차

용어 해설 .....	1
1. 웹 서비스 .....	2
1.1. 기본 개념 .....	2
1.2. 웹 서비스 표준 .....	3
1.2.1. SOAP 표준 .....	3
1.2.2. WSDL 표준 .....	4
1.2.3. UDDI 표준 .....	4
1.2.4. JAX-WS, JAXB, StAX 표준 .....	5
1.2.5. SAAJ 표준 .....	5
1.3. SOAP 메시지 교환과 SOAP 메시지 인코딩 .....	5
2. JEUS 웹 서비스 .....	6
2.1. 기본 구조 .....	6
2.2. 웹 서비스 설계 .....	7
2.2.1. 웹 서비스 Back-end 선택 .....	7
2.2.2. JEUS 웹 서비스의 방식 .....	8
2.2.3. 웹 서비스 구현 방식 선택 .....	8
2.2.4. SOAP 메시지 핸들러 생성 .....	9
Part I. JEUS 9 웹 서비스 .....	10
3. JEUS 웹 서비스 구현 .....	11
3.1. 개요 .....	11
3.2. Java 클래스 웹 서비스 구현 .....	11
3.3. EJB 웹 서비스 구현 .....	15
3.4. WSDL로부터 웹 서비스 구현 .....	20
4. 웹 서비스 생성과 배치 .....	25
4.1. 개요 .....	25
4.2. Java 클래스 웹 서비스 생성과 배치 .....	25
4.3. EJB 웹 서비스 생성과 배치 .....	26
4.4. WSDL 웹 서비스 생성과 배치 .....	26
4.5. Endpoint 주소 결정 방식 .....	27
4.5.1. 서블릿 Endpoint .....	27
4.5.2. EJB Endpoint .....	29
5. 웹 서비스 호출 .....	33
5.1. 개요 .....	33
5.2. 동적 프록시 방식의 웹 서비스 호출 .....	33
5.2.1. Client Artifact 생성 .....	33
5.2.2. Java SE 클라이언트 방식 .....	38
5.2.2.1. Java SE 클라이언트 프로그램 작성 .....	38
5.2.2.2. Java SE 클라이언트 프로그램 호출 .....	39
5.2.3. Jakarta EE 클라이언트 방식 .....	40

5.2.3.1. Jakarta EE 클라이언트 프로그램 작성	40
5.2.3.2. Jakarta EE 클라이언트 프로그램 호출	41
5.3. 디스패치 방식의 웹 서비스 호출	42
6. 표준 바인딩 선언 및 사용자화	43
6.1. 개요	43
6.2. 표준 바인딩 선언	43
6.2.1. 외부 문서(파일)에서 직접 선언	43
6.2.2. WSDL 문서 내에서 직접 선언	44
6.3. 표준 바인딩의 사용자화	45
6.3.1. 전체적인 바인딩	45
6.3.2. 패키지명의 사용자화	46
6.3.3. Wrapped 스타일	46
6.3.4. 비동기화	47
6.3.5. 프로바이더 인터페이스	47
6.3.6. 클래스명의 사용자화	48
6.3.7. Java 메소드의 사용자화	49
6.3.8. Java 파라미터의 사용자화	49
6.3.9. XML 스키마의 사용자화	50
6.3.10. 핸들러 체인의 사용자화	50
7. 핸들러 프레임워크	52
7.1. 개요	52
7.2. 핸들러 체인 우선순위	52
7.3. 핸들러 클래스 구성	53
7.3.1. 핸들러 클래스의 선언	53
7.4. 핸들러 클래스 설정	54
7.4.1. Java 클래스로부터 웹 서비스 구성	54
7.4.2. WSDL로부터 웹 서비스 구성	55
7.4.3. 클라이언트 구성	56
7.5. 핸들러 체인을 사용하는 웹 서비스 예제	56
7.6. 웹 서비스의 핸들러 프레임워크 실행	58
8. 프로바이더와 디스패치 인터페이스	60
8.1. 개요	60
8.2. 서비스 Endpoint 프로바이더 인터페이스	60
8.2.1. 프로바이더 인터페이스	60
8.2.2. 프로바이더 인터페이스 예제	61
8.2.3. 프로바이더 인터페이스 예제 실행	62
8.3. 클라이언트 디스패치 인터페이스	63
8.3.1. 디스패치 인터페이스	63
8.3.2. 디스패치 인터페이스 예제	65
8.3.3. 디스패치 예제 실행	66
8.4. XML/HTTP 바인딩	67

8.4.1. RESTful 웹 서비스 .....	67
8.4.2. RESTful 웹 서비스 예제 .....	68
8.4.3. RESTful 웹 서비스 예제 실행 .....	70
9. 비동기 웹 서비스 .....	72
9.1. 개요 .....	72
9.2. 클라이언트 비동기 오퍼레이션 .....	72
9.2.1. 비동기 메소드를 가진 SEI Stub 이용 방법 .....	72
9.2.1.1. 비동기화 바인딩 선언 .....	72
9.2.1.2. 비동기화 클라이언트 구성 .....	73
9.2.1.3. 비동기화 클라이언트 실행 .....	76
9.2.2. 디스패치 인터페이스 이용하는 방법 .....	77
9.3. 비동기 웹 서비스 .....	77
9.3.1. 비동기 웹 서비스 설정 .....	77
10. MIME Attachment 메시지 전송 .....	79
10.1. 개요 .....	79
10.2. MTOM/XOP .....	79
10.2.1. 기본 동작 .....	80
10.2.2. Attachment 바이너리 데이터 크기 설정 .....	81
10.2.3. MTOM/XOP 예제 .....	82
10.2.4. MTOM/XOP 예제 실행 .....	83
10.3. swaRef .....	84
10.3.1. swaRef 사용법 .....	84
10.3.2. swaRef 예제 .....	86
10.3.3. swaRef 예제 실행 .....	87
10.4. 스트리밍 방식으로 첨부 파일을 처리하는 방법 .....	88
11. Fast Infoset 웹 서비스 .....	89
11.1. 개요 .....	89
11.2. Fast Infoset 사용 .....	90
11.2.1. Content Negotiation .....	91
11.3. Fast Infoset 예제 .....	91
11.4. Fast Infoset 웹 서비스 실행 .....	92
12. JAX-WS JMS 기반 전송 .....	94
12.1. 개요 .....	94
12.2. JAX-WS JMS 기반 전송 설정 .....	94
12.2.1. JMS 서버 설정 .....	94
12.2.2. 웹 서비스 작성 .....	95
12.2.3. WSDL 설정 .....	95
12.2.4. 웹 서비스 클라이언트 작성 .....	96
13. 웹 서비스 정책 .....	97
13.1. 개요 .....	97
13.2. 웹 서비스의 정책(WS-Policy) .....	97

13.3. 서버 정책 설정	98
13.3.1. WSDL로부터 웹 서비스 구성	99
13.3.2. Java 클래스로부터 웹 서비스 구성	99
13.4. 클라이언트 정책 설정	100
14. 웹 서비스 Addressing	102
14.1. 개요	102
14.2. 서버 설정	103
14.2.1. Java 클래스로부터 설정	103
14.2.2. WSDL로부터 설정	105
14.3. 클라이언트 설정	105
14.4. 예제	106
14.5. 예제 실행	106
15. 신뢰성 메시징 기술	110
15.1. 개요	110
15.2. 서버 설정	110
15.2.1. WSDL로부터 설정	111
15.2.2. Java 클래스로부터 설정	112
15.3. 클라이언트 설정	113
15.4. 예제	113
15.5. 예제 실행	113
16. 웹 서비스 트랜잭션	115
16.1. 개요	115
16.2. 서버 설정	115
16.2.1. WSDL로부터 설정	115
16.2.2. Java 클래스로부터 설정	116
16.3. 클라이언트 설정	117
16.4. 코디네이터 서비스	117
16.5. 웹 서비스 트랜잭션 예제	117
17. 웹 서비스 보안	118
17.1. 개요	118
17.2. 전송 수준 보안	118
17.3. 메시지 수준 보안	119
17.3.1. 웹 서비스 보안 정책	119
17.3.2. 웹 서비스 보안	119
17.3.2.1. 사용자명 인증을 통한 대칭 바인딩의 인증 기능 강화	120
17.3.2.2. 상호 인증 보안	120
17.3.2.3. SSL을 통한 SAML 인증	121
17.3.3. 웹 서비스 보안 대화	122
17.3.4. 웹 서비스 신뢰	122
17.4. 메시지 수준 보안 설정	124
17.4.1. 공통 설정	124

17.4.1.1. 서버 설정	124
17.4.1.2. 클라이언트 설정	136
17.4.2. 사용자명 인증을 통한 대칭 바인딩의 인증 기능 강화	138
17.4.2.1. 서버 설정	138
17.4.2.2. 클라이언트 설정	139
17.4.3. 상호 인증 보안	140
17.4.3.1. 서버 설정	140
17.4.3.2. 클라이언트 설정	141
17.4.4. SSL을 통한 SAML 인증	142
17.4.4.1. 서버 설정	142
17.4.4.2. 클라이언트 설정	142
17.4.5. 보안 대화	144
17.4.5.1. 서버 설정	144
17.4.5.2. 클라이언트 설정	145
17.4.6. 웹 서비스 신뢰	146
17.4.6.1. 서버 설정	146
17.4.6.2. STS 설정	147
17.4.6.3. 클라이언트 설정	149
17.4.7. 실행	150
17.5. JAX-RPC(JEUS 5) 웹 서비스 보안 이전 방법	151
17.5.1. 암호화	152
17.5.2. 서명	154
17.5.3. Timestamp	156
17.5.4. 사용자명 토큰	158
17.6. 접근 제어 설정된 웹 서비스 호출 방법	159
17.6.1. Portable Artifact 생성	159
17.6.2. 웹 서비스의 클라이언트 작성	159
17.7. 접근 제어 설정	160
17.7.1. Java 클래스 웹 서비스 접근 제어 설정	161
17.7.2. EJB 웹 서비스의 접근 제어 설정	162
17.7.3. 접근 제어가 설정된 웹 서비스 호출	164
18. Server-Sent Event	165
18.1. 개요	165
18.2. JAX-RS 리소스에서 Server-Sent Events(SSE) 지원	165
18.3. JAX-RS 클라이언트에서 SSE 이벤트 처리	166
18.3.1. EventInput를 사용하여 SSE 이벤트 읽기	166
18.3.2. EventSource를 사용하여 비동기 SSE 처리	167
19. JEUS 웹 서비스 XML	169
19.1. 개요	169
19.2. JAXB(XML 바인딩을 위한 Java 아키텍처)	169
19.2.1. 바인딩 컴파일러(XJC) 관련 프로그래밍 기법	170



19.2.2. 스키마 생성기(Schemagen) 관련 프로그래밍 기법.....	173
19.3. JAXP(XML을 다루기 위한 Java 표준 API) .....	175
19.3.1. StAX(Java 스트리밍 XML 파서).....	175

# 용어 해설

## In/Out Parameter

In/Out Parameter는 입력과 출력에 사용되는 웹 서비스 오퍼레이션의 파라미터이다.

## JEUS

Java Enterprise User Solution의 약어이다. 본 안내서에서 소개하는 JEUS 9는 Jakarta EE 9 호환 WAS이다.

## Proxy Client

Proxy Client는 wsdl2java Ant Task에 의해 WSDL 문서로부터 생성된 Stub코드를 통해서 작성된 웹 서비스 클라이언트이다.

## SAAJ

SOAP message with Attachments API for Java의 약어이다. SAAJ는 JCP(Java Community Process)에 의해 정의된 SOAP 메시지 처리를 위한 표준 Java API이다.

## SOAP

Simple Object Access Protocol의 약어이다. SOAP은 프로그래밍언어와 운영체제와 상관없이 네트워크에서 XML 데이터를 운반하기 위해 W3C에 의해 정의된 표준 프로토콜이다.

## WAS

Web Application Server의 약어로 복잡한 웹 애플리케이션을 실행하고 관리하는 미들웨어이다.

## WSDL

Web Service Description Language의 약어이다. WSDL은 웹 서비스의 인터페이스와 기술을 구현하기 위해 W3C에 의해 정의된 표준 XML Instance이다.

# 1. 웹 서비스

본 장에서는 웹 서비스에 대한 기본 개념과 표준에 대해서 설명한다.

## 1.1. 기본 개념

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

웹 서비스는 인터넷의 URI로 접근 가능한 응용 프로그램이며, 웹 서비스의 인터페이스와 바인딩은 XML 문서로 정의되고 기술된다. 하나의 웹 서비스는 인터넷에 공개되어진 또 다른 웹 서비스와 상호 연동이 가능할 뿐 아니라 기존의 Back-end 응용 프로그램들과도 연동이 가능하다.

지금까지의 응용 프로그램 아키텍처는 2가지의 범주에 속해 있었다. 하나는 메인프레임을 기반으로 작동하는 단일화된 시스템이며, 다른 하나는 데스크탑에서 작동하는 클라이언트-서버(client-server) 기반의 시스템이었다. 이들 시스템은 물론 모두 잘 작동하지만 이러한 아키텍처 위에 동작하는 프로그램들은 그 시스템에서만 작동할 수 있게 맞추어진 프로그램들이라는 한계점을 가지고 있다. 이러한 아키텍처 내에서의 프로그램들은 아주 폐쇄적이어서 접근이 용이하지 않으므로 웹에 존재하는 무수히 많은 사용자들에게는 무용지물이다.

그래서 소프트웨어 산업은 서비스지향 아키텍처(SOA : Service-Oriented Architecture)라는 방향으로 진화하게 되었고, 이 기반위에서의 응용 프로그램들은 웹에서 동적으로 상호 작용을 할 수 있게 되었다. 응용 프로그램은 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 되었다. 또한 하나의 컴퓨터에 존재하지 않아도 되며, 재사용할 수 있게 되었다. XML과 HTTP와 같은 표준 웹 프로토콜을 사용하여 인터넷의 어떠한 사용자라도 쉽게 접근할 수 있게 되었다.

이러한 서비스 지향 기술은 이미 수년전부터 RMI, COM 그리고 CORBA와 같은 여러 다양한 형태의 기술들로 구현되어 왔으므로 최근에 발생한 완전한 새로운 형태의 것이라고 보기는 힘들다. 하지만 이렇게 언급한 기술들은 벤더나 구현된 기술에 종속적이라는 단점이 있다. 하지만 웹 서비스는 이러한 단점들을 극복한다.

다음은 웹 서비스의 주요 특징에 대한 설명이다.

- 웹을 통해 접근할 수 있다.
- 스스로 표방하고 서술한다.

웹 서비스는 스스로의 역할과 기능, 속성에 대해서 서술함에 따라서 웹 서비스 클라이언트가 서비스에 대해 이해할 수 있게 한다.

WSDL(Web Service Description Language)이라는 파일에 서비스를 서술하여 이를 공개함으로써 다른 응용 프로그램에서 서비스를 이용할 수 있게 한다.

- HTTP와 같은 표준 인터넷 프로토콜에 의해 전달되는 XML 메시지를 통해 웹 서비스 클라이언트와 교신한다. 웹 서비스 클라이언트는 응용 프로그램일 수도 있고, 다른 웹 서비스일 수도 있다.
- request-response 또는 one-way 방식으로 작동하며, 동기 또는 비동기 통신으로 호출된다. 이러한 작동

방식이나 통신 방식과는 무관하게 웹 서비스와 웹 서비스 클라이언트 간에 교환되는 근본적인 단위는 메시지이다.

웹 서비스는 다음과 같은 장점을 가지고 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 이루어지므로 웹 서비스를 지원하는 응용 프로그램들은 상호 작동에 문제가 없다.

- 플랫폼과 언어에 독립적이다.
- HTTP와 같은 웹 프로토콜을 사용하므로 방화벽과 같은 장애에도 문제가 없어 응용 프로그램에 대한 접근이 용이하다.

## 1.2. 웹 서비스 표준

J2EE(Java 2 Platform, Enterprise Edition) 스펙은 버전 1.4에 이르러 웹 서비스의 중요성이 반영되어 트랜잭션 지원이나 DB 연결, 생성 주기(Lifecycle) 관리와 같은 서비스들이 응용 프로그램의 소스 코드 수정없이 지원받을 수 있게 되었다.

Java EE(Java Enterprise Edition, J2EE에서 명칭 변경) 버전 5에서 새로운 스펙들이 추가되었으며, 이와 함께 기존 스펙의 기능들을 정리 및 향상하여 보다 성숙된 웹 서비스를 제공할 수 있게 되었다.

현재 JEUS 웹 서비스는 다음과 같은 표준을 따르고 있다.

- EWS(Implementing Enterprise Web Services) 표준
- JAX-WS(Java API for XML-based Web Services) 표준
- JAXB(Java Architecture for XML Binding) 표준
- SAAJ(SOAP with Attachments API for Java) 표준
- Streaming API for XML 표준
- Web Service Metadata for the Java Platform 표준
- XML, XML Namespace, XML Infoset, XML Schema 표준
- SOAP, MTOM, WS-Addressing 표준
- WS-ReliableMessaging, WS-Coordination, WS-AtomicTransaction 표준
- WSDL, WS-Policy, WS-MetadataExchange 표준
- WS-Security Policy, WS-Security, WS-Trust, WS-SecureConversation 표준

본 절에서는 이 중 몇 가지 주요 표준들에 대해 설명한다.

### 1.2.1. SOAP 표준

SOAP(Simple Object Access Protocol)은 분산 환경에서의 정보 교환을 목적으로 하는 경량의 XML 기반

프로토콜이다.

SOAP은 다음과 같은 정보 교환 방식을 지원한다.

- RPC(Remote Procedure Call) 방식

RPC 방식의 정보 교환은 요청-응답 프로세스를 허용하며, 서비스 End-Point는 프러시저 중심의 메시지를 받아서 적절한 응답 메시지를 작성하여 되돌려 준다.

- 메시지 기반(Message-oriented) 방식

메시지 기반 방식은 송신자가 서비스의 응답을 즉각적으로 요구하지 않아도 될 때에도 사용이 가능하며, 주로 비즈니스와 여러 가지 형태의 문서 타입을 교환할 필요가 있는 경우 사용된다. 메시지 기반 방식의 정보 교환은 문서(Document) 방식의 정보 교환이라고도 한다.

SOAP 프로토콜은 다음과 같은 요소들로 구성된다.

- SOAP 메시지를 서술하는 Envelope

Envelope는 누가 어떻게 메시지를 처리해야 하는지에 대한 정보를 가지고 있으며, 메시지의 바디를 감싸고 있다.

- 응용 프로그램에 사용되는 데이터 타입의 객체를 설명하는 인코딩 법칙(Encoding rule)
- 원격 프러시저 호출과 응답을 나타내는 데 필요한 준수 사항들

SOAP은 다음과 같은 특성을 갖는다.

- 프로토콜에 독립적이다.
- 구현 언어에 독립적이다.
- 플랫폼과 운영 시스템(Operating System)에 독립적이다.
- SOAP XML 메시지에 MIME 타입과 같은 타입으로 추가적인 메시지를 붙여서 보낼 수 있다.

## 1.2.2. WSDL 표준

WSDL(Web Service Description Language)은 웹 서비스 기술 언어라고 하며, RPC 기반과 메시지 기반의 서비스를 서술하는 XML 형태를 의미한다. WSDL 파일은 개발 툴이나 서비스 개발자 또는 배치자(서비스 제공자)에 의해서 생성되며 인터넷에 공개되면 그때부터 서비스는 다른 클라이언트에 의해 알려지게 되어 이는 클라이언트로 하여금 서비스에 접근할 수 있는 환경을 구축할 수 있게 한다.

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

## 1.2.3. UDDI 표준

UDDI(Universal Description, Discovery and Integration) 표준은 웹 서비스 관련 정보의 공개와 탐색을 위한 표준이다. 서비스 제공자는 UDDI라는 서비스 소비자에게 이미 알려진 온라인 저장소에 그들이 제공하는 서비스들을 저장하게 되고, 서비스 소비자들은 그 저장소에 접근함으로써 원하는 서비스들의 목록을 찾을 수 있다.

## 1.2.4. JAX-WS, JAXB, StAX 표준

JAX-WS(Java API for XML-based Web Services) 표준은 Sun Microsystems에서 제공하는 웹 서비스 API를 정의하는 표준이다. 이는 JAX-RPC 표준을 발전시킨 개념으로 XML의 바인딩을 위한 JAXB 표준과 표준 스트리밍 파서를 위한 StAX 표준, 기능이 향상된 새로운 SAAJ 표준을 기반으로 통합, 발전된 Java 진영의 노력의 산물이다. 이러한 API를 채택함으로 인해 J2EE 1.4 스펙에서 요구했던 많은 Descriptor 파일들을 모두 Java SE 5의 Annotation 기능으로 대체할 수 있게 되었다.

## 1.2.5. SAAJ 표준

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

## 1.3. SOAP 메시지 교환과 SOAP 메시지 인코딩

SOAP은 RPC 방식과 문서 방식의 정보 교환을 지원한다. SOAP 메시지는 RPC 방식이든 문서 방식이든 SOAP 메시지의 element에 정의되어 있는 EncodingStyle 속성에 의해 정의되는 인코딩 방식을 사용한다.

다음은 이러한 SOAP 메시지의 특성에 대한 설명이다.

- SOAP 메시지의 구성 요소

SOAP 메시지는 SOAP Envelope를 가지며 Envelope는 헤더와 바디라는 2개의 하위 요소가 있다. 헤더는 필수적으로 포함되어야 하는 것은 아니다. 헤더의 하위 요소들은 헤더 블록이며 각각의 헤더 블록은 데이터의 논리적인 묶음으로 볼 수 있다. SOAP 바디는 SOAP 메시지의 필수 요소이며 SOAP 메시지의 궁극적으로 전달하고자 하는 내용들이 담겨 있는 곳이다.

- SOAP 메시지의 인코딩

SOAP은 RPC 방식과 문서(Document) 방식의 정보 교환을 지원한다.

- RPC 방식

RPC 방식은 원격 프러시저 호출(RPC) 방식의 메시지 교환 방식을 가리키며, 클라이언트와 서버가 잘 정의된 프로그래밍 모델로 생성되어야 하며, 클라이언트는 인자를 가지는 메소드를 호출하고 서버는 응답으로 하나의 값을 반환한다. RPC 방식의 메시지 교환 방식에서는 SOAP은 메시지 바디의 형태를 별도로 정의한다.

- 문서(Document) 방식

문서 방식에서는 XML 문서가 교환되고 각각의 element의 의미는 서버와 클라이언트가 해석의 몫으로 남겨둔다. 즉, SOAP 메시지의 바디의 구조에 대한 제약 사항은 SOAP에서 규정하고 있지 않으며, 응용 프로그램이나 또는 별도로 정해진 XML 스키마에 의해 SOAP 바디에 위치한 XML 문서의 구조가 결정된다.

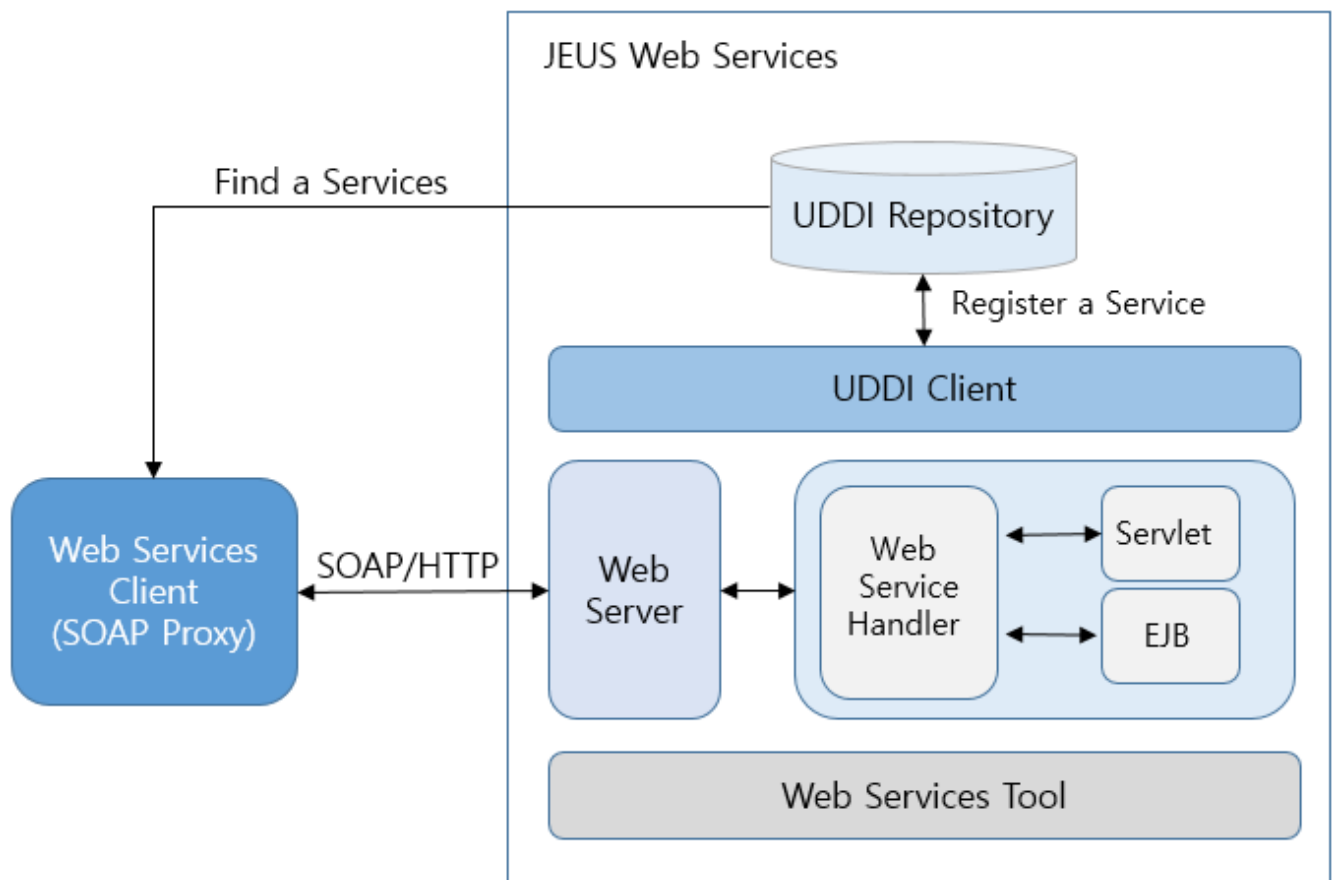
## 2. JEUS 웹 서비스

본 장에서는 JEUS 웹 서비스와 지원되는 스펙의 개념을 기술한다. 또한 JEUS 웹 서비스를 위한 설정 파일들과 툴들 그리고 시스템 변수에 대해서 살펴본다.

### 2.1. 기본 구조

Jakarta EE 8 호환 WAS(Web Application Server)인 JEUS는 Jakarta EE 8에서 요구하는 JAX-WS 웹 서비스를 지원한다. JAX-WS 웹 서비스의 가장 중요한 특징은 설정 파일들 없이 자유롭게 구현할 수 있는 POJO(Plain Old Java Object) 방식의 웹 서비스 구현이다. JEUS 웹 서비스는 Jakarta EE 8 스펙을 준수하는 벤더들과의 웹 서비스 상호 호환성이 보장된다.

JEUS 웹 서비스의 구조는 다음과 같다.



JEUS 웹 서비스의 구조

- Annotation 및 XML 설정 파일

JEUS 웹 서비스는 비즈니스 로직으로 EJB와 Java 클래스를 사용한다. EJB는 EJB 컨테이너 내부에서 실행되고 Java 클래스는 웹 컨테이너의 내부에서 실행된다.

JEUS 웹 서비스는 Java SE 5에서 지원하는 Annotation을 이용하여 웹 서비스와 관련된 정보들을 설정하는 JAX-WS 웹 서비스의 구현을 기본적으로 지원한다.

- 툴과 유틸리티

JEUS 웹 서비스에서는 WSDL 파일 및 Java 클래스의 생성 및 관리를 위해서 Command Line 툴과 Apache Ant 툴을 사용한다. 툴의 사용법은 다음 여러 장에 걸쳐서 설명한다.

Ant 툴 기능을 사용하기 위해서는 Apache Ant 1.6.1 이상을 인스톨해야 한다. <http://ant.apache.org>에서 Apache Ant를 다운로드할 수 있다.

- 시스템 환경변수

JEUS 웹 서비스를 위한 특별한 시스템 환경변수는 없고, "JEUS Web Engine 안내서"와 "JEUS EJB 안내서"에서 기술된 시스템 환경변수만으로 충분하다.

## 2.2. 웹 서비스 설계

본 절에서는 웹 서비스를 설계하는 과정에서 고려해야 할 사항에 대해서 설명한다.

### 2.2.1. 웹 서비스 Back-end 선택

웹 서비스 컴포넌트는 웹 컨테이너나 EJB 컨테이너에서 실행되는 Jakarta EE 컴포넌트이다.

웹 서비스의 Back-end는 다음의 개체들을 공개할 수 있다.

- Java 클래스 파일

Java 클래스 파일은 웹 컨테이너에서 실행된다. Java 클래스는 EJB를 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-end가 좋은 선택이다.

- Stateless Session EJB

EJB는 트랜잭션이 중요한 시스템에서 선택될 수 있는 프로그래밍 모델로 Stateless Session EJB는 EJB 컨테이너에서 실행된다.

DB에 업데이트나 삭제와 같은 작업들을 많이 하는 시스템에서는 비즈니스 로직을 EJB로 구현한다면 트랜잭션 관리에 효율적이므로 좋은 선택이라고 볼 수 있다.

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

이미 언급한 대로 기존의 비즈니스 로직은 필요에 의해 EJB(Stateless Session Bean 또는 CMP)로 구성되어 있을 수 있다. 이미 Stateless Session Bean으로 비즈니스 로직이 구현되어 있고, 이러한 상황에서 기존의 EJB 로직을 웹 서비스로 공개하고 싶다면 Stateless Session Bean 웹 서비스 Back-end를 선택하는 것이 좋다.

일반적으로 EJB 웹 서비스는 Stateless Session Bean만을 Back-end로 허용하므로 CMP는 웹 서비스와 연관이 없어 보일 수도 있다. 하지만 Stateless Session Bean이 CMP로 구현되어 있는 비즈니스 로직으로의 인터페이스 역할을 하도록 하고, Stateless Session Bean이 웹 서비스로 공개된다면 CMP 비즈니스 로직을 웹



서비스로 공개하게 된다. 기존의 CMP 비즈니스 로직의 장점을 그대로 이용하면서 웹 서비스 기능을 추가할 때 Stateless Session Bean 웹 서비스 Back-end는 좋은 선택이라 볼 수 있다.

## 2.2.2. JEUS 웹 서비스의 방식

JEUS 웹 서비스는 문서(Document) 방식을 지원한다.

XML 문서가 교환되고 각각의 element의 의미는 서버와 클라이언트가 해석의 몫으로 남겨둔다. 즉, SOAP 메시지의 바디의 구조에 대한 제약 사항은 SOAP에서 규정하고 있지 않으며, 응용 프로그램이나 혹은 별도로 정해진 XML 스키마에 의해 SOAP 바디에 위치한 XML 문서의 구조가 결정된다.

JEUS 웹 서비스에서 제공하는 문서 방식에는 표준 문서 방식과 WRAPPED 방식이 있다.

구분	설명
표준 문서 방식	각각의 웹 서비스 오퍼레이션은 1개의 인자(XML 문서)만 가질 수 있다.
WRAPPED 방식	여러 인자들을 하나의 복합 데이터 타입(complex data type)으로 포장하여 하나의 인자로 만들어 SOAP 메시지에 포함하게 되므로 웹 서비스 오퍼레이션들은 여러 개의 인자들을 가질 수 있다.

## 2.2.3. 웹 서비스 구현 방식 선택

일반적으로 웹 서비스 구현은 서비스 Endpoint로부터 시작하거나 WSDL로부터 시작할 수 있다. 이것은 개발자의 취향과 개발 환경에 따라서 선택되며 각각 장점을 가지고 있으며, JEUS 웹 서비스에서 JAX-WS 방식으로 구현될 수 있다.

- 서비스 Endpoint로부터 시작

서비스 구현이 단순하고 쉬우며 직관적으로 구현 가능하다.

- WSDL로부터 시작

구현 언어에 종립적으로 상호 운영성이 가능한 서비스를 구현하기에 적합하다.

### 서비스 Endpoint로부터 시작

서비스 Endpoint를 먼저 구현하고 이것으로부터 WSDL을 생성하는 방법이다. 이 방법은 서비스 구현을 쉽고 직관적으로 진행할 수 있는 장점이 있다. 이것은 개발자가 자신에게 익숙한 개발 환경, 즉 Java 환경에서 다른 플랫폼과의 상호 운영성에 대한 걱정 없이 서비스 Endpoint와 서비스 구현체를 개발할 수 있게 한다. 그 다음에 작성된 Java 코드로부터 플랫폼 독립적인 WSDL을 도출하게 된다.

이렇게 생성된 WSDL이 플랫폼에 독립적으로 정의되기 위해서 JAX-WS는 어떻게 서비스 Endpoint로부터 WSDL을 도출해야 하는지에 대한 지침을 제공한다. 이 방식은 Java 개발자에게는 웹 서비스 개발을 시작하는 가장 쉬운 방법이다.

JAX-WS 방식에서 유의할 사항은 서비스 설정에 관한 정보를 기술(description)하지 않아도 된다는 것이다. JAX-WS 스펙을 위해 JEUS 웹 서비스 구현에서는 별도의 서비스 설정 파일을 작성하지 않고도 wsgen 툴킷에 의해 웹

서비스를 구현할 수 있다.

JAX-WS 방식의 웹 서비스 구현에 대한 자세한 내용은 [JEUS 웹 서비스 구현](#)을 참고한다.

## WSDL로부터 시작

WSDL을 먼저 작성하고 이것으로부터 웹 서비스 구현을 생성하는 방법이다.

웹 서비스의 중심 목표 중 하나는 플랫폼 사이의 상호 운영성이다. 이러한 방법은 개발 언어에 종립적으로 상호 운영성 중심적인 서비스를 생성하기 좋은 방법이다. 웹 서비스 구현으로부터 생성된 WSDL은 플랫폼 특성이 반영된다. 즉, Java 편향된다. 그러나 WSDL로부터 시작된 웹 서비스의 기술(description)은 웹 서비스 기술에 사용된 XML 타입과 용어에서 보다 종립적이다. 즉, Java 환경에 익숙하지 않은 개발자에게도 친근할 수 있는 일반적인 명명법으로 WSDL을 작성할 수 있게 한다.

그러나 이 방식은 개발자로 하여금 WSDL에 대한 보다 깊은 이해를 요구한다. JAX-WS 웹 서비스 구현은 [JEUS 웹 서비스 구현](#)에서 설명하고 있다.

### 2.2.4. SOAP 메시지 핸들러 생성

SOAP 메시지의 헤더나 바디 또는 Attachment를 직접 다루려고 할 때 메시지 핸들러의 생성이 필요하다.

이때 JAX-WS 핸들러 프레임워크를 이용할 수 있다. 자세한 설명은 [핸들러 프레임워크](#)를 참고한다.

# Part I. JEUS 9 웹 서비스

JEUS 9 웹 서비스는 Jakarta EE 9의 웹 서비스 표준인 JAX-WS 웹 서비스를 지원한다. JAX-WS 웹 서비스의 가장 중요한 특징은 설정 파일들 없이 자유롭게 구현할 수 있는 POJO 스타일의 웹 서비스 구현이다. 이러한 POJO 스타일의 웹 서비스 구현은 Java SE 5의 Annotation의 기능에 힘입어 가능하게 되었다.

본 Part에서는 JAX-WS 웹 서비스를 구현하고 JEUS에서 구동시키는 방법, JAX-WS 웹 서비스를 호출하는 클라이언트의 작성 방법, 바인딩의 사용자화 선언, 핸들러, 메시지를 XML 수준으로 다루는 방법, 비동기, MIME Attachment로 좀 더 효율적으로 메시지를 전송하는 방법, 보다 빠른 웹 서비스의 구현에 대해 알아보고 마지막으로 XML을 다루는 다양한 방법들에 대해 설명한다.

앞으로 나오는 여러 장들을 통해 이러한 JAX-WS 웹 서비스를 JEUS 9 웹 서비스가 어떻게 지원하고 있는지 자세하게 설명할 것이다.

## 3. JEUS 웹 서비스 구현

본 장에서는 Java 클래스와 EJB, WSDL로 JEUS 웹 서비스를 구현하는 방법에 대해서 설명한다.

### 3.1. 개요

JEUS 9 웹 서비스는 Jakarta EE 9 웹 서비스 표준인 **JAX-WS 방식** 웹 서비스를 지원한다.

Jakarta EE 9 웹 서비스의 표준인 JAX-WS 웹 서비스는 JAXB(XML 문서와 Java 오브젝트 간의 데이터 바인딩을 위한 표준), SAAJ(XML SOAP 메시지를 구현 및 수정할 수 있도록 해주는 표준 API, JAX-RPC 웹 서비스 모델에서는 메시지 핸들러를 구현하기 위해 이를 직접 개발자가 사용했으나 JAX-WS 웹 서비스 모델에서는 메시지 핸들러를 위한 기본 Framework를 제공하며 SAAJ는 그 뒤에서 동작하게 된다)와 더불어 웹 서비스의 중심 컴포넌트에 해당되며 기존의 JAX-RPC 웹 서비스를 대신하기 위해 개발되었다.

JAX-RPC는 역사적으로 볼 때 JAXB가 개발되어 모습을 드러내기 전에 발표되었기 때문에 JAXB가 담당하는 데이터 바인딩 기능 또한 기본적으로 내포하고 있었다. 하지만 점점 XML에 관련된 표준들에 있어서 여러 가지 기능 추가와 함께 JAX-RPC 웹 서비스가 이를 유지보수하기 어려워지게 되었고, 데이터 바인딩에 집중하는 JAXB의 기능이 향상됨에 따라 더 이상 JAX-RPC가 데이터 바인딩 기능을 유지할 필요성이 없어지게 되었다. 이에 따라 순수 웹 서비스의 기능은 JAX-WS가 기존 JAX-RPC 웹 서비스가 함께 관여하던 데이터 바인딩 기능은 JAXB가 독립적으로 관리하게 된다. JAXB에 관해서는 뒤에서 자세히 다루도록 한다.

기본적으로 JAX-WS의 내용은 JSR 224(<http://jcp.org/jsr/detail/224.jsp>)에서 기술되어 있다. JAX-WS는 Java SE 5의 Annotation 기능에 힘입어 기존의 여러 웹 서비스 Deployment Descriptor(이하 DD) 파일들을 Annotation으로 처리, POJO(Plain Old Java Object) 방식의 웹 서비스를 구성할 수 있게 되었다.

본 장에서는 간단한 Java 클래스 파일로부터 그리고 WSDL로부터 JAX-WS 웹 서비스를 구현하는 방법에 대해 설명한다. Java 클래스로부터 웹 서비스를 구현할 때에는 서비스 Endpoint 구현 클래스만을 가지고 Annotation을 사용하여 **wsngen** 툴을 사용, Portable Artifact들을 얻는다. 그리고 WSDL로부터 웹 서비스를 구현할 때에는 작성한 WSDL 파일로부터 **wsimport** 툴을 사용하고, 서비스 Endpoint 인터페이스와 Portable Artifact들을 생성한 후 이를 구현하는 구현체 Java 클래스를 작성한다.

### 3.2. Java 클래스 웹 서비스 구현

Java Endpoint 구현 클래스로부터 웹 서비스를 구현할 때 이 Java Endpoint 구현 클래스에는 JAX-WS에 의해 약속된 다음의 몇 가지 규칙이 있다.

- jakarta.jws.WebService Annotation을 수반해야 한다.
- 메소드는 jakarta.jws.WebMethod Annotation을 수반할 수 있다.
- 모든 메소드들은 서비스 고유의 특정 Exception들과 함께 java.rmi.RemoteException을 던질 수 있다.
- 모든 메소드의 파라미터들과 반환 타입은 JAXB의 Java로부터 XML 스키마로의 매핑 정의에 명시된 것이어야 한다.
- 메소드의 파라미터 또는 반환 타입은 java.rmi.Remote 인터페이스를 직간접적으로 구현하는 것이어서는 안된다.

위의 규칙을 따른 간단한 Java Endpoint 구현 클래스의 예는 다음과 같다.

Java 클래스 웹 서비스 : <Addnumbersimpl.java>

```
package fromjava.server;

@WebService
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

위와 같이 @WebService라는 Annotation이 추가된 것을 주목하기 바란다. 이렇게 Annotation으로 'WebService'라고 설정하고 Endpoint 구현 로직을 구성하면 기본적인 서버의 웹 서비스 프로그래밍은 모두 완료된 것이다. 이렇게 Java Endpoint 구현 클래스를 구현했으면 다음으로는 JEUS 9 웹 서비스에서 제공하는 툴을 사용하여 Portable Artifact들을 생성해야 한다. 이러한 Portable Artifact들은 메소드들의 호출 또는 응답을 Java 오브젝트로 변환 또는 XML 문서로 변환하는 데 사용되는 몇 가지 Java Bean 클래스들과 서비스 고유의 특정 Exception 클래스들로 이루어진다. 이러한 Portable Artifact들을 생성하기 위해 JEUS 9 웹 서비스에서 제공해주는 툴인 **wsgen**을 사용한다.

다음은 **wsgen**의 사용법에 대한 설명이다.

```
$ wsgen -help

Usage: WSGEN [options] <SEI>

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
  -extension              allow vendor extensions - functionality
                          not specified by the specification.
                          Use of extensions may result in
                          applications that are not portable or
                          may not interoperate with other implementations
  -help                  display help
  -keep                  keep generated files
  -r <directory>         resource destination directory, specify
                          where to place resouce files such as WSDLs
  -s <directory>         specify where to place generated source files
  -verbose               output messages about what the compiler is doing
  -version               print version information
  -wsdl[:protocol]       generate a WSDL file. The protocol is optional.
                          Valid protocols are [soap1.1, Xsoap1.2],
                          the default is soap1.1.
                          The non stanadard protocols [Xsoap1.2]
                          can only be used in conjunction with the
                          -extension option.
  -servicename <name>    specify the Service name to use in the generated
                          WSDL Used in conjunction with the -wsdl option.
  -portname <name>       specify the Port name to use in the generated
                          WSDL Used in conjunction with the -wsdl option.
```

Examples:

```
wsgen -cp . example.Stock
wsgen -cp . example.Stock -wsdl
      -servicename {http://mynamespace}MyService
```



JEUS 9 웹 서비스는 wsgen의 Ant Task도 지원하는데 보다 자세한 콘솔 명령어에 대한 설명과 Ant Task 항목에 대한 설명은 JEUS Reference 안내서의 "wsgen" 콘솔 툴과 "wsgen" 플러그인의 내용을 참고한다.

wsgen 툴을 사용하여 위에서 생성한 Java Endpoint 구현 클래스를 사용하여 다음과 같이 Portable Artifact들을 생성한다.

Java 클래스 웹 서비스 : <build.xml>

```
...
<target name="build_server" depends="init">
  <antcall target="do-compile">
    <param name="javac.excludes" value="fromjava/client/" />
  </antcall>
  <antcall target="wsgen">
    <param name="sib.file" value="fromjava.server.AddNumbersImpl" />
  </antcall>
  <antcall target="do-package-war" />
</target>
...
```

wsgen 툴을 사용하여 위의 AddNumbersImpl Java Endpoint 구현 클래스로부터 Portable Artifact들과 WSDL 파일을 생성하면 그 결과는 다음과 같다.

```
AddNumbersImplService.wsdl
AddNumbersImplService_schema1.xsd
fromjava/server/jaxws/AddNumbers.class
fromjava/server/jaxws/AddNumbersResponse.class
```

AddNumbers와 AddNumbersResponse 파일들은 각각 JAXB가 addNumbers 메소드에 대한 요청과 응답을 Java 오브젝트나 XML 문서로 변환하기 위해 사용하는 Java Bean이다.

AddNumbersImplService.wsdl 파일은 이 웹 서비스의 WSDL 파일을 나타내며

AddNumbersImplService\_schema1.xsd 스키마 파일은 이 웹 서비스에 의해서 사용되는 데이터 타입을 정의한 것이고 WSDL 문서 내부에서 사용되고 있다.

다음은 각 파일의 구현된 내용이다.

Java 클래스 웹 서비스 : <AddNumbersImplService.wsdl>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions targetNamespace="http://server.fromjava/"
  name="AddNumbersImplService" xmlns:tns="http://server.fromjava/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```

xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
  <xsd:schema>
    <xsd:import namespace="http://server.fromjava/"
      schemaLocation="AddNumbersImplService_schema1.xsd" />
  </xsd:schema>
</types>
<message name="addNumbers">
  <part name="parameters" element="tns:addNumbers" />
</message>
<message name="addNumbersResponse">
  <part name="parameters" element="tns:addNumbersResponse" />
</message>
<portType name="AddNumbersImpl">
  <operation name="addNumbers">
    <input message="tns:addNumbers" />
    <output message="tns:addNumbersResponse" />
  </operation>
</portType>
<binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="addNumbers">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
<service name="AddNumbersImplService">
  <port name="AddNumbersImplPort"
    binding="tns:AddNumbersImplPortBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL" />
  </port>
</service>
</definitions>

```

위와 같이 WSDL 파일의 <message> element들에 대해 서버의 Portable Artifact들이 생성됨을 알 수 있다. 이러한 Portable Artifact들은 실제로 JAX-WS 엔진과 Endpoint 클래스 간의 메시지를 전달하는 역할을 한다. 2가지 <message> element의 Portable Artifact Java 클래스는 AddNumbers와 AddNumbersResponse이다.

다음은 클라이언트로부터 호출되는 메시지인 AddNumbers에 대한 Java Bean 클래스이다.

Java 클래스 웹 서비스 : <AddNumbers.java>

```

@XmlRootElement(name = "addNumbers", namespace = "http://server.fromjava/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbers", namespace = "http://server.fromjava/",
  propOrder = {"arg0", "arg1" })
public class AddNumbers {

  @XmlElement(name = "arg0", namespace = "")
  private int arg0;

```

```

@XmlElement(name = "arg1", namespace = "")
private int arg1;

public int getArg0() {
    return this.arg0;
}

public void setArg0(int arg0) {
    this.arg0 = arg0;
}

public int getArg1() {
    return this.arg1;
}

public void setArg1(int arg1) {
    this.arg1 = arg1;
}
}

```

다음은 서비스로부터 리턴되는 메시지인 AddNumbersResponse <message> element에 대한 Java Bean 클래스이다.

Java 클래스 웹 서비스 : <AddNumbersResponse.java>

```

@XmlRootElement(name = "addNumbersResponse", namespace = "http://server.fromjava/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbersResponse", namespace = "http://server.fromjava/")
public class AddNumbersResponse {

    @XmlElement(name = "return", namespace = "")
    private int _return;

    public int get_return() {
        return this._return;
    }

    public void set_return(int _return) {
        this._return = _return;
    }
}

```

### 3.3. EJB 웹 서비스 구현

본 절에서는 EJB 컨테이너에서 동작하는 Stateless Session Bean을 사용하여 구현하는 EJB 웹 서비스 프로그래밍 모델에 대해 설명한다.

JEUS 9에서 제공하는 EJB 4.0 프로그래밍 모델은 서비스 Endpoint 구현 클래스를 더욱 쉽게 작성할 수 있도록 여러 가지 기능들을 제공하고 있는데, jakarta.ejb.SessionBean 또는 jakarta.ejb.EntityBean 인터페이스를 더 이상 구현하지 않고 Annotation으로 명시하며 Session Bean의 경우 컴포넌트 인터페이스 또는 홈 인터페이스를 더 이상 구현하지 않아도 되는 것들이 그 중 일부이다.





JEUS 9에서 제공하는 기능에 대한 보다 더 자세한 내용은 "JEUS EJB 안내서"를 참고한다.

JEUS 9에서 제공하는 EJB 4.0 기능을 이용해서 구현한 Bean 클래스를 구현한 뒤, 이를 JAX-WS 서비스 Endpoint 구현 클래스로 사용하기 위해서는 단순히 Bean 클래스에 @WebService Annotation을 설정한다. 위의 규칙을 따른 간단한 EJB Endpoint 구현 클래스의 예는 다음과 같다.

EJB 웹 서비스 구현 : <Addnumbersimpl.java>

```
package fromejb.server;

@Stateless
@WebService
public class AddNumbersImpl {

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

위와 같이 @Stateless라는 Annotation과 @WebService라는 Annotation이 추가된 것을 주목한다. 이렇게 Annotation으로 "WebService"라고 설정하고 Endpoint 구현 로직을 구성하면 기본적인 서버의 웹 서비스 프로그래밍은 모두 완료된 것이다.

이렇게 EJB Endpoint 구현 클래스를 구현했으면 다음으로는 JEUS 9 웹 서비스에서 제공하는 툴을 사용하여 Portable Artifact들을 생성해야 한다. 이러한 Portable Artifact들은 메소드들의 호출 또는 응답을 Java 오브젝트로 변환 또는 XML 문서로 변환할 때 사용되는 몇 가지 Java Bean 클래스들과 서비스 고유의 특정 Exception 클래스들로 이루어진다. 이러한 Portable Artifact들을 생성하기 위해 JEUS 9 웹 서비스에서 제공해주는 툴인 **wsgen**을 사용한다.

다음은 **wsgen**의 사용법에 대한 설명이다.

```
$ wsgen -help

Usage: WSGEN [options] <SEI>

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
  -extension              allow vendor extensions - functionality
                        not specified by the specification.
                        Use of extensions may result in
                        applications that are not portable or
                        may not interoperate with other implementations
  -help                  display help
  -keep                  keep generated files
  -r <directory>         resource destination directory, specify
                        where to place resource files such as WSDLs
```

-s <directory>	specify where to place generated source files
-verbose	output messages about what the compiler is doing
-version	print version information
-wsdl[:protocol]	generate a WSDL file. The protocol is optional. Valid protocols are [soap1.1, Xsoap1.2], the default is soap1.1. The non standard protocols [Xsoap1.2] can only be used in conjunction with the -extension option.
-servicename <name>	specify the Service name to use in the generated WSDL Used in conjunction with the -wsdl option.
-portname <name>	specify the Port name to use in the generated WSDL Used in conjunction with the -wsdl option.

Examples:

```
wsgen -cp . example.Stock
wsgen -cp . example.Stock -wsdl
      -servicename {http://mynamespace}MyService
```



JEUS 9 웹 서비스는 wsgen의 Ant Task도 지원하는데 보다 자세한 콘솔 명령어에 대한 설명과 Ant Task 항목에 대한 설명은 JEUS Reference 안내서의 "wsgen" 콘솔 툴과 "wsgen" 플러그인의 내용을 참고한다.

wsgen 툴을 사용하여 위에서 생성한 Java Endpoint 구현 클래스를 사용해서 다음과 같이 Portable Artifact들을 생성한다.

EJB 웹 서비스 구현 : <build.xml>

```
...

<target name="build_server" depends="init">
  <antcall target="do-compile">
    <param name="javac.excludes" value="fromejb/client/" />
  </antcall>
  <antcall target="wsgen">
    <param name="sib.file" value="fromejb.server.AddNumbersImpl" />
  </antcall>
  <antcall target="do-package-jar" />
</target>

...
```

wsgen 툴을 사용하여 위의 AddNumbersImpl EJB Endpoint 구현 클래스로부터 Portable Artifact들과 WSDL 파일을 생성하면 그 결과는 다음과 같다.

```
AddNumbersImplService.wsdl
AddNumbersImplService_schema1.xsd
fromjava/server/jaxws/AddNumbers.class
fromjava/server/jaxws/AddNumbersResponse.class
```

AddNumbers와 AddNumbersResponse 파일들은 각각 JAXB가 addNumbers 메소드에 대한 요청과 응답을

Java 오브젝트나 XML 문서로 변환하기 위해 사용하는 Java Bean이다.

AddNumbersImplService.wsdl 파일은 이 웹 서비스의 WSDL 파일을 나타내며

AddNumbersImplService\_schema1.xsd 스키마 파일은 이 웹 서비스에 의해서 사용되는 데이터 타입을 정의한 것이고 WSDL 문서 내부에서 사용되고 있다.

다음은 각 파일의 구현된 내용이다.

EJB 웹 서비스 구현 : <AddNumbersImplService.wsdl>

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://server.fromejb/"
  name="AddNumbersImplService" xmlns:tns="http://server.fromejb/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://server.fromejb/"
        schemaLocation="AddNumbersImplService_schema1.xsd" />
    </xsd:schema>
  </types>
  <message name="addNumbers">
    <part name="parameters" element="tns:addNumbers" />
  </message>
  <message name="addNumbersResponse">
    <part name="parameters" element="tns:addNumbersResponse" />
  </message>
  <portType name="AddNumbersImpl">
    <operation name="addNumbers">
      <input message="tns:addNumbers" />
      <output message="tns:addNumbersResponse" />
    </operation>
  </portType>
  <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <operation name="addNumbers">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <service name="AddNumbersImplService">
    <port name="AddNumbersImplPort"
      binding="tns:AddNumbersImplPortBinding"
      soap:address location="REPLACE_WITH_ACTUAL_URL" />
  </port>
</service>
</definitions>
```

위와 같이 WSDL 파일의 <message> element들에 대해 서버의 Portable Artifact들이 생성된 것을 알 수 있다. 이러한 Portable Artifact들은 실제로 JAX-WS 엔진과 Endpoint 클래스 간의 메시지를 전달하는 역할을 한다.

2가지 <message> element의 Portable Artifact Java 클래스는 AddNumbers와 AddNumbersResponse 이다.

다음은 클라이언트로부터 호출되는 메시지인 AddNumbers에 대한 Java Bean 클래스이다.

EJB 웹 서비스 구현 : <AddNumbers.java>

```
@XmlRootElement(name = "addNumbers", namespace = "http://server.fromejb/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbers", namespace = "http://server.fromejb/", propOrder = {
    "arg0", "arg1" })
public class AddNumbers {

    @XmlElement(name = "arg0", namespace = "")
    private int arg0;

    @XmlElement(name = "arg1", namespace = "")
    private int arg1;

    public int getArg0() {
        return this.arg0;
    }

    public void setArg0(int arg0) {
        this.arg0 = arg0;
    }

    public int getArg1() {
        return this.arg1;
    }

    public void setArg1(int arg1) {
        this.arg1 = arg1;
    }
}
```

다음은 서비스로부터 리턴되는 메시지인 AddNumbersResponse <message> element에 대한 Java Bean 클래스이다.

EJB 웹 서비스 구현 : <AddNumbersResponse.java>

```
@XmlRootElement(name = "addNumbersResponse", namespace = "http://server.fromejb/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbersResponse", namespace = "http://server.fromejb/")
public class AddNumbersResponse {

    @XmlElement(name = "return", namespace = "")
    private int _return;

    public int get_return() {
        return this._return;
    }

    public void set_return(int _return) {
        this._return = _return;
    }
}
```

### 3.4. WSDL로부터 웹 서비스 구현

WSDL로부터 웹 서비스를 구현할 때에는 JEUS 9 웹 서비스에서 제공하는 툴인 **wsimport**를 사용해서 우선 서비스 Endpoint 인터페이스를 생성해야 한다.

다음은 본 절에서 사용하는 WSDL 파일의 예이다.

WSDL 웹 서비스 구현 : <AddNumbers.wsdl>

```
<?xml version="1.0" encoding="UTF-8"?>

<definitions name="AddNumbers" targetNamespace="urn:AddNumbers"
  xmlns:impl="urn:AddNumbers"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <types>
    <xsd:schema elementFormDefault="qualified"
      targetNamespace="urn:AddNumbers"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="addNumbersResponse">
        <sequence>
          <element name="return" type="xsd:int" />
        </sequence>
      </complexType>
      <element name="addNumbersResponse"
        type="impl:addNumbersResponse" />
      <complexType name="addNumbers">
        <sequence>
          <element name="arg0" type="xsd:int" />
          <element name="arg1" type="xsd:int" />
        </sequence>
      </complexType>
      <element name="addNumbers" type="impl:addNumbers" />
    </xsd:schema>
  </types>

  <message name="addNumbers">
    <part name="parameters" element="impl:addNumbers" />
  </message>
  <message name="addNumbersResponse">
    <part name="result" element="impl:addNumbersResponse" />
  </message>

  <portType name="AddNumbersPortType">
    <operation name="addNumbers">
      <input message="impl:addNumbers" name="add" />
      <output message="impl:addNumbersResponse"
        name="addResponse" />
    </operation>
  </portType>

  <binding name="AddNumbersBinding"
    type="impl:AddNumbersPortType">
```

```

<soap:binding
  transport="http://schemas.xmlsoap.org/soap/http"
  style="document" />
<operation name="addNumbers">
  <soap:operation soapAction="" />
  <input>
    <soap:body use="literal" />
  </input>
  <output>
    <soap:body use="literal" />
  </output>
</operation>
</binding>

<service name="AddNumbersService">
  <port name="AddNumbersPort" binding="impl:AddNumbersBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL" />
  </port>
</service>
</definitions>

```

이와 같이 WSDL 파일을 구현했으면 다음으로는 JEUS 9 웹 서비스에서 제공하는 **wsimport** 툴을 사용하여 서비스 Endpoint 인터페이스와 Portable Artifact들을 생성해야 한다.

다음은 **wsimport**의 사용법에 대한 설명이다.

```
$ wsimport -help
```

Usage: wsimport [options] <WSDL\_URI>

where [options] include:

-b <path>	specify jaxws/jaxb binding files or additional schemas (Each <path> must have its own -b)
-B<jaxbOption>	Pass this option to JAXB schema compiler
-catalog <file>	specify catalog file to resolve external entity references supports TR9401, XCatalog, and OASIS XML Catalog format.
-d <directory>	specify where to place generated output files
-extension	allow vendor extensions - functionality not specified by the specification. Use of extensions may result in applications that are not portable or may not interoperate with other implementations
-help	display help
-httpproxy:<host>:<port>	specify a HTTP proxy server (port defaultsto 8080)
-keep	keep generated files
-p <pkg>	specifies the target package
-quiet	suppress wsimport output
-s <directory>	specify where to place generated source files
-target <version>	enerate code as per the given JAXWS spec version e.g. 2.0 will generate compliant code for JAXWS 2.0 spec
-verbose	output messages about what the compiler is doing
-version	print version information
-wsdllocation <location>	@WebServiceClient.wsdlLocation value

Examples:

```
wsimport stock.wsdl -b stock.xml -b stock.xjb
```

```
wsimport -d generated http://example.org/stock?wsdl
```



JEUS 9 웹 서비스는 wsimport의 Ant Task도 지원하는데, 보다 자세한 콘솔 명령어에 대한 설명과 Ant Task 항목에 대한 설명은 JEUS Reference 안내서의 "wsimport" 콘솔 툴과 "wsimport" 플러그인의 내용을 참고한다.

wsimport 툴을 사용하여 위에서 생성한 WSDL 파일로 다음과 같이 서비스 Endpoint 인터페이스와 Portable Artifact들을 생성한다.

WSDL 웹 서비스 구현 : <build.xml>

```
...

<target name="build_server" depends="init">
  <mkdir dir="${build.classes.dir}" />
  <antcall target="wsimport">
    <param name="package.name" value="fromwsdl.server" />
    <param name="binding.file" value="" />
    <param name="wsdl.file" value="${src.web}/WEB-INF/wsdl/AddNumbers.wsdl" />
  </antcall>
  <antcall target="do-compile">
    <param name="javac.excludes" value="fromwsdl/client/" />
  </antcall>
  <antcall target="do-package-war" />
</target>

...
```

위의 wsimport Ant Task의 결과로 생성되는 서비스 Endpoint 인터페이스 및 Portable Artifact들은 다음과 같다.

```
fromwsdl/server/AddNumbers.class
fromwsdl/server/AddNumbersPortType.class
fromwsdl/server/AddNumbersResponse.class
fromwsdl/server/AddNumbersService.class
fromwsdl/server/ObjectFactory.class
fromwsdl/server/package-info.class
```

AddNumbers와 AddNumbersResponse 파일들은 각각 JAXB가 addNumbers 메소드에 대한 요청과 응답을 Java 오브젝트나 XML 문서로 변환하기 위해 사용하는 Java Bean이다.

AddNumbersPortType 파일은 서비스 Endpoint 인터페이스를 나타내며, AddNumbersService 파일은 클라이언트에서 Proxy의 용도로 사용하는 Java 클래스이다. 그리고 나머지 ObjectFactory 클래스와 package-info 클래스는 JAXB가 생성한 파일들이다.

이러한 각각의 파일들의 내용은 다음과 같다. 먼저 **wsimport** 툴을 사용하여 위의 WSDL 파일로부터 얻은 서비스 Endpoint 인터페이스인 AddNumbersPortType 클래스를 구현한다.

WSDL 웹 서비스 구현 : <AddNumbersPortType.java>

```
@WebService(name = "AddNumbersPortType", targetNamespace = "urn:AddNumbers")
```

```

@XmlSeeAlso( { ObjectFactory.class })
public interface AddNumbersPortType {
    @WebMethod
    @WebResult(targetNamespace = "urn:AddNumbers")
    @RequestWrapper(localName = "addNumbers",
        targetNamespace = "urn:AddNumbers",
        className = "fromwsdl.server.AddNumbers")
    @ResponseWrapper(localName = "addNumbersResponse",
        targetNamespace = "urn:AddNumbers",
        className = "fromwsdl.server.AddNumbersResponse")
    public int addNumbers(
        @WebParam(name = "arg0", targetNamespace = "urn:AddNumbers")
        int arg0,
        @WebParam(name = "arg1", targetNamespace = "urn:AddNumbers")
        int arg1);
}

```

위와 같이 생성된 서비스 Endpoint 인터페이스는 동적 바인딩(Dynamic binding) 또는 런타임에 필요한 Annotation들을 가지고 있다. 이러한 Annotation들은 XML 문서에서 Java 오브젝트 또는 Java 오브젝트에서 XML 문서로의 변환을 위해 필요하다.



이 서비스 Endpoint 인터페이스는 다시 WSDL과 스키마 파일들을 생성하는 데 사용될 수 있다. 그러나 이렇게 생성된 WSDL과 스키마 파일은 그 서비스 Endpoint 인터페이스를 얻기 위해 사용된 본래의 WSDL과 스키마 파일과 일치하지 않을 수 있다.

또한 JAXB가 addNumbers 메소드에 대한 요청과 응답을 Java 오브젝트나 XML 문서로 변환하기 위해 사용하는 Java Bean 클래스인 AddNumbers 클래스와 AddNumbersResponse 클래스는 다음과 같다.

WSDL 웹 서비스 구현 : <AddNumbers.java>

```

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbers", propOrder = { "arg0", "arg1" })
public class AddNumbers {

    protected int arg0;

    protected int arg1;

    public int getArg0() {
        return arg0;
    }
    public void setArg0(int value) {
        this.arg0 = value;
    }
    public int getArg1() {
        return arg1;
    }
    public void setArg1(int value) {
        this.arg1 = value;
    }
}

```



WSDL 웹 서비스 구현 : <AddNumbersResponse.java>

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbersResponse", propOrder = { "_return" })
public class AddNumbersResponse {

    @XmlElement(name = "return")
    protected int _return;

    public int getReturn() {
        return _return;
    }
    public void setReturn(int value) {
        this._return = value;
    }
}
```

이 2개의 Java Bean 클래스는 위에서 작성한 WSDL 파일의 <message> element들인 AddNumbers, AddNumbersResponse에 해당하는 것을 알 수 있다.

다음으로는 이렇게 WSDL로부터 생성된 서비스 Endpoint 인터페이스와 Portable Artifact들 중 실제로 서비스에 관한 비즈니스 로직을 담고 있는 서비스 Endpoint 구현체 클래스를 작성하는 일이다. 이 구현체 Java 클래스를 생성하기 위해서는 우선 서비스 Endpoint 인터페이스의 이름을 명시하는 Annotation을 이 구현체 Java 클래스에 제공해야 한다. 또한 작성한 WSDL 파일의 위치와 WSDL 파일의 name space(Target Name Space), 서비스 이름, 포트 이름을 설정한다.

WSDL 웹 서비스 구현 : <AddNumbersImpl.java>

```
package fromwsdl.server;

@jakarta.jws.WebService(
    endpointInterface = "fromwsdl.server.AddNumbersPortType",
    wsdlLocation="WEB-INF/wsdl/AddNumbers.wsdl",
    targetNamespace = "urn:AddNumbers",
    serviceName = "AddNumbersService",
    portName = "AddNumbersPort"
)
public class AddNumbersImpl {
    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

위와 같이 구현체 Java 클래스에 서비스 Endpoint 인터페이스의 이름을 @WebService라는 Annotation에 endpointInterface, wsdlLocation, targetNamespace, serviceName, portName이라는 변수들로 값을 지정해주는 것을 알 수 있다.

## 4. 웹 서비스 생성과 배치

본 장에서는 JAX-WS 웹 서비스를 Java 클래스와 WSDL 파일로부터 생성하고 deploy하여 패키징하는 방법에 대해 설명한다. 또한 JAX-WS 웹 서비스를 JEUS 9에서 동작시키는 방법에 대해 설명한다.

### 4.1. 개요

기본적으로 웹 서비스를 Java 클래스로부터 구현하든 WSDL로부터 구현하든 배치하는 과정은 거의 동일하다. JAX-WS 웹 서비스는 POJO 방식의 웹 서비스 구현을 지향하므로 이러한 DD의 개수는 JAX-RPC 웹 서비스에 비해 대폭적으로 줄어든다.

JEUS 5 JAX-RPC 방식 웹 서비스의 구현에서 일반적으로 사용되었던 DD 없이 프로그래밍할 수 있고, 이것은 웹 서비스 개발자가 매우 쉽고 빠르고 편리하게 웹 서비스를 개발하도록 도와준다(Description-free programming).

### 4.2. Java 클래스 웹 서비스 생성과 배치

Java 클래스로부터 구현한 웹 서비스를 WAR 파일로 구성하는 것은 서비스 Endpoint 구현 클래스와 그 밖의 Java 클래스들(Portable Artifact들)을 여러 DD 파일과 함께 WAR 포맷으로 묶는 것을 의미한다.

다음은 이전 장에서 Java 클래스로부터 구현한 웹 서비스를 WAR 형태로 구성하는 모습을 디렉터리별로 나타낸 것이다.

```
META-INF/MANIFEST.MF
WEB-INF/classes/AddNumbersImplService_schema1.xsd
WEB-INF/classes/AddNumbersImplService.wsdl
WEB-INF/classes/fromjava/server/AddNumbersImpl.class
WEB-INF/classes/fromjava/server/jaxws/AddNumbers.class
WEB-INF/classes/fromjava/server/jaxws/AddNumbersResponse.class
```

위의 WAR 파일은 서비스 Endpoint 구현 클래스와 Portable Artifact 클래스들과 함께 유일한 DD인 web.xml로 구성된다. 사실 JAX-WS 웹 서비스의 구현에는 web.xml 파일조차 필요가 없으나 여기서는 원하는 Endpoint 주소를 얻기 위해 web.xml을 사용하였다. Endpoint 주소에 대한 보다 자세한 사항은 [Endpoint 주소 결정 방식](#)을 참고한다.

이와 같이 생성된 웹 서비스 WAR 파일은 JEUS 9에서 제공하는 방식으로 deploy되어 접근할 수 있다. 생성된 WAR 파일을 JEUS 9에 deploy하기 위해 콘솔에서 다음의 명령을 실행한다.

```
$ ant build deploy
```

정상적으로 deploy되면 실제 이 서비스에 접근할 수 있는 실제 주소는 다음과 같다.

```
http://host:port/AddNumbers/AddNumbersImplService
```

### 4.3. EJB 웹 서비스 생성과 배치

Java 클래스로부터 구현한 웹 서비스를 JAR 파일로 구성하는 것은 서비스 Endpoint 구현 클래스와 그 밖의 Java 클래스(Portable Artifact)들을 여러 DD 파일과 함께 JAR 포맷으로 묶는 것을 의미한다.

다음은 이전 장에서 EJB Stateless Session Bean으로부터 구현한 웹 서비스를 JAR 형태로 구성하는 모습을 디렉터리별로 나타낸 것이다.

```
META-INF/MANIFEST.MF
fromejb/server/AddNumbersImpl.class
fromejb/server/jaxws/AddNumbers.class
fromejb/server/jaxws/AddNumbersResponse.class
```

위의 JAR 파일은 서비스 Endpoint 구현 클래스와 Portable Artifact 클래스들로 이루어진다. Endpoint 주소에 대한 보다 자세한 사항은 [Endpoint 주소 결정 방식](#)을 참고한다.

이와 같이 생성된 웹 서비스 JAR 파일은 JEUS 9에서 제공하는 방식으로 deploy되어 접근할 수 있다. 생성된 JAR 파일을 JEUS 9에 deploy하기 위해 콘솔에서 다음과 같은 명령을 실행한다.

```
$ ant build deploy
```

정상적으로 deploy되면 이 서비스에 접근할 수 있는 실제 주소는 다음과 같다.

```
http://host:port/AddNumbersImplService/AddNumbersImpl
```

### 4.4. WSDL 웹 서비스 생성과 배치

WSDL로부터 구현한 웹 서비스를 WAR 파일을 구성하는 것은 서비스 Endpoint 인터페이스와 그것을 구현한 서비스 Endpoint 클래스 그리고 그 밖의 Portable Artifact들을 여러 DD 파일과 함께 WAR 포맷으로 묶는 것을 의미한다.

다음은 이전 장에서 WSDL로부터 구현한 웹 서비스를 WAR 형태로 구성하는 모습을 디렉터리별로 나타낸 것이다.

```
META-INF/MANIFEST.MF
WEB-INF/wsdl/AddNumbers.wsdl
WEB-INF/classes/fromjava/server/AddNumbers.class
WEB-INF/classes/fromjava/server/AddNumbersImpl.class
WEB-INF/classes/fromjava/server/AddNumbersPortType.class
WEB-INF/classes/fromjava/server/AddNumbersResponse.class
WEB-INF/classes/fromjava/server/AddNumbersService.class
WEB-INF/classes/fromjava/server/ObjectFactory.class
WEB-INF/classes/fromjava/server/package-info.class
```

위의 WAR 파일은 서비스 Endpoint 인터페이스인 AddNumbersPortType과 이를 구현하는 AddNumbersImpl 그리고 여러 가지 Portable Artifact 클래스들과 함께 유일한 DD인 web.xml 파일들로 구성된다. 사실 JAX-WS 웹 서비스의 구현에는 web.xml 파일조차 필요가 없으나 여기서는 원하는 Endpoint 주소를 얻기 위해 web.xml을

사용하였다. Endpoint 주소에 대한 보다 자세한 사항은 [Endpoint 주소 결정 방식](#)을 참고한다.

이와 같이 생성된 웹 서비스 WAR 파일은 JEUS 9에서 제공하는 방식으로 deploy되어 웹 서비스로 공개될 수 있다. 생성된 WAR 파일을 JEUS 9에 deploy하기 위해 콘솔에서 다음과 같은 명령을 실행한다.

```
$ ant build deploy
```

정상적으로 deploy되면 실제 이 서비스에 접근할 수 있는 실제 주소는 다음과 같다.

```
http://host:port/AddNumbers/AddNumbersService
```

## 4.5. Endpoint 주소 결정 방식

JAX-WS 웹 서비스는 web.xml을 비롯한 webservicex.xml 등의 기존 JEUS 5의 JAX-RPC 웹 서비스에서 필요했던 DD 파일이 없어도(Descriptor-free) JEUS 9에 deploy가 가능하다.

본 절에서는 서블릿 기반의 웹 서비스와 EJB 기반의 웹 서비스에 대해 이러한 JAX-WS가 JEUS 9에 deploy될 때 실제로 이 서비스에 접근할 수 있는 각각의 주소 결정 방식을 설명한다.

### 4.5.1. 서블릿 Endpoint

서블릿 Endpoint의 경우 URL 결정 우선순위는 다음과 같다.

#### 1. web.xml 파일의 <url-pattern>을 사용하는 경우

web.xml 파일의 <url-pattern>을 사용하는 경우 입력된 값이 실제 이 웹 서비스에 접근할 수 있는 URL 주소이다. 이는 @EndpointDescription Endpoint 주소의 기본값을 overwrite한다.

다음은 web.xml 파일과 AddNumbersImpl과 같은 Endpoint 클래스의 예이다.

web.xml 파일의 <url-pattern> 사용 : <web.xml>

```
<web-app>
  <display-name>fromwsdl</display-name>
  <description>fromwsdl</description>
  <servlet>
    <servlet-name>fromwsdl</servlet-name>
    <display-name>fromwsdl</display-name>
    <servlet-class>fromwsdl.server.AddNumbersImpl</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>fromwsdl</servlet-name>
    <url-pattern>/addnumbers</url-pattern>
  </servlet-mapping>
</web-app>
```

Endpoint 클래스 : <AddNumbersImpl.java>

```
@WebService(serviceName="AddNumbers")
@EndpointDescription(endpointUrl="MyService")
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

이 웹 서비스에 접근할 수 있는 실제 URL 주소는 다음과 같다.

```
http://server:port/context/addnumbers
```

## 2. @EndpointDescription Annotation을 사용하는 경우

@EndpointDescription라는 Annotation에 endpointUrl이라는 변수값이 정해져 있을 경우에는 이 값이 실제 이 웹 서비스에 접근할 수 있는 URL 주소이다. 이는 @WebService Annotation의 serviceName Endpoint 주소값을 overwrite한다.

다음은 Endpoint 클래스의 예이다.

@EndpointDescription Annotation 사용 : <AddNumbersImpl.java>

```
@WebService(serviceName="AddNumbers")
@EndpointDescription(endpointUrl="MyService")
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

이 웹 서비스에 접근할 수 있는 실제 URL 주소는 다음과 같다.

```
http://server:port/context/MyService
```

## 3. @WebService Annotation의 serviceName 속성을 사용하는 경우

@WebService라는 Annotation에 serviceName이라는 변수값이 정해져 있을 경우에는 이 값이 실제 이 웹 서비스에 접근할 수 있는 URL 주소이다. 이는 Endpoint 주소의 기본값을 overwrite한다.

다음은 Endpoint 클래스의 예이다.

@WebService Annotation의 serviceName 속성 사용 : <AddNumbersImpl.java>

```
@WebService(serviceName="AddNumbers")
public class AddNumbersImpl {
```

```

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}

```

이 웹 서비스에 접근할 수 있는 실제 URL 주소는 다음과 같다.

```
http://server:port/context/AddNumbers
```

#### 4. Endpoint 주소의 기본값 + "Service" 사용

@WebService라는 Annotation 이외에 아무런 설정이 없다면 기본값으로 'Endpoint 클래스 이름 + Service'가 이 웹 서비스의 Endpoint 주소의 기본값이다.

다음은 Endpoint 클래스의 예이다.

Endpoint 클래스 : <AddNumbersImpl.java>

```

@WebService
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}

```

이 웹 서비스에 접근할 수 있는 실제 URL 주소는 다음과 같다.

```
http://server:port/context/AddNumbersImplService
```

### 4.5.2. EJB Endpoint

EJB Endpoint의 경우 Endpoint URL 결정 우선순위는 서블릿 Endpoint의 경우와 같고, 컨텍스트의 결정 방식만 다르다.

컨텍스트의 결정 우선순위는 다음과 같다.

#### 1. @EndpointDescription Annotation을 사용하는 경우

- Endpoint

@EndpointDescription라는 Annotation에 endpointUrl이라는 변수값이 정해져 있을 경우에는 이 값이 실제 이 웹 서비스에 접근할 수 있는 URL 주소이다. 이는 @WebService Annotation의 name Endpoint 주소값을 overwrite한다.

다음은 Endpoint 클래스의 예이다.

@EndpointDescription Annotation 사용 (1) : <AddNumbersImpl.java>

```
@WebService(name="AddNumbersService")
@EndpointDescription(endpointUrl="MyService")
@Stateless
public class AddNumbersImpl {

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

이 웹 서비스에 접근할 수 있는 실제 URL 주소는 다음과 같다.

```
http://server:port/context/MyService
```

- Context

@jeus.webservices.annotation.EndpointDescription Annotation에 contextPath 속성을 사용하는 경우 컨텍스트 기본값을 overwrite한다.

다음은 Endpoint 클래스에 대한 예이다.

@EndpointDescription Annotation 사용 (2) : <AddNumbersImpl.java>

```
@WebService(name="Hello", serviceName="AddNumbersService")
@jeus.webservices.annotation.EndpointDescription(contextPath="EJBService",
    endpointUrl="MyEndpoint")
@Stateless
public class AddNumbersImpl {

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

이 웹 서비스에 접근할 수 있는 실제 URL 주소는 다음과 같다.

```
http://server:port/EJBService/MyEndpoint
```

## 2. @WebService Annotation의 serviceName 속성을 사용하는 경우

- Endpoint

@WebService라는 Annotation에 name이라는 변수값이 정해져 있을 경우에는 이 값이 실제 이 웹 서비스에 접근할 수 있는 URL 주소이다. 이는 Endpoint 주소의 기본값을 overwrite한다.

다음은 Endpoint 클래스의 예이다.

@WebService Annotation의 serviceName 속성 사용 (1) : <AddNumbersImpl.java>

```
@WebService(name="AddNumbersService")
@Stateless
public class AddNumbersImpl {

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

이 웹 서비스에 접근할 수 있는 실제 URL 주소는 다음과 같다.

```
http://server:port/context/AddNumbersService
```

#### ◦ Context

@WebService Annotation에 serviceName 속성을 사용하는 경우 컨텍스트 기본값을 overwrite한다.

다음은 Endpoint 클래스의 예이다.

@WebService Annotation의 serviceName 속성 사용 (2) : <AddNumbersImpl.java>

```
@WebService(name="Hello", serviceName="AddNumbersService")
@Stateless
public class AddNumbersImpl {

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

이 웹 서비스에 접근할 수 있는 실제 URL 주소는 다음과 같다.

```
http://server:port/AddNumbersService/Hello
```

### 3. 컨텍스트의 기본값



- Endpoint

@WebService라는 Annotation 이외에 아무런 설정이 없다면 기본값으로 Endpoint 클래스 이름이 이 웹 서비스의 Endpoint 주소의 기본값이다.

다음은 Endpoint 클래스의 예이다.

컨텍스트의 기본값 사용 Endpoint 클래스 : <AddNumbersImpl.java>

```
@WebService
@Stateless
public class AddNumbersImpl {

    public AddNumbersImpl() {

    }

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

이 웹 서비스에 접근할 수 있는 실제 URL 주소는 다음과 같다.

```
http://server:port/context/AddNumbersImpl
```

- Context

컨텍스트(context) 기본값은 그 웹 서비스의 WSDL 문서의 ServiceName이다. 또한, 이 ServiceName은 Endpoint 클래스 이름에 Service를 붙인 값이 기본값으로 설정된다.

따라서 위의 Endpoint 클래스의 웹 서비스에 대해 다음과 같은 주소로 접근할 수 있다.

```
http://server:port/AddNumbersImplService/AddNumbersImpl
```

## 5. 웹 서비스 호출

본 장에서는 웹 서비스 호출 방식에 대해서 알아보고 예제를 통해 실제 구현 방식을 설명한다.

### 5.1. 개요

하나의 JAX-WS 웹 서비스 클라이언트 애플리케이션은 원격의 웹 서비스 Endpoint에 다음의 2가지 방식으로 접근할 수 있다.

- 동적 프록시(Dynamic Proxy) 방식
- 디스패치(Dispatch) 방식

### 5.2. 동적 프록시 방식의 웹 서비스 호출

동적 프록시(Dynamic Proxy) 방식의 웹 서비스 호출은 크게 Java SE 방식과 Jakarta EE 방식으로 나뉘어진다. 이 2가지 방식은 모두 공통적으로 Client Artifact들을 생성하는 것을 전제로 하고 있다.

본 절에서는 이러한 Client Artifact들의 생성 방법에 대해 설명하고, Java SE 방식 그리고 Jakarta EE 방식의 웹 서비스 호출에 대해 자세히 설명한다.

#### 5.2.1. Client Artifact 생성

웹 서비스의 클라이언트 호출은 그 서비스가 Java 클래스로부터 생성되든 WSDL 파일로부터 생성되든 결국 그 서비스가 Publish하는 WSDL 파일을 이용하여 Client Artifact들을 생성한 후 호출하는 Java 클래스의 작성을 통해 이루어진다. 따라서 본 절에서는 간단히 앞 장에서 Java 클래스를 통해 생성한 웹 서비스를 호출하는 클라이언트를 위한 Client Artifact들을 생성해 본다.

웹 서비스의 Client Artifact(프록시)들은 그 서비스가 제공하는 WSDL로부터 JEUS 9 웹 서비스에서 제공하는 툴인 **wsimport**를 통해 얻어진 서비스 인터페이스 클래스들과 서비스 Endpoint 인터페이스 클래스로 이루어진다.

콘솔에서 wsimport를 이용해서 Client Artifact들을 얻는 방법은 다음과 같다.

```
//db-font-size 88% $ >wsimport -help
Usage: wsimport [options] <WSDL_URI>

where [options] include:
  -b <path>                specify jaxws/jaxb binding files or additional schemas
                           (Each <path> must have its own -b)
  -B<jaxbOption>            Pass this option to JAXB schema compiler
  -catalog <file>           specify catalog file to resolve external
                           entity references supports TR9401,
                           XCatalog, and OASIS XML Catalog format.
  -d <directory>           specify where to place generated output files
  -extension                allow vendor extensions - functionality
                           not specified by the specification. Use
                           of extensions may result in applications
```

	that are not portable or may not interoperate with other implementations
-help	display help
-httpproxy:<host>:<port>	specify a HTTP proxy server (port defaults to 8080)
-keep	keep generated files
-p <pkg>	specifies the target package
-quiet	suppress wsimport output
-s <directory>	specify where to place generated source files
-target <version>	generate code as per the given JAXWS spec version e.g. 2.0 will generate compliant code for JAXWS 2.0 spec
-verbose	output messages about what the compiler is doing
-version	print version information
-wsdllocation <location>	@WebServiceClient.wsdlLocation value

Examples:

```
wsimport stock.wsdl -b stock.xml -b stock.xjb
wsimport -d generated http://example.org/stock?wsdl
```



JEUS 9 웹 서비스는 wsimport의 Ant Task도 지원하는데 보다 자세한 콘솔 명령어에 대한 설명과 Ant Task 항목에 대한 설명은 JEUS Reference 안내서의 "wsimport" 콘솔 톨과 "wsimport" 플러그인의 내용을 참고한다.

다음과 같은 원격 웹 서비스의 WSDL 파일로 그 아래의 wsimport의 Ant Task를 사용하여 Portable Artifact들을 생성한다. (<http://host:port/AddNumbers/AddNumbersImplService?wsdl>)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions targetNamespace="http://server.fromjava/"
  name="AddNumbersImplService" xmlns:tns="http://server.fromjava/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://server.fromjava/"
        schemaLocation="AddNumbersImplService_schema1.xsd" />
    </xsd:schema>
  </types>
  <message name="addNumbers">
    <part name="parameters" element="tns:addNumbers" />
  </message>
  <message name="addNumbersResponse">
    <part name="parameters" element="tns:addNumbersResponse" />
  </message>
  <portType name="AddNumbersImpl">
    <operation name="addNumbers">
      <input message="tns:addNumbers" />
      <output message="tns:addNumbersResponse" />
    </operation>
  </portType>
  <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <operation name="addNumbers">
      <soap:operation soapAction="" />
    </operation>
  </binding>
</definitions>
```

```

        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
        </output>
    </operation>
</binding>
<service name="AddNumbersImplService">
    <port name="AddNumbersImplPort"
        binding="tns:AddNumbersImplPortBinding">
        <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
</service>
</definitions>

```

Client Artifact 생성 : <build.xml>

```

...
<target name="build_client" depends="do-deploy-success, init">
    <antcall target="wsimport">
        <param name="package.name" value="fromjava.client" />
        <param name="binding.file" value="" />
        <param name="wsdl.file"
            value="http://localhost:8088/AddNumbers/AddNumbersImplService?wsdl" />
    </antcall>
    <antcall target="do-compile">
        <param name="javac.excludes" value="fromjava/server/" />
    </antcall>
</target>
...

```

위의 wsimport Ant Task를 가지고 호출할 웹 서비스의 WSDL 파일을 사용하여 Portable Artifact들을 생성하면 그 결과로 생성되는 파일들은 다음과 같다.

```

fromwsdl/client/AddNumbers.class
fromwsdl/client/AddNumbersImpl.class
fromwsdl/client/AddNumbersImplService.class
fromwsdl/client/AddNumbersResponse.class
fromwsdl/client/ObjectFactory.class
fromwsdl/client/package-info.class

```

AddNumbers와 AddNumbersResponse 파일들은 각각 JAXB가 addNumbers 메소드에 대한 요청과 응답을 Java 오브젝트로 혹은 XML 문서로 변환하기 위해 사용하는 Java Bean이다.

AddNumbersImpl 파일은 서비스 Endpoint 인터페이스를 나타내며 AddNumbersImplService 파일은 클라이언트에서 프록시의 용도로 사용하는 서비스 인터페이스의 Java 클래스이다. 그리고 나머지 ObjectFactory 클래스와 package-info 클래스는 JAXB가 생성한 파일들이다.

다음은 wsimport 툴을 사용하여 위의 원격 웹 서비스의 WSDL 파일로부터 얻은 서비스 인터페이스 클래스인 AddNumbersImplService 클래스이다.

```
//db-font-size 87% @WebServiceClient(name = "AddNumbersImplService",
    targetNamespace = "http://server.fromjava/",
    wsdlLocation = "http://localhost:8088/AddNumbers/AddNumbersImplService?wsdl")
public class AddNumbersImplService extends Service {

    private final static URL ADDNUMBERSIMPLSERVICE_WSDL_LOCATION;
    private final static WebServiceException ADDNUMBERSIMPLSERVICE_EXCEPTION;
    private final static QName ADDNUMBERSIMPLSERVICE_QNAME = new QName("http://server.fromjava/",
        "AddNumbersImplService");

    static {
        URL url = null;
        WebServiceException e = null;
        try {
            url = new URL(
                "http://localhost:8088/AddNumbers/AddNumbersImplService?wsdl");
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        ADDNUMBERSIMPLSERVICE_WSDL_LOCATION = url;
        ADDNUMBERSIMPLSERVICE_EXCEPTION = e;
    }

    public AddNumbersImplService() {
        super(__getWsdlLocation(), ADDNUMBERSIMPLSERVICE_QNAME);
    }

    public AddNumbersImplService(WebServiceFeature... features) {
        super(__getWsdlLocation(), ADDNUMBERSIMPLSERVICE_QNAME, features);
    }

    public AddNumbersImplService(URL wsdlLocation) {
        super(wsdlLocation, ADDNUMBERSIMPLSERVICE_QNAME);
    }

    public AddNumbersImplService(URL wsdlLocation, WebServiceFeature... features) {
        super(wsdlLocation, ADDNUMBERSIMPLSERVICE_QNAME, features);
    }

    public AddNumbersImplService(URL wsdlLocation, QName serviceName) {
        super(wsdlLocation, serviceName);
    }

    public AddNumbersImplService(URL wsdlLocation, QName serviceName,
        WebServiceFeature... features) {
        super(wsdlLocation, serviceName, features);
    }

    @WebEndpoint(name = "AddNumbersImplPort")
    public AddNumbersImpl getAddNumbersImplPort() {
        return super.getPort(new QName("http://server.fromjava/", "AddNumbersImplPort"),
            AddNumbersImpl.class);
    }

    @WebEndpoint(name = "AddNumbersImplPort")
    public AddNumbersImpl getAddNumbersImplPort(WebServiceFeature... features) {
        return super.getPort(new QName("http://server.fromjava/", "AddNumbersImplPort"),

```

```

        AddNumbersImpl.class, features);
    }

    private static URL __getWsdLocation() {
        if (ADDNUMBERSIMPLSERVICE_EXCEPTION!= null) {
            throw ADDNUMBERSIMPLSERVICE_EXCEPTION;
        }
        return ADDNUMBERSIMPLSERVICE_WSDL_LOCATION;
    }
}

```

위의 서비스 인터페이스는 클라이언트에서 실제 프록시 객체를 얻을 때 사용한다.

클라이언트에서는 위의 클래스에서 AddNumbersImplService 생성자를 통해 얻은 AddNumbersImplService 객체로부터 getAddNumbersImplPort() 메소드를 통해 서비스 Endpoint 인터페이스인 AddNumbersImpl을 얻을 수 있다.

다음은 wsimport 툴을 사용하여 위의 원격 웹 서비스의 WSDL 파일로부터 얻은 서비스 Endpoint 인터페이스인 AddNumbersImpl 클래스이다.

서비스 Endpoint 인터페이스 : <AddNumbersImpl.java>

```

@WebService(name = "AddNumbersImpl", targetNamespace = "http://server.fromjava/")
@XmlSeeAlso( { ObjectFactory.class })
public interface AddNumbersImpl {

    @WebMethod
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "addNumbers",
        targetNamespace = "http://server.fromjava/",
        className = "fromjava.client.AddNumbers")
    @ResponseWrapper(localName = "addNumbersResponse",
        targetNamespace = "http://server.fromjava/",
        className = "fromjava.client.AddNumbersResponse")
    public int addNumbers(@WebParam(name = "arg0", targetNamespace = "")
        int arg0, @WebParam(name = "arg1", targetNamespace = "")
        int arg1);
}

```

위와 같이 생성된 서비스 Endpoint 인터페이스는 Dynamic Binding 혹은 런타임에 동작하는 Java 오브젝트로 혹은 XML 문서로의 변환을 위한 Annotation들을 가지고 있다(이 서비스 Endpoint 인터페이스는 다시 WSDL과 스키마 파일들을 생성하는 데 사용될 수 있으나 이 서비스 Endpoint 인터페이스를 얻기 위해 사용된 WSDL이나 스키마 파일과 일치하지 않을 수 있다).

JAXB가 addNumbers 메소드에 대한 요청과 응답을 Java 오브젝트로 혹은 XML 문서로 변환하기 위해 사용하는 Java Bean 클래스인 AddNumbers 클래스와 AddNumbersResponse 클래스는 다음과 같다.

Java Bean 클래스 : <AddNumbers.java>

```

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbers", propOrder = { "arg0", "arg1" })
public class AddNumbers {

    protected int arg0;
}

```

```

protected int arg1;

public int getArg0() {
    return arg0;
}

public void setArg0(int value) {
    this.arg0 = value;
}

public int getArg1() {
    return arg1;
}

public void setArg1(int value) {
    this.arg1 = value;
}
}

```

Java Bean 클래스 : <AddNumbersResponse.java>

```

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "addNumbersResponse", propOrder = { "_return" })
public class AddNumbersResponse {

    @XmlElement(name = "return")
    protected int _return;

    public int getReturn() {
        return _return;
    }

    public void setReturn(int value) {
        this._return = value;
    }
}

```

위의 2개의 Java Bean 클래스는 원격 웹 서비스의 WSDL 파일의 message element들인 AddNumbers, AddNumbersResponse에 해당하는 것을 알 수 있다.

## 5.2.2. Java SE 클라이언트 방식

본 절에서는 [Client Artifact 생성](#)에서 얻은 클라이언트 Portable Artifact들을 가지고 실제로 원격의 웹 서비스를 호출하는 로직을 담고 있는 클라이언트 클래스를 Java SE 방식으로 작성하고 호출해 본다.

### 5.2.2.1. Java SE 클라이언트 프로그램 작성

Java SE 클라이언트 클래스를 작성하는 것은 매우 간단하다. 위에서 얻은 서비스 인터페이스인 AddNumbersImplService 객체를 하나 생성하고 그로부터 서비스 Endpoint 인터페이스인 AddNumbersImpl을 구현한 객체를 얻는다. 실제로 이 객체는 직접 동적 프록시를 통해 원격의 웹 서비스를 호출하는 로직을 담고 있다. 이렇게 서비스 Endpoint 인터페이스를 구현하는 객체를 얻었으면 이로부터 실제의 웹

서비스를 호출하는 메소드를 호출한다.

다음은 위의 wsimport Ant Task를 통해 얻은 Portable Artifact들을 가지고 원격의 웹 서비스를 호출하는 코드의 일부분이다.

Java SE 클라이언트 프로그램 : <AddNumbersClient.java>

```
public class AddNumbersClient {

    public static void main(String[] args) {
        AddNumbersImpl port = new AddNumbersImplService().getAddNumbersImplPort();

        int number1 = 10;
        int number2 = 20;

        System.out.println("#####");
        System.out.println("### JAX-WS Webservies examples - fromjava ###");
        System.out.println("#####");
        System.out.println("Invoking addNumbers(" + number1 + ", " + number2 + ")");
        int result = port.addNumbers(number1, number2);
        System.out.println("Result: " + result);
    }
}
```

### 5.2.2.2. Java SE 클라이언트 프로그램 호출

본 절에서는 지금까지 구현한 클래스들 및 기타 설정 파일들을 사용하여 웹 서비스를 구현하여 JEUS 9에 deploy한 후 클라이언트를 실행하는 방법을 설명한다.

원격의 웹 서비스에 접근하는 클라이언트 프로그램의 호출 방식은 그 서비스가 Java로부터 구현하거나 WSDL로부터 구현하거나 동일하다.

다음과 같이 Java로부터 구현한다면 fromjava 디렉터리에서 서비스를 생성하고, WSDL 파일로부터 구현한다면 fromwsdl 디렉터리에서 서비스를 생성하여 JEUS 9에 deploy한다.

```
$ ant build deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 빌드한다. Java SE 클라이언트에서 wsimport의 과정을 거치므로 서비스의 deploy가 모두 완료되었을 때 클라이언트의 구성이 가능하다.

다음과 같이 클라이언트를 생성하고 서비스를 호출한다.

```
$ ant run
...

run:
[java] #####
[java] ##### JAX-WS Webservies examples - fromjava #####
[java] #####
[java] Invoking addNumbers(10, 20)
[java] Result: 30
```



...

BUILD SUCCESSFUL

호출 결과로 클라이언트가 정상적으로 이 서비스를 호출하고 원하는 값을 얻었음을 알 수 있다.

### 5.2.3. Jakarta EE 클라이언트 방식

본 절에서는 간단히 앞 장에서 WSDL 문서를 통해 생성한 웹 서비스를 호출하는 클라이언트를 Jakarta EE 방식으로 작성하고 호출한다. 시작하기 전에 우선 [Client Artifact 생성](#)에서 설명한 방식으로 클라이언트 Portable Artifact들을 얻은 것을 가정한다.

#### 5.2.3.1. Jakarta EE 클라이언트 프로그램 작성

Jakarta EE 클라이언트 방식에서 중요한 점은 기존 JAX-RPC 환경의 Jakarta EE 웹 서비스 클라이언트 구성에서는 서블릿일 경우 web.xml에 EJB일 경우에는 ejb-jar.xml에 <service-ref>를 추가하여 웹 서비스 클라이언트 프록시 생성에 필요한 정보를 JNDI에 등록할 수 있었으나, JAX-WS 웹 서비스 환경에서의 Jakarta EE 클라이언트 구성은 @WebServiceRef Annotation을 설정함으로써 그와 동일한 작업이 가능하다는 것이다.

다음은 @WebServiceRef Annotation을 설정하는 하나의 예이다.

```
...
@WebServiceRef(wsdlLocation="http://host:port/TmaxService/TmaxService?wsdl")
static TmaxServiceImplService tsvc;
...
```

위에서 얻은 서비스 인터페이스인 AddNumbersService 타입을 이용하여 클라이언트 Java 클래스의 멤버 변수로 @WebServiceRef Annotation과 함께 선언한다. 그러면 이 멤버 변수는 런타임 중 클라이언트 클래스가 초기화될 때 실제 set 메소드가 없어도 서블릿일 경우 서블릿 컨테이너, EJB일 경우 EJB 컨테이너에서 자동으로 이 값을 Injection한다.

이로부터 서비스 Endpoint 인터페이스인 AddNumbersPortType을 구현한 객체를 얻는다. 실제로 이 객체는 직접 동적 프록시를 통해 원격의 웹 서비스를 호출하는 로직을 담고 있다. 이렇게 서비스 Endpoint 인터페이스를 구현하는 객체를 얻었으면, 실제의 웹 서비스를 호출하는 메소드를 호출한다.

다음은 위의 wsimport Ant Task를 통해 얻은 Portable Artifact들을 가지고 원격의 웹 서비스를 호출하는 클라이언트 프로그램이다.

Jakarta EE 클라이언트 : <AddNumbersClient.java>

```
public class AddNumbersClient extends HttpServlet {

    @WebServiceRef
    static AddNumbersImplService svc;

    protected void doGet(HttpServletRequestRequest arg0, HttpServletResponse arg1)
        throws ServletException, IOException {
        AddNumbersImpl port = svc.getAddNumbersImplPort();
```

```

int number1 = 10;
int number2 = 20;

System.out.println("#####");
System.out.println("### JAX-WS Webservices examples - fromjava ###");
System.out.println("#####");
System.out.println("Invoking addNumbers(" + number1 + ", " + number2 + ")");
int result = port.addNumbers(number1, number2);
System.out.println("#####");
System.out.println("### JAX-WS Webservices examples - fromjava ###");
System.out.println("#####");
System.out.println("Result: " + result);
}
}

```

### 5.2.3.2. Jakarta EE 클라이언트 프로그램 호출

지금까지 구현한 클래스들 및 기타 설정 파일들을 사용하여 웹 서비스를 구현하여 JEUS 9에 deploy한 후 Jakarta EE 클라이언트를 실행할 수 있다. 원격 웹 서비스에 접근하는 클라이언트 프로그램의 호출 방식은 그 서비스가 Java로부터 구현하거나, WSDL로부터 구현하거나 동일하다.

다음과 같이 Java로부터 구현한다면 fromjava 디렉터리에서 서비스를 생성하고, WSDL 파일로부터 구현한다면 fromwsdl 디렉터리에서 서비스를 생성하여 JEUS 9에 deploy한다.

```
$ ant build deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, Jakarta EE 클라이언트를 빌드한다. Jakarta EE 클라이언트에서 wsimport의 과정을 거치므로 서비스의 deploy가 모두 완료되었을 때 클라이언트의 구성이 가능하다.

다음과 같이 클라이언트를 생성하고 호출한다.

```

$ ant run
...

#####
### JAX-WS Webservices examples - fromjava ###
#####
Invoking addNumbers(10, 20)

...

#####
### JAX-WS Webservices examples - fromjava ###
#####
Result: 30

...

```

위와 같이 콘솔에서 입력하면 웹 브라우저를 통해 클라이언트 서블릿이 정상적으로 이 서비스를 호출하고 원하는

값을 서버의 로그를 통해 알 수 있다.

### 5.3. 디스패치 방식의 웹 서비스 호출

디스패치(Dispatch) 방식의 웹 서비스 호출은 XML 구성을 `java.lang.transform.Source` 또는 `jakarta.xml.soap.SOAPMessage`, 즉 XML 수준에서 다루는 것을 선호하는 개발자들을 위한 것이다. 디스패치 방식의 웹 서비스 호출은 메시지 모드와 페이로드(Payload) 모드로 사용될 수 있고 XML/HTTP 바인딩(`jakarta.activation.DataSource`)으로 레스트(REST) 웹 서비스를 생성할 때 사용될 수 있다. 이에 대한 자세한 내용은 [프로바이더와 디스패치 인터페이스](#)를 참고한다.

## 6. 표준 바인딩 선언 및 사용자화

본 장에서는 WSDL 문서에서 Java 클래스를 사용한 표준 바인딩 선언 방법 및 사용자화에 대해 자세하게 설명한다.

### 6.1. 개요

앞에서 살펴본 바와 같이 기본적인 JAX-WS 웹 서비스는 Java 클래스나 WSDL 문서로부터 구현할 수 있다.

보다 확장 가능하고 다양한 기능을 제공하는 웹 서비스를 구현하기 위해서 여러 가지 기능을 추가하거나 다양한 설정을 할 수 있다. 그 중 한 가지 방법은 그러한 기능이나 설정을 Java 클래스에 Annotation을 부여함으로써 wsgen 툴을 통해 얻을 수 있고, 또 다른 방법은 WSDL 문서에 직접 혹은 간접적으로 기능을 추가하거나 설정하여 wsimport 툴을 통해 얻을 수 있다.

본 장에서는 WSDL 문서를 가지고 wsimport 툴을 통해 웹 서비스를 구성하거나 클라이언트를 구성할 때 사용할 수 있는 바인딩 사용자화의 전체적인 모습에 대해 설명한다. 보다 자세한 기능에 대한 소개와 설정 사용법 및 Java 클래스로부터 wsgen 툴을 통해 원하는 웹 서비스를 구현하는 방법은 계속되는 다음 장에서 설명한다.

JEUS 웹 서비스는 WSDL 문서에서 Java 클래스로의 바인딩 선언 및 사용자화(주로 wsimport 툴을 사용하여 작업하는 경우)를 JAX-WS에서 요구하는 표준화된 방식으로 지원한다. 기존 JAX-RPC 방식의 웹 서비스에서는 이러한 표준이 명세화되어 있지 않았기 때문에 서로 다른 벤더 사이에 Portable하지 않은 웹 서비스의 생성 문제를 야기시켰다.

이와 같은 JEUS 9 웹 서비스의 WSDL 문서에서 Java 클래스로의 표준화된 바인딩 선언 및 사용자화는 웹 서비스를 구현하는 데 있어서 2가지 역할을 하는데 이는 다음과 같다.

- 거의 모든 WSDL 컴포넌트들로부터 서비스 Endpoint 인터페이스 클래스, 메소드 이름, 파라미터 이름, 예외(Exception) 클래스 등과 같은 Java 언어로의 매핑을 사용자화할 수 있다.
- 비동기화, 프로바이더, 래퍼(Wrapper) 방식, 부가적인 Header들과 같은 기능을 사용자화할 수 있다.

### 6.2. 표준 바인딩 선언

모든 바인딩에 관련된 element들은 <http://java.sun.com/xml/ns/jaxws> Namespace에 속하며 그 Namespace에 속하는 접두어인 jaxws를 붙여 jaxws:bindings라는 element로 바인딩을 선언하게 된다.

바인딩을 선언하는 장소에 따라 2가지로 나뉘 수 있다.

- 외부 문서를 통해 바인딩을 선언하는 방법
- 바인딩할 WSDL 문서 내에 직접 바인딩 선언을 포함시키는 방법

#### 6.2.1. 외부 문서(파일)에서 직접 선언

외부 문서(파일)에서 직접 선언하는 방법은 주로 웹 서비스를 이용하는 클라이언트에 의해 wsimport 툴의 파라미터로 WSDL 문서의 위치와 함께 전달된다. 여기서 사용되는 WSDL 문서의 위치값은 다음과 같이 바인딩 선언 문서의 wsdlLocation이라는 element 값으로 추가한다.

외부 문서(파일)에서 직접 선언 : <custom-client.xml>

```
<jaxws:bindings
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  jaxws:xmlns="http://java.sun.com/xml/ns/jaxws">
  ...
</jaxws:bindings>
```

또한 WSDL 문서 컴포넌트의 바인딩 설정을 위해서는 위에서 선언한 최상위 element 아래에 자식 element로 추가한다. 선언할 바인딩에 해당하는 컴포넌트는 node라는 element 내에 XPath 문법으로 위치시킨다. 또한 그 컴포넌트에 대해 선언할 바인딩을 그 아래 자식 element로써 추가한다.

다음은 WSDL 문서 컴포넌트의 바인딩 설정을 위해 element를 추가한 예제이다.

WSDL 문서 컴포넌트의 바인딩 설정 : <custom-client.xml>

```
<jaxws:bindings
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  jaxws:xmlns="http://java.sun.com/xml/ns/jaxws">
  <jaxws:bindings node="wsdl:definitions"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
    <jaxws:package name="customize.client"/>
    ...
  </jaxws:bindings>
```

위의 예제에서 이 바인딩 선언은 WSDL 문서 내의 'wsdl:definitions'에 대해 생성되는 Java 클래스들에 대해 그 패키지명을 customize.client로 사용자화하겠다는 의미이다.

이와 같이 생성된 외부 바인딩 선언 문서는 **wsimport** 툴에서 다음과 같이 실행할 수 있다.

```
$ wsimport -b custom-client.xml http://localhost:8080/AddNumbers/addnumbers?WSDL
```

## 6.2.2. WSDL 문서 내에서 직접 선언

WSDL 문서 내에서 직접 선언하는 방법은 앞에서 설명한 외부 문서(파일)에서 직접 선언하는 방법과 다음과 같은 차이가 있다.

- WSDL 문서 내에 확장 element로 <jaxws:bindings> element를 사용한다.
- WSDL 문서 내에서 선언할 경우 node 속성은 사용하지 않는다.
- <jaxws:bindings> element 내에 자식 element로 <jaxws:bindings> element를 사용하지 않는다.
- <jaxws:bindings> element는 그것을 감싸고 있는 WSDL 컴포넌트에만 영향을 준다.

다음은 WSDL 문서 내에서 wsdl:portType에 대한 사용자화 예제이다.

WSDL 문서 내에서 직접 선언 : <AddNumbers.wsdl>

```
<wsdl:portType name="AddNumbersImpl">
  <jaxws:bindings xmlns:jaxws="https://jakarta.ee/xml/ns/jaxws">
  <jaxws:class name="MathUtil"/>
  </jaxws:bindings>
</wsdl:portType>
```

```

    ...
</jaxws:bindings>
<wsdl:operation name="addNumber">
    ...
</wsdl:portType>

```

<jaxws:bindings>라는 확장 element는 이 wsdl:portType으로부터 생성되는 서비스 Endpoint 인터페이스 클래스의 이름에 대한 사용화를 나타내는데, 표준 기본값으로 생성되는 클래스 이름(여기에서는 AddNumbersImpl)을 MathUtil로 사용자화할 것을 나타내고 있다.

## 6.3. 표준 바인딩의 사용자화

본 절에서는 다음의 여러 가지 선언된 바인딩의 사용자화에 대해 자세히 설명한다.

- 전체적인 바인딩
- 패키지명의 사용자화
- Wrapped 스타일
- 비동기화
- 프로바이더 인터페이스
- 클래스명의 사용자화
- Java 메소드의 사용자화
- Java 파라미터의 사용자화
- XML 스키마의 사용자화
- 핸들러 체인의 사용자화

### 6.3.1. 전체적인 바인딩

전체적인 바인딩은 외부 문서 파일로 정의된 바인딩 선언에서 유효하며 jaxws:bindings@wsdlLocation에 명시한 WSDL 문서의 wsdl:definition 내 전체에 적용된다.

다음은 전체적인 바인딩에 해당하는 element이다.

```

<jaxws:package name="..." />
<jaxws:enableWrapperStyle />
<jaxws:enableAsyncMapping />

```

다음은 실제로 사용되는 예이다.

전체적인 바인딩 : <custom-client.xml>

```

<jaxws:bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"

```

```

    xmlns="http://java.sun.com/xml/ns/jaxws">
<package name="customize.client"/>
    <enableWrapperStyle>true</enableWrapperStyle>
    <enableAsyncMapping>false</enableAsyncMapping>
</jaxws:bindings>

```

### 6.3.2. 패키지명의 사용자화

기본값으로 wsimport 툴을 사용하여 WSDL 문서로부터 Java 클래스를 생성하는 경우 클래스의 패키지명은 그 WSDL 파일 문서의 Namespace에 따라 정해진다.

이러한 클래스의 패키지명을 사용자화하여 다른값으로 설정하기 위해서는 jaxws:package element로 바인딩 사용자화 선언을 한다. wsimport 툴을 사용해서 생성하는 경우 -p 옵션으로 패키지명을 변경할 수 있는데 이는 앞에서 jaxws:package element로 바인딩 사용자화한 것을 overwrite한다.

패키지명의 사용자화 (1) : <custom-client.xml>

```

<jaxws:bindings
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  xmlns="http://java.sun.com/xml/ns/jaxws">
<package name="customize.client" />
  ...
</jaxws:bindings>

```

이는 줄여서 다음과 같이 나타낼 수도 있다.

패키지명의 사용자화 (2) : <custom-client.xml>

```

<jaxws:bindings
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  wsdlLocation="http://localhost:8080/jaxws-external-customize/addnumbers?WSDL"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <bindings node="wSDL:definitions">
<package name="customize.client" />
  ...
</jaxws:bindings>

```

### 6.3.3. Wrapped 스타일

wsimport 툴은 기본값으로 WSDL 문서 내의 wSDL:portType으로 정의된 추상 오퍼레이션에 대해 Wrapped 스타일의 규칙을 적용한다. 이러한 Wrapped 스타일은 바인딩 사용자화에 의해 가능하지 않도록 할 수 있다.

Wrapped 스타일 사용자화 : <custom-client.xml>

```

<jaxws:bindings
  xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  xmlns="http://java.sun.com/xml/ns/jaxws">
<enableWrapperStyle>true</enableWrapperStyle>
  <bindings node="wSDL:definitions/"

```

```

        wsdl:portType[@name='AddNumbersImpl']/
        wsdl:operation[@name='addNumbers']">
<enableWrapperStyle>false</enableWrapperStyle>
...
</jaxws:bindings>

```

위와 같이 jaxws:enable WrapperStyle element로써 설정할 수 있으며 wsdl:definitions, wsdl:portType, wsdl:operation element 하위에서 각각 사용될 수 있다.

- wsdl:definitions 아래에서 사용될 경우 모든 wsdl:portType 속성의 모든 wsdl:operations element들에 적용된다.
- wsdl:portType 하위에 사용될 경우 그 portType의 모든 wsdl:operations에 적용된다.
- wsdl:operation 하위에 적용될 경우에는 해당 오퍼레이션에 대해서만 적용된다.

### 6.3.4. 비동기화

클라이언트 애플리케이션은 jaxws:enableAsyncMappingbinding element를 선언함으로써 wsimport하는 경우 비동기 Polling 또는 Callback 방식의 오퍼레이션들을 생성한다. 비동기 클라이언트 애플리케이션에 대해서는 [비동기 웹 서비스](#)에서 자세히 설명한다.

적용되는 WSDL 문서 내의 컴포넌트 및 적용 범위는 위에서 설명한 Wrapped 스타일의 규칙과 동일하다.

비동기화 사용자화 : <custom-client.xml>

```

<jaxws:bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8080/AddNumbers/addnumbers?WSDL"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <enableAsyncMapping>false</enableAsyncMapping>
  <bindings node="wsdl:definitions/"
    wsdl:portType[@name='AddNumbersImpl']/
    wsdl:operation[@name='addNumbers']">
    <enableAsyncMapping>true</enableAsyncMapping>
  ...
</jaxws:bindings>

```

### 6.3.5. 프로바이더 인터페이스

본 절에서는 프로바이더(provider) 인터페이스를 바인딩 선언을 통해 설정하는 방법을 알아본다. 자세한 내용은 [프로바이더와 디스패치 인터페이스](#)를 참고한다.

WSDL 문서 내의 특정 포트 번호(port)를 프로바이더 인터페이스로 설정하고 싶을 경우에는 wsdl:port 노드를 XPath 표현법으로 나타내고 jaxws:provider 바인딩을 설정하면 된다. 기본값은 설정하지 않음으로 되어 있다. 이는 웹 서비스를 WSDL로부터 생성하는 경우 유효하다.



### 6.3.6. 클래스명의 사용자화

WSDL 문서 내의 `wsdl:portType`, `wsdl:fault`, `soap:headerfault`, `wsdl:server`는 Java 클래스로 생성되는데 여기서 `jaxws:class` 바인딩 선언을 하면 원하는 클래스명으로 변경할 수 있다. 이러한 WSDL 문서 내의 컴포넌트들에 대한 각각의 Java 클래스에 대한 설명은 다음과 같다.

- 서비스 Endpoint 인터페이스 클래스

다음은 서비스 Endpoint 인터페이스 클래스명에 대한 바인딩 사용자화의 예이다.

서비스 Endpoint 인터페이스 클래스 바인딩 사용자화 : `<custom-client.xml>`

```
<jaxws:bindings node="wsdl:definitions/wsdl:portType[@name='AddNumbersImpl']">
  <jaxws:class name="TmaxUtil" />
</jaxws:bindings>
```

위의 예에서 서비스 Endpoint 인터페이스 클래스명은 기본값인 'AddNumbersImpl'이 아닌 'TmaxUtil'이 된다.

- 예외(Exception) 클래스

다음은 예외 클래스명에 대한 바인딩 사용자화의 예이다.

예외 클래스명 바인딩 사용자화 : `<custom-client.xml>`

```
<jaxws:bindings node="wsdl:definitions/
  wsdl:portType[@name='AddNumbersImpl']/
  wsdl:operation[@name='addNumbers']/
  wsdl:fault[@name='AddNumbersException']">
  <jaxws:class name="TmaxException" />
</jaxws:bindings>
```

위의 예에서 예외(Exception) 클래스명은 기본값인 'AddNumbersException'이 아닌 'TmaxException'이 된다.

- 서비스 클래스

다음은 서비스 클래스명에 대한 바인딩 사용자화의 예이다.

서비스 클래스명 바인딩 사용자화 : `<custom-client.xml>`

```
<jaxws:bindings node="wsdl:definitions/
  wsdl:service[@name='AddNumbersService']">
  <jaxws:class name="TmaxService" />
</jaxws:bindings>
```

위의 예에서 서비스 클래스명은 기본값인 'AddNumbersService'가 아닌 'TmaxService'가 된다.

### 6.3.7. Java 메소드의 사용자화

서비스 Endpoint의 메소드 이름 혹은 서비스 클래스의 포트를 가져오기 위한 메소드명을 사용자화하는 경우에는 `jaxws:method` 바인딩 선언을 이용한다.

- 서비스 Endpoint 인터페이스 메소드

다음은 서비스 Endpoint 인터페이스의 메소드 이름을 변경하는 바인딩 선언의 예이다.

서비스 Endpoint 인터페이스 메소드 사용자화 : `<custom-client.xml>`

```
<jaxws:bindings node="wsdl:definitions/  
  wsdl:portType[@name='AddNumbersImpl']/  
  wsdl:operation[@name='addNumbers']">  
  <jaxws:method name="add" />  
</jaxws:bindings>
```

`wsimport` 툴은 서비스 Endpoint 인터페이스의 메소드 이름으로 기본값인 'addNumbers' 메소드 이름 대신 'add'라는 메소드 이름을 취한다.

- 포트에 접근하기 위한 서비스 클래스의 메소드

다음은 포트에 접근하기 위한 서비스 클래스의 메소드 이름을 변경하는 바인딩 선언의 예이다.

포트에 접근하기 위한 서비스 클래스 메소드 사용자화 : `<custom-client.xml>`

```
<jaxws:bindings node="wsdl:definitions/  
  wsdl:service[@name='AddNumbersService']/  
  wsdl:port[@name='AddNumbersImplPort']">  
  <jaxws:method name="getTmaxUtil" />  
</jaxws:bindings>
```

`wsimport` 툴은 포트에 접근하기 위한 서비스 클래스의 메소드 이름으로 기본값인 'getAddNumbersImplPort' 대신 'getTmaxUtil'이라는 메소드 이름을 취한다.

### 6.3.8. Java 파라미터의 사용자화

`jaxws:parameter` 바인딩 선언은 생성되는 Java 메소드의 파라미터 이름을 원하는 것으로 변경할 때 사용된다. `wsdl:operation` 또는 `wsdl:portType`의 메소드 파라미터를 변경할 수 있다.

Java 파라미터의 사용자화 : `<custom-client.xml>`

```
<jaxws:bindings node="wsdl:definitions/  
  wsdl:portType[@name='AddNumbersImpl']/  
  wsdl:operation[@name='addNumbers']">  
  <jaxws:parameter part="definitions/  
    message[@name='addNumbers']/  
    part[@name='parameters']"  
    element="tns:number1" name="num1"/>  
</jaxws:bindings>
```

위와 같이 Java 파라미터의 사용자화 선언은 wsdl:operation의 메소드 파라미터를 'number1'에서 'num1'으로 변경된 것을 알 수 있다.

### 6.3.9. XML 스키마의 사용자화

WSDL 문서 내에 선언된 XML 스키마는 jaxws:bindings element 하위에서 표준 JAXB 바인딩 element를 사용함으로써 변경할 수 있다.

XML 스키마의 사용자화 (1) : <custom-client.xml>

```
<jaxws:bindings node="wsdl:definitions/
  wsdl:types/
  xsd:schema[@targetNamespace='http://tmaxsoft.com']">
  <jaxb:schemaBindings>
    <jaxb:package name="fromwsdl.server"/>
  </jaxb:schemaBindings>
</jaxws:bindings>
```

또한 WSDL 문서에 의해 import되는 XML 스키마 파일 또한 바인딩 사용화를 할 수 있는데, 이때에는 JAXB 표준 확장 바인딩 선언 파일을 사용한다.

XML 스키마의 사용자화 (2) : <custom-client.xml>

```
<jxb:bindings
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:jxb="http://java.sun.com/xml/ns/jaxb"
  version="1.0">
  <jxb:bindings
    schemaLocation=
      "http://localhost:8088/AddNumbers/schema1.xsd"
    node="/xsd:schema">
    <jxb:schemaBindings>
      <jxb:package name="fromjava.client"/>
    </jxb:schemaBindings>
  </jxb:bindings>
  ...
```

외부 표준 JAXB 바인딩 선언 파일은 wsimport 툴의 -b 스위치에 의해 전달된다.

### 6.3.10. 핸들러 체인의 사용자화

jaxws:bindings 바인딩 선언은 핸들러를 추가하거나 사용자화하는 데에도 사용된다. 핸들러에 대한 자세한 사항은 [핸들러 프레임워크](#)를 참고한다.

바인딩 사용자화 선언에 핸들러 사항을 추가하거나 변경할 때에는 핸들러 체인 설정에 관한 스키마(JAR 181)에 명시된 것과 같이 핸들러 체인 설정을 jaxws:bindings 안에 설정한다.

핸들러 체인의 사용자화 : <custom-client.xml>

```
<jaxws:bindings
```

```

xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:jaxb="http://java.sun.com/xml/ns/jaxb"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
wsdlLocation="http://localhost:8080/jaxws-fromwsdlhandler/addnumbers?WSDL"
xmlns:jaxws="https://jakarta.ee/xml/ns/jaxws"
xmlns:jakartaee="https://jakarta.ee/xml/ns/jakartaee">
<jaxws:bindings node="wsdl:definitions">
  <jakartaee:handler-chain>
    <jakartaee:handler-chain-name>
      LoggingHandlers
    </jakartaee:handler-chain-name>
    <jakartaee:handler>
      <jakartaee:handler-name>Logger</jakartaee:handler-name>
      <jakartaee:handler-class>
        fromwsdlhandler.common.LoggingHandler
      </jakartaee:handler-class>
    </jakartaee:handler>
  </jakartaee:handler-chain>
</jaxws:bindings>
</jaxws:bindings>

```

이와 같은 외부 바인딩 선언 문서와 WSDL 문서를 wsimport 툴을 통해 Portable Artifact들을 생성하면 핸들러 설정 파일이 또한 생성된다. 생성된 서비스 Endpoint 인터페이스 클래스에는 JAX-WS 런타임이 핸들러들을 찾기 위한 @jakarta.jws.HandlerChain Annotation이 추가된다.

## 7. 핸들러 프레임워크

본 장에서는 웹 서비스의 핸들러 프레임워크에 대한 기본 개념 및 구성과 예를 통해 설정 방법에 대해 설명한다.

### 7.1. 개요

JAX-WS 웹 서비스는 핸들러를 위한 보다 쉬운 플러그인 형태의 프레임워크를 제공하는데, 이는 JEUS 9 웹 서비스의 런타임(Runtime) 시스템 기능을 보다 향상시킬 수 있다.

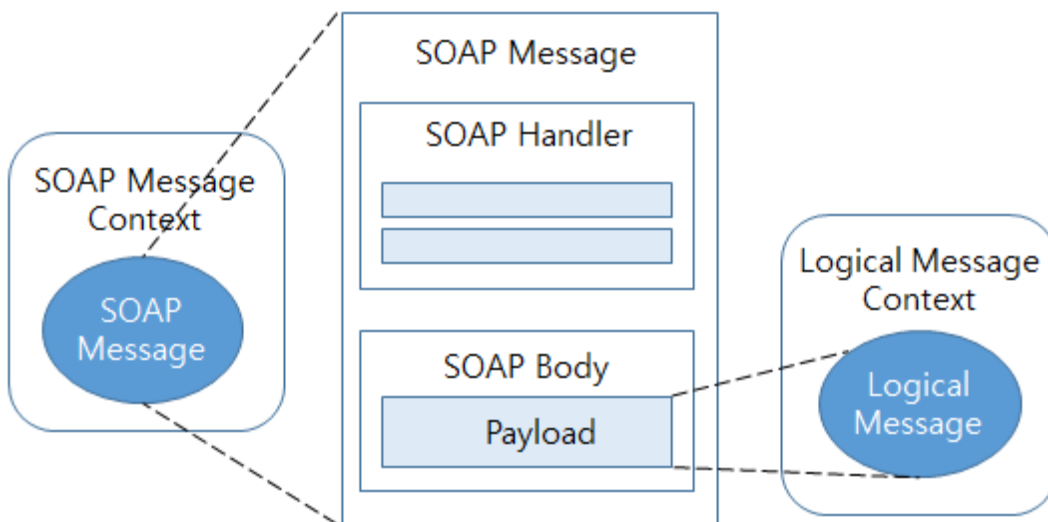
이러한 핸들러의 종류는 다음의 2가지로 구분할 수 있다.

- 논리적 핸들러(Logical Handler)

프로토콜에 무관하게 메시지의 페이로드(payload)에 접근할 수 있는 핸들러이다.

- SOAP 핸들러(SOAP Handler)

Header를 포함한 SOAP 메시지 전체에 접근할 수 있는 핸들러이다.



Relationship between the message contexts

핸들러 프레임워크는 다음과 같은 기준에 의해 사용될 수 있다.

- 전체 SOAP 메시지를 필요로 할 경우 SOAP 핸들러를 사용한다.
- SOAP 메시지의 XML 문서 페이로드만을 필요로 할 경우에는 논리적 핸들러를 사용한다.

기타 다른 경우에는 Endpoint 클래스에서 처리하도록 웹 서비스를 구성하며 특별히 Java 오브젝트를 필요로 하는 경우에는 JEUS EJB에서 지원하는 Interceptor를 사용한다. JEUS EJB Interceptor에 대해서는 "JEUS EJB 안내서"를 참고한다.

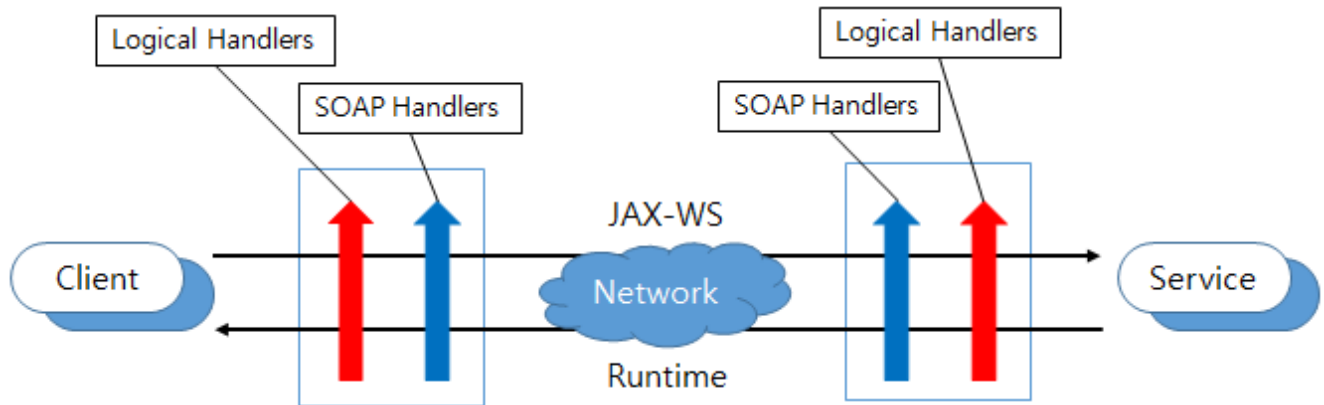
### 7.2. 핸들러 체인 우선순위

핸들러 체인은 다음의 그림과 같이 와이어(Wire) 상에서 나가는(Outbound) 메시지인 경우 모든 Logical 핸들러가

SOAP 핸들러보다 앞서 처리된다. 반대로 들어오는(Inbound) 메시지인 경우 모든 SOAP 핸들러가 Logical 핸들러보다 앞서 처리된다.



즉, 클라이언트 또는 서비스 Endpoint는 프로그래밍에서 논리적 핸들러가 SOAP 핸들러보다 앞에 설정되어 있어도 결국 서비스 생성 혹은 호출하는 경우에는 모든 논리적 핸들러는 SOAP 핸들러를 앞서 처리된다.



Handler Framework

## 7.3. 핸들러 클래스 구성

본 절에서는 사용자가 핸들러 클래스를 구성하는 방법과 핸들러 클래스에서 사용되는 메시지 컨텍스트(MessageContext) 클래스에 대해 설명한다.

### 7.3.1. 핸들러 클래스의 선언

사용자는 핸들러 클래스를 구성하기 위해 Logical 핸들러 혹은 SOAP 핸들러 인터페이스를 구현하는 클래스를 작성한다.

다음은 각각의 핸들러 클래스의 예이다.

핸들러 클래스 : <MyLogicalHandler.java>

```
public class MyLogicalHandler implements
    LogicalHandler<LogicalMessageContext> {
    public boolean handleMessage(LogicalMessageContext messageContext) {
        LogicalMessage msg = messageContext.getMessage();
        return true;
    }
}
```

핸들러 클래스 : <MySOAPHandler.java>

```
public class MySOAPHandler implements
    SOAPHandler<SOAPMessageContext> {
    public boolean handleMessage(SOAPMessageContext messageContext) {
        SOAPMessage msg = messageContext.getMessage();
    }
}
```

```

        return true;
    }
}

```

위와 같이 Logical 핸들러, SOAP 핸들러는 공통적으로 Handler 인터페이스를 구현하고 있다. Handler 인터페이스는 handleMessage( ) 그리고 handleFault( ) 메소드를 지니고 있다.

2가지 메소드는 공통적으로 MessageContext 클래스를 상속하는 객체를 파라미터로 넘겨 받는데, 그 객체는 현재 핸들러로 들어온 메시지가 들어오는(inbound) 것인지 나가는(outbound) 것인지를 구분하는 역할을 한다. 이러한 사용자 핸들러 클래스는 다음과 같이 @PostConstruct Annotation과 @PreDestroy Annotation을 사용할 수 있다.

핸들러 클래스 : <MyLogicalHandler.java>

```

public class MyLogicalHandler implements
    LogicalHandler<LogicalMessageContext> {
    @PostConstruct
    public void methodA() {}

    @PreDestroy
    public void methodB() {}
}

```

위와 같이 선언된 MyLogicalHandler에서 @PostConstruct Annotation으로 선언된 메소드인 'methodA'는 이 핸들러가 생성된 후 호출된다. 또한 @PreDestroy Annotation으로 선언된 메소드인 'methodB'는 이 핸들러가 없어지기 전에 호출된다.

## 7.4. 핸들러 클래스 설정

본 절에서는 구성된 사용자 핸들러를 웹 서비스에 적용시키는 방법에 대해 설명한다.

### 7.4.1. Java 클래스로부터 웹 서비스 구성

Java 클래스로부터 웹 서비스를 구성할 때에는 서비스 Endpoint 구현 클래스에 @HandlerChain Annotation으로 설정하여 wsgen 툴을 사용해서 웹 서비스를 구성한다.

Java 클래스 웹 서비스 구성 : <MyServiceImpl.java>

```

@WebService
@HandlerChain( file="handlers.xml")
public class MyServiceImpl {
    ...
}

```

위와 같이 @HandlerChain Annotation에 설정한 handlers.xml의 모습은 아래와 같다. 실제 서버의 핸들러 설정은 handlers.xml이라는 메타 데이터 파일을 통해 설정된다.

```
<?xml version="1.0" encoding="UTF-8"?>
<jws:handler-chains xmlns:jws="https://jakarta.ee/xml/ns/jakartaee">
  <jws:handler-chain>
    <jws:handler>
      <jws:handler-class>fromjava.handler.TmaxHandler</jws:handler-class>
    </jws:handler>
  </jws:handler-chain>
</jws:handler-chains>
```

## 7.4.2. WSDL로부터 웹 서비스 구성

WSDL로부터 웹 서비스를 구성할 때에는 웹 서비스를 구성할 WSDL 문서에 간접적으로 바인딩 사용자화를 설정하여 wsimport 툴을 사용해서 웹 서비스를 구성한다.

다음은 외부 파일을 이용하여 WSDL 문서에 간접적으로 바인딩을 사용자화하는 예이다.

```
<bindings xmlns="http://java.sun.com/xml/ns/jaxws">
  <handler-chains xmlns:x="https://jakarta.ee/xml/ns/jakartaee">
    <handler-chain>
      <handler>
        <handler-class>fromwsdl.handler.TmaxHandler</handler-class>
      </handler>
    </handler-chain>
  </handler-chains>
</bindings>
```

위와 같이 바인딩 사용자화 파일에 <handler-chains>가 추가된 것에 주목한다.

다음과 같이 <handler-chains> 아래에 <handler-chain>을 여러 개 설정할 수도 있다.

```
<handler-chains xmlns:x="https://jakarta.ee/xml/ns/jakartaee">
  <handler-chain>
    <service-name-pattern xmlns:tm="http://tmaxsoft.com">
      tm:Tmax*Service
    </service-name-pattern>
    <handler>...</handler>
  </handler-chain>

  <handler-chain>
    <port-name-pattern xmlns:tm="http://tmaxsoft.com">
      tm:TmaxPort
    </port-name-pattern>
    <handler>...</handler>
  </handler-chain>

  <handler-chain>
    <protocol-bindings>##SOAP11_HTTP</protocol-bindings>
    <handler>...</handler>
  </handler-chain>
</handler-chains>
```



위와 같이 여러 개의 <handler-chain>이 하나의 <handler-chains>로 구성될 때에는 어떤 핸들러에 서비스 이름이나 포트 이름 또는 프로토콜과 같은 속성을 부여할 수 있다.

### 7.4.3. 클라이언트 구성

위 서비스의 구성 중 WSDL로부터 웹 서비스를 구성하는 방법과 동일하다.

## 7.5. 핸들러 체인을 사용하는 웹 서비스 예제

본 절에서는 로그를 출력하는 사용자 SOAP 핸들러를 구현하여 웹 서비스에 이용하는 간단한 예제를 살펴본다.

Logical 핸들러 클래스에 속하는 jakarta.xml.ws.handler.LogicalHandler 또는 SOAP 핸들러에 속하는 jakarta.xml.ws.handler.SOAPHandler 클래스는 추상 인터페이스인 jakarta.xml.ws.handler.Handler를 상속하고 있다. 이 2가지 클래스의 구현으로 사용자가 원하는 핸들러를 생성할 수 있다.

여기서 구현할 핸들러 클래스인 'LoggingHandler'는 SOAP 핸들러인 jakarta.xml.ws.handler.soap.SOAPHandler를 구현하는 클래스이다.

핸들러 체인을 사용하는 웹 서비스 : <LoggingHandler.java>

```
public class LoggingHandler implements SOAPHandler<SOAPMessageContext> {

    public Set<QName> getHeaders() {
        return null;
    }

    public void close(MessageContext messageContext) {

    }

    public boolean handleFault(SOAPMessageContext smc) {
        return true;
    }

    public boolean handleMessage(SOAPMessageContext smc) {
        Boolean inboundProperty =
            (Boolean) smc.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY);

        System.out.println("\n#####");
        System.out.println("### JAX-WS Webservices examples - handler ###");
        System.out.println("#####");

        if (inboundProperty.booleanValue()) {
            System.out.println("\nClient message:");
        } else {
            System.out.println("\nServer message:");
        }

        SOAPMessage message = smc.getMessage();
        try {
            message.writeTo(System.out);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

    }
    return true;
}
}

```

위와 같이 이 핸들러에서는 핸들러로 들어오는 메시지가 서버로 들어오는 메시지인지 클라이언트로부터 나가는 메시지인지를 검사하고 출력한다.

## 서비스 클래스

다음은 예제의 서비스 부분 Java 클래스이다. 서비스 부분에서 @HandlerChain Annotation을 사용하여 서버의 핸들러 부분을 설정하는 모습을 확인할 수 있다.

핸들러 체인을 사용하는 웹 서비스 클래스 : <handlers.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<handler-chains xmlns="xmlns=https://jakarta.ee/xml/ns/jakartaee">
  <handler-chain>
    <handler>
      <handler-class>fromjavahandler.common.LoggingHandler</handler-class>
    </handler>
  </handler-chain>
</handler-chains>

```

다음은 @HandlerChain에 등록된 파일인 handlers.xml의 예이다.

핸들러 체인을 사용하는 웹 서비스 클래스: <AddNumbersImpl.java>

```

@WebService
@HandlerChain(file = "handlers.xml")
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}

```

## 클라이언트 클래스

다음은 이 예제의 클라이언트 부분 Java 클래스이다. 클라이언트 부분에서는 WSDL 문서로 클라이언트를 생성할 때 사용하는 wsimport 툴에 바인딩 사용자 선언을 추가한다.

핸들러 체인을 사용하는 웹 서비스 클라이언트 클래스 : <AddNumbersClient.java>

```

public class AddNumbersClient {

    public static void main(String[] args) {
        AddNumbersImpl port = new AddNumbersImplService().getAddNumbersImplPort();
        port.addNumbers(10, 20);
    }
}

```

다음은 이러한 바인딩 사용자 선언의 예이다.

핸들러 체인을 사용하는 웹 서비스 바인딩 사용자 선언 : <custom-client.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bindings xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8088/AddNumbers/addnumbers?wsdl"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <bindings node="wsdl:definitions"
    xmlns:jws="https://jakarta.ee/xml/ns/jakartaee">
    <jws:handler-chains>
      <jws:handler-chain>
        <jws:handler>
          <jws:handler-class>
            fromjavahandler.common.LoggingHandler
          </jws:handler-class>
        </jws:handler>
      </jws:handler-chain>
    </jws:handler-chains>
  </bindings>
</bindings>
```

## 7.6. 웹 서비스의 핸들러 프레임워크 실행

본 절에서는 지금까지 구현한 클래스들 및 기타 설정 파일들을 이용하여 핸들러 프레임워크를 실행하는 방법에 대해서 설명한다. 기타 서비스 Endpoint 인터페이스의 구현 클래스 및 기타 설정 파일들은 앞 장에서 설명한 예제의 내용과 동일하다.

다음과 같이 핸들러 프레임워크를 설정한 서비스를 생성하여 JEUS에 deploy한다.

```
$ ant build deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 빌드하고 호출한다. 클라이언트에서 wsimport의 과정을 거치므로 서비스의 deploy가 모두 완료되었을 때 클라이언트의 구성이 가능하다.

다음과 같이 핸들러 프레임워크를 설정한 클라이언트를 생성하고 서비스에 호출한다. 콘솔에서 입력하면 메시지를 주고받는 모습이 LoggingHandler에 의해 서비스와 클라이언트의 화면에 모두 나타나는 것을 알 수 있다.

```
$ ant run

...

run:

[java] #####
[java] ### JAX-WS Webservicess examples - handler ###
[java] #####

[java] Client message:
[java] <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body><ns2:addNumbers xmlns:ns2="http://server.fromjavahandler/"><arg0>10</arg0>
```

```
<arg1>20</arg1></ns2:addNumbers></S:Body></S:Envelope>
[java] #####
[java] ### JAX-WS Webservicex examples - handler ###
[java] #####

[java] Server message:
[java] <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Header/><S:Body><ns2:addNumbersResponse xmlns:ns2="http://server.fromjavahandler/">
<return>30</return></ns2:addNumbersResponse></S:Body></S:Envelope>

...

BUILD SUCCESSFUL
```

## 8. 프로바이더와 디스패치 인터페이스

본 장에서는 서비스 Endpoint의 프로바이더 인터페이스와 클라이언트의 디스패치 인터페이스에 대해서 설명한다.

### 8.1. 개요

웹 서비스의 서비스와 클라이언트는 서로 XML 기반의 메시지를 통해 서로 통신한다. 이때 JAX-WS 서비스 Endpoint 인터페이스는 개발자에게 Java 객체에서 XML 기반의 메시지 혹은 XML 기반의 메시지에서 Java 객체로의 복잡한 상호 변환 과정을 감추는 높은 수준의 추상화를 제공한다. 그러나 어떤 경우에는 개발자가 이러한 서비스와 클라이언트 간의 XML 기반 메시지 교환을 순수하게 메시지 레벨에서(SOAP 메시지 핸들러와 같은 복잡한 프로그래밍 모델의 도움 없이) 다루는 것을 보다 선호하는 경우가 있다.

JEUS 9 웹 서비스는 서비스 Endpoint의 경우 **프로바이더(Provider)**라는 인터페이스와 클라이언트의 경우 **디스패치(Dispatch)**라는 인터페이스를 제공한다. 이를 통해 교환되는 메시지를 순수하게 메시지 수준에서 접근할 수 있도록 하고 있다. 프로바이더와 디스패치 인터페이스는 웹 서비스 개발자나 그 서비스를 소비하는 클라이언트의 개발자에게 메시지를 XML 메시지 수준에서 다룰 수 있도록 해준다.

본 장에서는 이러한 2가지의 인터페이스에 대해 설명한다.

### 8.2. 서비스 Endpoint 프로바이더 인터페이스

본 절에서는 프로바이더 인터페이스에 대해 설명한다.

#### 8.2.1. 프로바이더 인터페이스

프로바이더 인터페이스는 웹 서비스 Endpoint가 XML 문서를 XML 메시지 레벨에서 사용할 때 사용한다. Endpoint는 메시지 또는 메시지 페이로드(payload)에 저수준의 Generic(Java SE 5의 새로운 기능) API를 통해 이에 접근할 수 있다.

다음은 프로바이더 인터페이스의 사용방법이다.

1. @jakarta.xml.ws.WebServiceProvider Annotation을 가지고 있어야 한다.
2. 해당 클래스는 Provider<javax.xml.transform.Source>, Provider<jakarta.xml.soap.SOAPMessage>, Provider<jakarta.activation.DataSource>를 구현해야 한다.
  - Provider<javax.xml.transform.Source>를 구현해서 메시지 페이로드 사용

메시지의 페이로드를 Source 객체로 사용하기 위해서는 @ServiceMode Annotation을 Service.Mode.PAYLOAD 값으로 설정한다. 사실 이는 @WebServiceProvider Annotation만 선언되어 있으면 자동으로 설정되는 기본값이므로 생략할 수 있다.

```
@WebServiceProvider
public class ProviderImpl implements Provider<Source> {
    public Source invoke(Source source) {
        ...
    }
}
```

```
}
}
```

- Provider<jakarta.xml.soap.SOAPMessage>를 구현해서 메시지 사용

메시지를 SOAPMessage 객체로 사용하기 위해서는 @ServiceMode Annotation을 Service.Mode.Message 값으로 설정한다.

```
@WebServiceProvider
@ServiceMode(value=Service.Mode.MESSAGE)
public class ProviderImpl implements Provider<SOAPMessage> {
    public SOAPMessage invoke(SOAPMessage msg) {
        ...
    }
}
```

- Provider<jakarta.activation.DataSource>를 구현해서 메시지 사용

메시지를 Source 객체로 사용하기 위해서는 @ServiceMode Annotation을 Service.Mode.Message 값으로 설정한다. 요청 메시지가 SOAP 메시지일 경우 Attachment를 제외한 SOAP Part 부분만이 Source 객체로 넘어오게 된다. 리턴되는 값이 null 값이라면 이 웹 서비스는 단방향(one-way) 방식의 웹 서비스임을 의미한다.

```
@WebServiceProvider
@ServiceMode(value=Service.Mode.MESSAGE)
public class ProviderImpl implements Provider<Source> {
    public Source invoke(Source source) {
        ...
        return null;
    }
}
```

3. Endpoint가 메시지(Service.Mode.MESSAGE)에 접근할 것인지 혹은 페이로드(Service.Mode.PAYLOAD)에 접근할 것인지에 대해서는 @jakarta.xml.ws.ServiceMode Annotation이 담당한다.

@jakarta.xml.ws.ServiceMode Annotation이 없을 경우 페이로드만을 다루는 것이 기본값이다.

실제로 웹 서비스의 Endpoint를 구성할 때 프로바이더 인터페이스를 구현하는 서비스 구현 클래스는 Endpoint 인터페이스에 대한 어떠한 정보도 주지 못한다. 따라서 반드시 WSDL로부터 웹 서비스를 생성하는 방식을 취해야 한다. 또한 프로바이더 인터페이스를 구현하는 서비스 구현 클래스로 웹 서비스를 구성하는 경우 WSDL로부터 얻은 Portable Artifact들 중 어떤 포트들은 프로바이더 인터페이스를 구현하는 서비스 구현 클래스에 의해 실제로 사용되지 않게 된다. 따라서 WSDL로부터 wsimport 툴을 사용하여 Portable Artifact들을 생성하는데 성능상의 이슈가 큰 경우 이러한 포트들이 생성되지 않도록 외부 바인딩 선언을 이용할 수 있다.

## 8.2.2. 프로바이더 인터페이스 예제

다음은 Provider<Source> 인터페이스를 구현해서 메시지의 페이로드 부분을 다루기 위한 서비스 구현 클래스의 예이다.

```
@ServiceMode(value = Service.Mode.PAYLOAD)
@WebServiceProvider(wsdlLocation = "WEB-INF/wsdl/AddNumbers.wsdl",
    targetNamespace = "http://tmaxsoft.com",
    serviceName = "AddNumbersService", portName = "AddNumbersPort")
public class AddNumbersImpl implements Provider<Source> {
    public Source invoke(Source source) {
        try {
            DOMResult dom = new DOMResult();
            Transformer trans = TransformerFactory.newInstance().newTransformer();
            trans.transform(source, dom);
            Node node = dom.getNode();
            Node root = node.getFirstChild();
            Node first = root.getFirstChild();
            int number1 = Integer.decode(first.getFirstChild().getNodeValue());
            Node second = first.getNextSibling();
            int number2 = Integer.decode(second.getFirstChild().getNodeValue());
            int sum = number1 + number2;

            String body =
                "<ns:addNumbersResponse xmlns:ns=\"http://tmaxsoft.com\"><ns:return>"
                + sum + "</ns:return></ns:addNumbersResponse>";
            Source sumsource =
                new StreamSource(new ByteArrayInputStream(body.getBytes()));

            return sumsource;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

위의 예제에서 AddNumbersImpl이라는 클래스는 @WebServiceProvider라는 Annotation과 함께 Provider<Source>를 구현하며 @ServiceMode Annotation으로 메시지 페이로드를 다루는 것을 알 수 있다.

### 8.2.3. 프로바이더 인터페이스 예제 실행

본 절에서는 지금까지 구현한 클래스들 및 기타 설정 파일들을 사용하여 프로바이더 인터페이스를 구현하는 웹 서비스를 실행하는 법을 설명한다.

다음과 같이 프로바이더 인터페이스를 구현하는 웹 서비스를 생성하여 JEUS에 deploy한다.

```
$ ant deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되었으면, 클라이언트를 빌드한다. 클라이언트에서 wsimport의 과정을 거치므로 서비스의 deploy가 모두 완료되었을 때 클라이언트의 구성이 가능하다.

다음과 같이 클라이언트를 생성해서 서비스를 호출한다. 화면에 메시지를 주고받는 모습이 나타나며 프로바이더 인터페이스를 구현하는 웹 서비스의 모습을 알 수 있다.

```

$ ant run

...

run:
  [java] #####
  [java] ### JAX-WS Webservicess examples - Provider ###
  [java] #####
  [java] Testing Provider webservicess...
  [java] Success!

...

BUILD SUCCESSFUL

...

---[HTTP request]---
Host: localhost:8088
Content-length: 199
Content-type: text/xml; charset=utf-8
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2,
*/*; q=.2
Connection: keep-alive
Soapaction: ""
User-agent: JAX-WS RI 3.0 - JEUS 9
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><addNumbers xmlns="http://tmaxsoft.com"><arg0>10</arg0><arg1>20</arg1></addNumbers></S:Body></S:Envelope>
-----
---[HTTP response 200]---
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns:addNumbersResponse xmlns:ns="http://tmaxsoft.com"><ns:return>30</ns:return></ns:addNumbersResponse></S:Body></S:Envelope>
-----

...

```

## 8.3. 클라이언트 디스패치 인터페이스

본 절에서는 클라이언트 디스패치 인터페이스에 대해 설명한다.

### 8.3.1. 디스패치 인터페이스

서비스 Endpoint의 프로바이더 인터페이스와 대칭되는 개념으로 클라이언트에서도 애플리케이션이 XML 문서를 메시지 수준으로 다룰 수가 있다.

서버의 프로바이더 인터페이스와 마찬가지로 클라이언트의 디스패치 인터페이스를 구현하는 클라이언트 애플리케이션 또한 WSDL로부터 얻은 Portable Artifact들 중 어떤 포트들은 디스패치 인터페이스를 구현하는 클래스에 의해 실제로 사용되지 않게 된다. 따라서 WSDL로부터 wsimport 툴을 사용하여 Portable Artifact들을 생성하는데 성능상의 이슈가 큰 경우 이러한 포트들이 생성되지 않도록 외부 바인딩 선언을 이용할 수 있다.



디스패치 인터페이스의 사용 방법은 다음과 같다.

1. javax.xml.transform.Source, jakarta.xml.soap.SOAPMessage, jakarta.activation.DataSource를 구현한다.

서비스 Endpoint의 프로바이더 인터페이스가 Provider<Source>를 구현해서 클라이언트의 메시지에서 페이로드 부분을 다루거나 Provider<SOAPMessage> 또는 Provider<Source>를 구현해서 클라이언트의 메시지를 다루는 것과 마찬가지로 클라이언트의 디스패치 인터페이스는 SOAPMessage, Source 또는 JAXB Object로 메시지를 구성할 수가 있다.

2. 디스패치 객체의 생성을 위한 서비스 객체 생성(동적인 서비스 오퍼레이션 호출)한다.

서비스(jakarta.xml.ws.Service)는 동적인 서비스 생성을 위한 팩토리의 역할을 한다. 동적인 서비스의 생성이란 WSDL 없이 서비스 객체를 생성하기 위해 실제 서비스의 여러 바인딩에 대한 정보를 동적으로 할당하여 생성하기 위한 것이다.

다음은 동적인 서비스의 생성을 위해 사용하는 코드의 예이다.

```
Service service = Service.createService(QName serviceQName);
Service service = Service.createService(URL wsdlLocation, QName serviceQName);
```

또한 이렇게 생성된 서비스 객체에 대해 디스패치 객체가 바인드되는 특성의 포트나 Endpoint를 다음과 같이 addPort() 메소드를 통해 추가할 수 있다. 이렇게 추가된 포트를 통해 클라이언트 애플리케이션은 디스패치 인터페이스를 통해 호출할 수 있다.

위의 서비스 객체를 동적으로 wsdlLocation의 정보를 통해 생성하는 경우 다음 메소드는 동작하지 않는다는 점에 주의한다. 이미 wsdlLocation의 정보를 통해 서비스는 포트를 구성했기 때문이다.

```
service.addPort(QName portName, String SOAPBinding.SOAP11HTTP_BINDING,
    String endpointAddress);
service.addPort(QName portName, String SOAPBinding.SOAP12HTTP_BINDING,
    String endpointAddress);
service.addPort(QName portName, String HTTPBinding.HTTP_BINDING,
    String endpointAddress);
```

위는 각각 차례대로 SOAP 1.1, SOAP 1.2 그리고 XML/HTTP 바인딩을 QName과 Endpoint 주소를 통해 포트를 구성하는 방식이다.

WSDL 파일로부터 생성되는 서비스 객체에 대해서는 위의 오퍼레이션들은 의미가 없다. 그 이유는 wsimport 툴에 의해 WSDL 파일로부터 생성될 때의 서비스는 이러한 포트 바인딩 혹은 QName, Endpoint 주소와 같은 정보들을 이미 생성되는 Portable Artifact들 중 서비스 객체가 알고 있기 때문이다.

다음은 일반적인 WSDL 파일로부터 wsimport 툴을 통해 생성된 Portable Artifact들 중 서비스 객체를 생성하는 예제 코드이다.

```
Service service = new AddNumbersService();
```

3. 디스패치 객체를 생성한다. 서비스 객체를 동적으로 또는 WSDL 파일로부터 생성했으면 디스패치 객체는 다음의

서비스 클래스의 2가지 메소드를 통해 생성된다.

```
Dispatch dispatch = service.createDispatch(QName portName,
    Class clazz, Service.Mode mode);
Dispatch dispatch = service.createDispatch(QName portName,
    JAXBContext jaxbcontext, Service.Mode mode);
```

디스패치 객체는 javax.xml.transform.Source 혹은 JAXB Data Binding Object를 이용할 수 있는데 이 2가지의 경우 Service 객체로부터 Service.Mode.PAYLOAD 혹은 Service.Mode.MESSAGE 모드와 createDispatch 메소드를 통해 디스패치 객체를 생성한다.

또한 디스패치 객체가 jakarta.xml.soap.SOAPMessage를 이용할 경우에는 Service 객체로부터 Service.Mode.MESSAGE 모드와 createDispatch 메소드를 통해 디스패치 객체를 생성한다.

### 8.3.2. 디스패치 인터페이스 예제

다음은 원격 웹 서비스의 WSDL 문서로부터 디스패치 인터페이스를 구현한 클라이언트 웹 서비스의 예이다.

디스패치 인터페이스 : <AddNumbersClient.java>

```
public class AddNumbersClient {

    public static void main(String[] args) {
        try {
            Service service = new AddNumbersService();
            String request = "<addNumbers xmlns='http://tmaxsoft.com'>
                <arg0>10</arg0><arg1>20</arg1></addNumbers>";
            QName portQName = new QName("http://tmaxsoft.com", "AddNumbersPort");
            Dispatch<Source> sourceDispatch = service.createDispatch(portQName,
                Source.class, Service.Mode.PAYLOAD);

            System.out.println("#####");
            System.out.println("### JAX-WS Webservices examples - Dispatch ###");
            System.out.println("#####");
            System.out.println("Testing Dispatch webservices...");

            Source result =
                sourceDispatch.invoke(new StreamSource(new StringReader(request)));

            DOMResult dom = new DOMResult();
            Transformer trans = TransformerFactory.newInstance().newTransformer();
            trans.transform(result, dom);
            Node node = dom.getNode();
            Node root = node.getFirstChild();
            Node first = root.getFirstChild();
            int sum = Integer.decode(first.getFirstChild().getNodeValue());
            if (sum == 30) {
                System.out.println("Success!");
            } else {
                System.out.println("Fail!");
            }
        } catch (ProtocolException jex) {
            jex.printStackTrace();
        } catch (TransformerException e) {
            e.printStackTrace();
        }
    }
}
```

```

    }
}
}

```

위의 예제에서 AddNumbersClient이라는 클래스는 WSDL로부터 얻은 서비스 객체에 대해 createDispatch() 메소드를 통해 Dispatch<Source> 타입의 sourceDispatch 객체를 얻어 프로그래밍하는 것을 알 수 있다.

### 8.3.3. 디스패치 예제 실행

지금까지 구현한 클래스들 및 기타 설정 파일들을 사용하여 디스패치 인터페이스를 구현하는 웹 서비스를 실행하는 방법은 다음과 같다.

디스패치 인터페이스를 구현하는 웹 서비스를 생성하여 JEUS에 deploy한다.

```
$ ant deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 빌드한다. 클라이언트에서 wsimport의 과정을 거치므로 서비스의 deploy가 모두 완료되었을 때 클라이언트의 구성이 가능하다.

다음과 같이 클라이언트를 생성해서 서비스를 호출한다. 화면에 메시지를 주고받는 모습이 나타나며 디스패치 인터페이스를 구현하는 웹 서비스의 모습을 알 수 있다.

```

$ ant run

...

run:
[java] #####
[java] ### JAX-WS Webservicess examples - Dispatch ###
[java] #####
[java] Testing Dispatch webservicess...
[java] Success!

...

BUILD SUCCESSFUL

...

---[HTTP request]---
Host: localhost:8088
Content-length: 199
Content-type: text/xml; charset=utf-8
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Soapaction: ""
User-agent: JAX-WS RI 3.0 - JEUS 9
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><addNumbers xmlns="http://tmaxsoft.com"><arg0>10</arg0><arg1>20</arg1></addNumbers></S:Body></S:Envelope>

```

```

-----
---[HTTP response 200]---
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><addNumbersResponse xmlns="http://tmaxsoft.com"><return>30</return></addNumbersResponse></S:Body></S:Envelope>
-----
...

```

## 8.4. XML/HTTP 바인딩

웹 서비스에서는 메시지를 주고받을 때 SOAP 메시지가 아닌 XML 문서 자체를 그대로 HTTP에 실어 보내는 경우가 있다. 이와 같은 웹 서비스를 RESTful 웹 서비스라고도 한다. 기본적으로 RESTful 웹 서비스의 서비스와 클라이언트는 프로바이더와 디스패치 인터페이스를 사용한다. 프로바이더와 디스패치 인터페이스에 대해서는 각각 [서비스 Endpoint 프로바이더 인터페이스](#)와 [클라이언트 디스패치 인터페이스](#)를 참고한다.

본 절에서는 이러한 RESTful 웹 서비스를 지원하기 위한 JEUS 웹 서비스의 XML/HTTP 바인딩에 대해 설명한다.

### 8.4.1. RESTful 웹 서비스

RESTful 웹 서비스의 주요 특징은 다음과 같다.

- HTTP의 GET Request 또한 Endpoint에 전달될 수 있다.
- 표준 MessageContext.QUERY\_STRING과 MessageContext.PATH\_INFO를 통해 필요한 HTTP request의 쿼리(Query String)와 경로(path) 정보를 얻는다.

RESTful 웹 서비스는 보통 WSDL 문서를 통해 웹 서비스와 클라이언트를 구성하지 않는다. RESTful 웹 서비스는 사전에 미리 약속된 XML 문서의 스키마를 통해 웹 서비스 공급자와 소비자는 웹 서비스를 구성한다. 공급자는 서비스 클래스를 wsgen 툴을 통해 구성하며 클라이언트 또한 Portable Artifact의 도움 없이 독립적으로 애플리케이션을 구성한다.

JAX-WS는 보다 편리한 RESTful 웹 서비스를 위해 프로바이더와 디스패치 인터페이스에서 XML/HTTP 바인딩을 지원하는데 이를 통해 RESTful 웹 서비스를 공급자와 소비자는 JEUS 웹 서비스를 통해 보다 쉽게 구성할 수 있다.

### 서비스 Endpoint 구성

[서비스 Endpoint 프로바이더 인터페이스](#)에서 프로바이더 인터페이스를 구현하기 위해 서비스 Endpoint가 Provider<Source> 또는 Provider<SOAPMessage>를 구현하는 클래스로 생성해서 바인딩된 Source와 SOAPMessage 객체를 통해 메시지의 페이로드 혹은 전체를 다룰 수 있도록 한다고 설명했다.

프로바이더 Endpoint는 바인딩 식별자를 사용해서 다른 바인딩으로 설정할 수 있는데 이러한 바인딩 식별자는 Endpoint 클래스가 @BindingType Annotation을 설정함으로 가능하다. 이러한 바인딩 식별자가 없을 경우 기본 바인딩인 SOAP1.1/HTTP가 선언된다.

다음은 이러한 바인딩 식별자를 이용해서 XML/HTTP 바인딩을 선언한 예이다.

```
@ServiceMode(value=Service.Mode.MESSAGE)
```

```
@BindingType(value=HTTPBinding.HTTP_BINDING)
public class ProviderImpl implements Provider<Source> {
    public Source invoke(Source source) {
        ...
    }
}
```

위에서와 같이 서비스 Endpoint 클래스는 들어오는 메시지를 XML/HTTP로 바인딩하겠다는 선언을 했으므로 Provider<SOAPMessage>가 아닌 Provider<Source>를 구현하고 있다.

## 클라이언트 구성

RESTful 웹 서비스의 클라이언트 구성은 [디스패치 인터페이스](#)의 방법2에서 살펴본 바와 같이 동적인 서비스 객체를 생성하고 HTTPBinding.HTTP\_BINDING으로 포트를 생성해서 다음과 같이 프로그래밍한다.

```
Service service = Service.createService(QName serviceName);
service.addPort(QName portName, String HTTPBinding.HTTP_BINDING, String endpointAddress);
```

이렇게 생성된 서비스 객체를 사용해서 디스패치 객체를 생성하고, 필요하다면 이 서비스 객체에 Request를 POST 또는 GET 방식으로 지정하고 필요한 쿼리 스트링(MessageContext.QUERY\_STRING)과 패스(MessageContext.PATH\_INFO) 정보를 등록하여 서비스를 호출하면 된다.

다음은 이러한 클라이언트 코드를 구성하는 예이다.

```
...

Dispatch<Source> d = service.createDispatch(portQName, Source.class, Service.Mode.MESSAGE);
Map<String, Object> requestContext = d.getRequestContext();
requestContext.put(MessageContext.HTTP_REQUEST_METHOD, new String("GET"));
requestContext.put(MessageContext.QUERY_STRING, queryString);
requestContext.put(MessageContext.PATH_INFO, path);
Source result = d.invoke(null);

...
```



invoke 메소드에 아무런 파라미터도 넘기지 않는다는 점에 주의한다.

## 8.4.2. RESTful 웹 서비스 예제

본 절에서는 RESTful 웹 서비스와 클라이언트 예제에 대해서 설명한다.

### RESTful 웹 서비스 예제

다음은 RESTful 웹 서비스에서 서비스에 해당하는 Java 클래스의 예이다.

RESTful 웹 서비스 : <AddNumbersImpl.java>

```
@WebServiceProvider(serviceName = "AddNumbersService")
@ServiceMode(value = Service.Mode.MESSAGE)
@BindingType(value = HTTPBinding.HTTP_BINDING)
public class AddNumbersImpl implements Provider<Source> {

    @Resource(type = Object.class)
    protected WebServiceContext wsContext;

    public Source invoke(Source source) {
        try {
            MessageContext mc = wsContext.getMessageContext();
            String query = (String) mc.get(MessageContext.QUERY_STRING);
            StringTokenizer st = new StringTokenizer(query, "&/");
            st.nextToken();
            int number1 = Integer.parseInt(st.nextToken());
            st.nextToken();
            int number2 = Integer.parseInt(st.nextToken());
            int sum = number1 + number2;
            String body =
                "<ns:addNumbersResponse xmlns:ns=\"urn:AddNumbers\"><ns:return>"
                + sum + "</ns:return></ns:addNumbersResponse>";
            Source resultsrc =
                new StreamSource(new ByteArrayInputStream(body.getBytes()));
            return resultsrc;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

위와 같이 RESTful 웹 서비스의 선언은 BindingType Annotation으로 HTTPBinding.HTTP\_BINDING을 설정하고 Resource Annotation으로 WebServiceContext 타입의 멤버변수 wsContext를 설정한 것을 알 수 있다. wsContext는 런타임에 JAX-WS RI가 자동으로 Injection한다.

대략적인 이 클래스의 모습을 살펴보면 WebServiceContext로부터 MessageContext를 얻고 이를 통해 쿼리(Query String) 값을 얻은 후 sendSource 메소드를 통해 파싱 및 반환해줄 Source를 직접 형성해서 이를 리턴값으로 넘겨주고 있다.

## RESTful 웹 서비스 클라이언트 예제

다음은 RESTful 웹 서비스에서 클라이언트에 해당하는 Java 클래스의 예이다.

RESTful 웹 서비스 클라이언트 : <AddNumbersClient.java>

```
public class AddNumbersClient {

    public static void main(String[] args) throws Exception {
        String endpointAddress = "http://localhost:8088/AddNumbers/addnumbers";
        URL url = new URL(endpointAddress + "?num1=10&num2=20");
        System.out.println("#####");
        System.out.println("### JAX-WS Webservices examples - RESTful ###");
        System.out.println("#####");
    }
}
```

```

System.out.println("Testing RESTful webservices...");
InputStream in = url.openStream();
StreamSource source = new StreamSource(in);

try {
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    StreamResult sr = new StreamResult(bos);
    Transformer trans = TransformerFactory.newInstance().newTransformer();
    Properties oprops = new Properties();
    oprops.put(OutputKeys.OMIT_XML_DECLARATION, "yes");
    trans.setOutputProperties(oprops);
    trans.transform(source, sr);
    System.out.println(bos.toString());
    bos.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
}

```

위와 같이 RESTful 웹 서비스의 클라이언트는 WSDL로부터 생성할 Portable Artifact 코드들의 모습은 보이지 않는다. 단지 URL 객체를 통해 직접 만든 Source를 스트림 형태로 보내주고 있음을 알 수 있다.

### 8.4.3. RESTful 웹 서비스 예제 실행

본 절에서는 구현한 클래스들 및 기타 설정 파일들을 가지고 RESTful 웹 서비스를 실행하는 방법에 대해서 설명한다.

다음과 같이 RESTful 웹 서비스를 생성하여 JEUS에 deploy한다.

```
$ ant build deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 빌드한다.

다음과 같이 RESTful 웹 서비스를 위한 클라이언트를 생성하고 서비스를 호출한다. 다음과 같이 콘솔에서 입력하면 정상적으로 메시지를 주고받는 모습을 확인할 수 있다.

```

$ ant run

...

run:
[java] #####
[java] ### JAX-WS Webservices examples - RESTful ###
[java] #####
[java] Testing Restful webservices...
[java] <ns:addNumbersResponse xmlns:ns="urn:AddNumbers"><ns:return>30</ns:r
eturn></ns:addNumbersResponse>

...

BUILD SUCCESSFUL

```

...

---[HTTP request]---

Host: localhost:8088

Content-type: application/x-www-form-urlencoded

Accept: text/html, image/gif, image/jpeg, \*; q=.2, \*/\*; q=.2

Connection: keep-alive

User-agent: JAX-WS RI 3.0 - JEUS 9

-----

---[HTTP response 200]---

<ns:addNumbersResponse xmlns:ns="urn:AddNumbers"><ns:return>30</ns:return></ns:addNumbersResponse>

-----

...



## 9. 비동기 웹 서비스

본 장에서는 클라이언트의 비동기 오퍼레이션과 비동기 프로바이더를 이용한 비동기 웹 서비스 설정 방법에 대해서 설명한다.

### 9.1. 개요

서비스와 클라이언트 간의 웹 서비스 호출에서 클라이언트는 서버의 응답을 받을 때까지 그 스레드를 블록시키고 기다리고 있다. 이와 같은 비효율성을 개선하기 위해 JAX-WS 웹 서비스는 다음과 같은 비동기 오퍼레이션(Operation)을 구성한다.

- 클라이언트 측면에서 JAX-WS API를 사용한 비동기 오퍼레이션

JAX-WS 웹 서비스의 클라이언트를 구성하기 위해서는 호출할 서비스의 WSDL 파일을 이용한다. 서비스 WSDL 파일을 바인딩 사용자화 선언을 통해 정적인 비동기 메소드를 가진 서비스 Endpoint 인터페이스 Stub을 생성하고 그것을 구현하는 클라이언트 클래스를 구성함으로써 클라이언트의 비동기 오퍼레이션을 구성하는 방법이다.

- 서비스 측면에서 JEUS가 제공하는 비표준적인 비동기 오퍼레이션

웹 서비스 Endpoint를 Servlet 3.0의 비동기 처리 방식으로 동작하는 비동기 웹 서비스로 구성하는 방법이다. JAX-WS 표준은 서비스 측면의 비동기 오퍼레이션을 규정하고 있지 않다.

### 9.2. 클라이언트 비동기 오퍼레이션

본 절에서는 클라이언트 측면에서 JAX-WS API를 사용한 비동기 오퍼레이션에 대해 설명한다.

#### 9.2.1. 비동기 메소드를 가진 SEI Stub 이용 방법

비동기화 wsdl:operation은 Polling과 Callback 메소드로 매핑된다. Polling 메소드는 jakarta.xml.ws.Response 인터페이스를 리턴하고, Callback 메소드는 jakarta.xml.ws.AsyncHandler 인터페이스를 리턴한다. 우선 서비스의 WSDL로부터 이러한 비동기 메소드를 가진 SEI(Service Endpoint Interface) Stub을 얻기 위해 사용하는 비동기화 바인딩 선언에 대해 알아본다.

##### 9.2.1.1. 비동기화 바인딩 선언

WSDL에서 명시된 wsdl:operation element를 비동기화된 매핑으로 사용하기 위해서는 JEUS 웹 서비스의 wsimport와 같은 툴을 이용하여 비동기화 wsdl:operation 매핑에 기반한 SEI를 생성해야 한다.

본 절에서는 이러한 비동기화 wsdl:operation 매핑을 생성하는 방법에 대해 설명한다.

비동기화 wsdl:operation 매핑을 생성하기 위해서는 wsimport 툴에 의한 바인딩 사용자화 설정 작업을 해야 한다. 다음은 바인딩 설정 파일인 custom-schema.xml의 한 예이다.

비동기화 바인딩 : <custom-schema.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<bindings xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  wsdlLocation="http://localhost:8088/AddNumbers/addnumbers?wsdl"
  xmlns="http://java.sun.com/xml/ns/jaxws">
  <bindings node="wsdl:definitions">
    <enableAsyncMapping>true</enableAsyncMapping>
  </bindings>
</bindings>
```

위와 같이 wsdl:definitions에 대해 바인딩을 설정하면 이 WSDL 문서 내의 모든 wsdl:operation element들에 대해 비동기화 설정이 된다.

다음은 custom-schema.xml 파일을 사용하는 build.xml의 한 부분이다.

비동기화 바인딩 : <build.xml>

```
...
<target name="build_client" depends="do-deploy-success, init">
  <antcall target="wsimport">
    <param name="package.name" value="async.client" />
    <param name="binding.file" value="-b ${src.conf}/custom-client.xml" />
    <param name="wsdl.file"
      value="http://localhost:8088/AddNumbers/addnumbers?wsdl" />
  </antcall>
  <antcall target="do-compile">
    <param name="javac.excludes" value="fromjava/server/" />
  </antcall>
</target>
...
```

이와 같이 바인딩 사용자 선언으로 wsimport 툴을 이용하여 Portable Artifact들을 생성하면 생성된 비동기 메소드들을 가진 SEI는 다음과 같다.

```
public int addNumbers(int number1, int number2)
    throws java.rmi.RemoteException;
public Response<AddNumbersResponse> addNumbers(int number1, int number2);
public Future<?> addNumbers(int number1, int number2,
    AsyncHandler<AddNumbersResponse>);
```

위와 같이 Response<AddNumbersResponse>와 Future<?>를 리턴값으로 갖는 메소드가 2개 생성된다. 이는 각각 Polling 방식과 Callback 방식의 메소드인데 이들을 사용해서 클라이언트 Java 클래스를 어떻게 구성하는지에 대해 다음 절에서 설명한다.

### 9.2.1.2. 비동기화 클라이언트 구성

비동기화 클라이언트는 Polling 메소드를 사용하거나 Callback 메소드를 사용해서 구성할 수 있다.

## Polling 메소드를 사용하는 클라이언트의 구성 방법

wsimport 툴로부터 얻은 비동기 SEI의 Polling 메소드는 다음과 같다.

```
public Response<AddNumbersResponse> addNumbers(int number1, int number2);
```

다음은 이러한 Polling 메소드로 매핑된 메소드를 사용하여 구현한 클라이언트 웹 서비스 예의 일부분이다. 클라이언트 애플리케이션은 SEI의 비동기 Polling 메소드를 호출하게 되고 언제 결과값이 반환되는지를 확인할 수 있다.

```
jakarta.xml.ws.Response<AddNumbersResponse> resp = port.addNumbersAsync(10, 20);
while(!resp.isDone()){
}
System.out.println(resp.get().getReturn());
...
```

위와 같이 Polling 메소드로 매핑된 메소드는 jakarta.xml.ws.Response 타입의 객체를 리턴한다. 이는 java.util.concurrent.Future<T>로부터 상속된 isDone() 메소드를 통해서 언제 이 오퍼레이션이 완료되어 결과를 반환하는지를 결정할 수 있다.

다음은 Polling 메소드를 지원하는 클라이언트 애플리케이션 예제 코드의 일부분이다.

Polling 메소드를 사용하는 클라이언트 : <AddNumbersClient.java>

```
public class AddNumbersClient {

    ...

    public static void main(String[] args) {
        try {
            AddNumbersImpl port = new AddNumbersService().getAddNumbersImplPort();

            // Asynchronous polling
            Response<AddNumbersResponse> resp = port.addNumbersAsync(10, 20);
            Thread.sleep(2000);
            AddNumbersResponse output = resp.get();
            System.out.println("#####");
            System.out.println("### JAX-WS Webservises examples - polling ###");
            System.out.println("#####");
            System.out.printf("call webservises in an Asynchronous Polling way...");
            System.out.printf("result : %d\n", output.getReturn());

            ...

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    ...
}
```

## Callback 메소드를 사용하는 클라이언트의 구성 방법

wsimport 툴로부터 얻은 비동기 SEI의 Callback 메소드는 다음과 같다.

```
public Future<?> addNumbers(int number1, int number2, AsyncHandler<AddNumbersResponse>);
```

Callback 메소드로 매핑된 메소드는 클라이언트 개발자가 추가적인 파라미터로써 `jakarta.xml.ws.AsyncHandler`를 구현한 핸들러 객체를 제공한다.

다음은 `AsyncHandler`를 구현한 핸들러 객체의 예제 코드이다.

Callback 메소드 사용하는 클라이언트 핸들러 객체 : `<AddNumbersClient.java>`

```
public class AddNumbersClient {
    ...
    static class AddNumbersCallbackHandler implements
        AsyncHandler<AddNumbersResponse> {

        private AddNumbersResponse output;

        public void handleResponse(Response<AddNumbersResponse> response) {
            try {
                output = response.get();
            } catch (ExecutionException e) {
                e.printStackTrace();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        AddNumbersResponse getResponse() {
            return output;
        }
    }
}
```

위와 같이 추가적인 `AsyncHandler`를 구현한 핸들러 객체는 런타임에 서버로부터 그 웹 서비스 오퍼레이션의 결과를 얻을 수 있을 때 `handleResponse`라는 메소드를 호출하게 되고, 클라이언트 애플리케이션은 `getResponse()` 메소드를 통해 그 결과값을 얻을 수 있다.

다음은 위에서 구현한 핸들러 객체를 이용하여 SEI의 비동기 Callback 메소드를 이용하는 클라이언트 애플리케이션 예제 코드의 일부분이다. 클라이언트 애플리케이션은 SEI의 비동기 Callback 메소드를 호출하게 되고 언제 결과값이 반환되는지를 확인할 수 있다.

```
AddNumbersCallbackHandler callbackHandler = new AddNumbersCallbackHandler();
Future<?> resp = port.addNumbersAsync(number1, number2, callbackHandler);
while(!resp.isDone()){
}
System.out.println(callbackHandler .getResponse().getReturn());
```

위와 같이 Callback 메소드로 매핑된 메소드는 `javax.util.concurrent.Future` 타입의 객체를 리턴한다. 이는 `isDone()` 메소드를 통해서 언제 이 오퍼레이션이 완료되어 결과를 반환하는지를 결정할 수 있다.

다음은 SEI의 비동기 Callback 메소드를 이용하는 클라이언트 애플리케이션 예제 코드의 일부분이다.

Callback 메소드를 이용하는 클라이언트 : <AddNumbersClient.java>

```
public class AddNumbersClient {

    public static void main(String[] args) {
        try {
            AddNumbersImpl port = new AddNumbersService().getAddNumbersImplPort();
            ...

            // Asynchronous callback
            AddNumbersCallbackHandler callbackHandler = new AddNumbersCallbackHandler();
            Future<?> response = port.addNumbersAsync(10, 20, callbackHandler);
            Thread.sleep(2000);

            output = callbackHandler.getResponse();
            System.out.println("#####");
            System.out.println("### JAX-WS Webservices examples - callback ###");
            System.out.println("#####");
            System.out.printf("call webservices in an Asynchronous Callback way...");
            System.out.printf("result: %d\n", output.getReturn());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    ...
}
```

### 9.2.1.3. 비동기화 클라이언트 실행

본 절에서는 구현한 클래스들 및 기타 설정 파일들을 가지고 핸들러 프레임워크를 실행하는 방법에 대해서 설명한다.

다음과 같이 비동기화 오퍼레이션을 설정한 웹 서비스를 생성하여 JEUS에 deploy한다.

```
$ ant build deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면 클라이언트를 빌드한다.

핸들러 프레임워크를 설정한 클라이언트를 생성하고 클라이언트로부터 서비스를 호출한다.

콘솔에 다음과 같이 입력하면 Polling 방식과 Callback 방식으로 2번의 응답을 정상적으로 받는 모습을 확인할 수 있다.

```
$ ant run

...

run:
```

```
[java] #####
[java] ### JAX-WS Webservicess examples - polling ###
[java] #####
[java] call webservicess in an Asynchronous Polling way...result : 30
[java] #####
[java] ### JAX-WS Webservicess examples - callback ###
[java] #####
[java] call webservicess in an Asynchronous Callback way...result: 30
```

...

BUILD SUCCESSFUL

## 9.2.2. 디스패치 인터페이스 이용하는 방법

디스패치 인터페이스를 이용하여 클라이언트의 비동기 오퍼레이션을 사용하는 것은 기본적으로 비동기 메소드를 가진 SEI Stub 이용하는 것과 개념은 동일하다. 다만 생성된 디스패치 객체에서 제공하는 `invokeAsync` 메소드를 사용한다.

다음은 `invokeAsync` 메소드를 사용하는 예제이다.

```
Response<T> response = dispatch.invokeAsync(T);
Future<?> response = dispatch.invokeAsync(T, AsyncHandler);
```

위에서와 같이 `invokeAsync(T)`는 Polling 방식의 비동기를 지원하는 메소드이고, `invokeAsync(T, AsyncHandler)`는 Callback 방식의 비동기를 지원하는 메소드이다. `AsyncHandler`는 Callback 방식으로 사용하기 위해 사용자 핸들러를 구현한 것이다.

## 9.3. 비동기 웹 서비스

JEUS JAX-WS는 Servlet 3.0 비동기 처리 기반의 비동기 웹 서비스 구성 방법을 제공한다. 웹 서비스 Endpoint는 요청을 받고 응답을 보낼 때까지 동기화되어 서블릿 컨테이너가 할당된 요청 처리 스레드를 점유하고, 서비스에서의 처리 시간이 길어진다. 이로 인하여 다른 요청들이 대기하는 상황이 자주 발생할 수 있다. 이 경우 처리 시간이 긴 서비스에는 별도의 스레드를 할당하고, 요청 처리 스레드는 컨테이너에 반환하여 대기 요청을 효율적으로 처리하게 할 수 있다.

본 절에서는 비동기 웹 서비스의 설정방법에 대해서 설명한다.

### 9.3.1. 비동기 웹 서비스 설정

JEUS는 JAX-WS 웹 서비스를 비동기 웹 서비스로 구성하기 위해서는 다음과 같은 방법을 제공한다.

- **@jeus.webservices.jaxws.api.AsyncWebService Annotation을 사용하는 경우**

다음은 Annotation을 사용하여 웹 서비스를 비동기 웹 서비스로 설정하는 예제이다.

비동기 웹 서비스 설정 : <AddNumbersImpl.java>

```
@WebService(serviceName="AddNumbers")
@AsyncWebService
public class AddNumbersImpl {

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

- **web.xml의 <async-supported>를 사용하는 경우**

이미 구현된 웹 서비스는 web.xml에 Servlet 3.0이 지원하는 async-supported 사용하여 비동기 웹 서비스로 설정할 수 있다.

비동기 웹 서비스 설정 : <web.xml>

```
<web-app>
    <servlet>
        <servlet-name>AddNumbers</servlet-name>
        <servlet-class>fromwsdl.server.AddNumbersImpl</servlet-class>
    <async-supported>true</async-supported>
    </servlet>
    <servlet-mapping>
        <servlet-name>AddNumbers</servlet-name>
        <url-pattern>/addnumbers</url-pattern>
    </servlet-mapping>
</web-app>
```

# 10. MIME Attachment 메시지 전송

본 장에서는 MIME Attachment를 사용한 메시지 전송 방식에 대해서 설명한다.

## 10.1. 개요

웹 서비스를 구현하는 데 있어서 클라이언트 또는 서버로부터 웹 서비스의 파라미터 또는 리턴값은 SOAP 메시지의 Body 부분에 페이로드의 형태로 포함되어 있고 이 메시지가 네트워크를 통해 전송된다. 이때 그러한 파라미터, 리턴값이 사진 또는 음악 파일과 같은 크기가 큰 바이너리 데이터일 경우 그 데이터의 크기가 크면 클수록 효율은 떨어진다.

JAX-WS 웹 서비스는 MTOM(Message Transmission and Optimization Mechanism)과 XOP(XML Binary Optimized Packaging)를 통해 `xs:base64Binary` 또는 `xs:hexBinary`와 같은 element로 정의된 바이너리 데이터가 최적화되어 전송되도록 한다. 또한 `swaRef`라는 스키마를 지원하여 이 스키마로 정의된 XML element의 콘텐츠는 모두 MIME Attachment의 형태로 전송하도록 하는 기능을 제공한다.

JEUS 9 JAX-WS 웹 서비스는 `MessageContext` 프로퍼티를 사용하여 첨부 파일을 스트리밍 방식으로 송수신할 수 있다. 이 기능은 특히 대용량 첨부 파일을 수신할 때 유용하다.

## 10.2. MTOM/XOP

MTOM은 XOP와 함께 XML binary data such as `xs:base64Binary` 또는 `xs:hexBinary`와 같은 XML 바이너리 데이터가 네트워크에 어떻게 최적화되어 전송될 수 있는지에 대해 정의하고 있다.

`xs:base64Binary`와 같은 XML 타입은 SOAP Envelope에 포함되어 전송되기 마련인데 이러한 타입의 데이터, 이미지나 음악과 같이 데이터의 크기가 커지면 커질수록 효율성은 크게 떨어진다. 이러한 문제를 해결하기 위해 MTOM은 이러한 바이너리 데이터의 크기가 어떤 정해진 값보다 클 경우 MIME Attachment의 형태로 XOP 패키징하여 메시지를 전송한다. 이렇게 `xs:base64Binary` 또는 `xs:hexBinary` 타입의 element 값이 MIME Attachment의 형태로 XOP 패키징될 때 그 MIME Attachment의 Content-Type은 `xmime:expectedContentType` 속성에 지정해줄 수 있고 이는 JAXB에서 지원하는 타입 매핑의 적용을 받는다.

`xs:base64Binary` 또는 `xs:hexBinary` 타입의 스키마 element는 `xmime:expectedContentType` 속성과 함께 사용했을 때 JAXB에서 지원하는 타입 매핑을 적용받아 이 element 값의 크기가 어떤 상한선을 넘게 되었을 때 다음과 같은 MIME 타입으로 MIME Attachment에 포함된다. `xmime:expectedType` 속성을 사용하지 않으면 보통의 `byte[]`로 매핑되고 이 element 값의 크기가 어떤 상한선을 넘게 되면 Content-Type에 일반적인 `application/octet-stream`와 같은 값으로 MIME Attachment에 포함된다.

이러한 `xmime:expectedContentType`의 Java 타입으로의 JAXB 타입 매핑은 다음과 같다.

MIME 타입	Java 타입
image/gif	java.awt.Image
image/jpeg	java.awt.Image
text/plain	java.lang.String
text/xml or application/xml	javax.xml.transform.Source



MIME 타입	Java 타입
*/*	jakarta.activation.DataHandler

따라서 `<element name="image" type="base64Binary"/>`와 같은 element는 순수하게 `byte[]` 타입으로 매핑되지만, `<element name="image" type="base64Binary" xmlns:xmime="http://www.w3.org/2005/05/xmlmime" expectedContentType="image/jpeg"/>`와 같은 element는 `java.awt.Image` 타입으로 매핑된다.

### 10.2.1. 기본 동작

MTOM/XOP 환경을 이용할 수 있도록 WSDL 문서의 `wsdl:type`에 `xs:base64Binary` 또는 `xs:hexBinary`와 같은 타입으로 정의된 element가 `xmime:expectedContentType` 속성으로 정의된다. 정의된 속성은 **wsimport** 툴을 통해 JAXB의 특정 타입 매핑으로 서비스 인터페이스 및 Portable Artifact들이 생성되었을 때 클라이언트 및 서버에서는 MTOM/XOP 환경을 동작시킬 수 있다.

#### 서버에서의 MTOM 동작

서버에서 MTOM을 동작시키기 위해서는 서비스 Endpoint 구현 클래스에 `@jakarta.xml.ws.soap.MTOM` Annotation을 추가한다.

```
@jakarta.xml.ws.soap.MTOM
@WebService(endpointInterface = "com.tmax.mtom.Server")
public class ServerImpl implements Server {
    ...
}
```

다음과 같이 서비스 Endpoint 구현 클래스에 `@BindingType` Annotation을 추가시켜 MTOM을 동작시키는 방법도 있다.

```
@BindingType(value=jakarta.xml.ws.SOAPBinding.SOAP11HTTP_MTOM_BINDING)
@BindingType(value=jakarta.xml.ws.SOAPBinding.SOAP12HTTP_MTOM_BINDING)
```

#### 클라이언트에서의 MTOM 동작

클라이언트에서 MTOM을 동작시키기 위해서는 서비스로부터 프록시(proxy) 또는 디스패치(dispatch) 객체를 `jakarta.xml.ws.soap.MTOMFeature`라는 파라미터를 통해 얻는 방법이다.

```
Server port = new ServerService().getServerPort(new MTOMFeature());
jakarta.xml.ws.Service.createDispatch(..., new jakarta.xml.ws.soap.MTOMFeature())
```

다음은 프록시 또는 디스패치가 MTOM이 설정되어있는지 확인하는 메소드이다.

```

Server port = new ServerService.getServerPort();
SOAPBinding binding = (SOAPBinding)((BindingProvider)port).getBinding();
boolean mtomEnabled = binding.isMTOMEnabled();
binding.setMTOMEnabled(true);

```

## 10.2.2. Attachment 바이너리 데이터 크기 설정

JAX-WS 웹 서비스는 `xs:base64Binary` 그리고 `xs:hexBinary` 타입의 element에 대한 Java 오브젝트에 대해서 그 크기가 1Kbyte가 넘으면 클라이언트 또는 서버로부터 전송할 때 MIME Attachment의 형태로 XOP 인코딩되어 메시지에 패키징된다. 그렇지 않으면 SOAP 메시지 안에 포함한다.

XOP 인코딩되어 메시지에 패키징될 때 원래의 element에는 `<xop:Include href=...>`와 같은 값이 설정되는데 이 element의 href 속성에 들어가는 값은 Attachment의 Content-Id 값이다. 또한 스키마의 `xs:base64Binary`, `xs:hexBinary`로 정의된 타입의 `xmime:expectedContentTypes` 속성의 값이 Attachment의 Content-Type 값으로 설정된다.

Attachment 형태로 취급될 바이너리 데이터의 크기를 설정하는 방법은 다음과 같다.

- 서버

@MTOM Annotation로 설정한다.

```

@jakarta.xml.ws.soap.MTOM(threshold=3000)
@WebService(endpointInterface = "com.tmax.mtom.Server")
public class ServerImpl implements Server {
    ...
}

```

- 클라이언트

MTOMFeature 클래스를 통해 설정한다.

```

Server port = new ServerService().getServerPort(new MTOMFeature(3000));
jakarta.xml.ws.Service.createDispatch(..., new jakarta.xml.ws.soap.MTOMFeature())

```

다음은 실제 WSDL의 `wsdl:type`의 스키마 예제이다.

```

<element name="Detail" type="types:DetailType"/>
<complexType name="DetailType">
    <sequence>
        <element name="Photo" type="base64Binary"/>
        <element name="image" type="base64Binary"
            xmime:expectedContentTypes="image/jpeg"/>
    </sequence>
</complexType>

```

실제 WSDL의 `wsdl:type` 스키마가 위와 같은 모습일 경우에 네트워크에 전송되는 메시지는 다음과 같다.



```

        targetNamespace="http://tmaxsoft.com/mtom/data"
        xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
        elementFormDefault="qualified">
        <complexType name="DetailType">
            <sequence>
<element name="image" type="base64Binary"
            xmime:expectedContentTypes="image/jpeg" />
            </sequence>
        </complexType>
        <element name="Detail" type="types:DetailType" />
        <element name="DetailResponse" type="types:DetailType" />
    </schema>
</wsdl:types>
<wsdl:message name="HelloIn">
    <wsdl:part name="data" element="types:Detail" />
</wsdl:message>
<wsdl:message name="HelloOut">
    <wsdl:part name="data" element="types:DetailResponse" />
</wsdl:message>
<wsdl:portType name="Hello">
    <wsdl:operation name="Detail">
        <wsdl:input message="tns:HelloIn" />
        <wsdl:output message="tns:HelloOut" />
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="HelloBinding" type="tns:Hello">
    <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="Detail">
        <soap:operation />
        <wsdl:input>
            <soap:body use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="HelloService">
    <wsdl:port name="HelloPort" binding="tns:HelloBinding">
        <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

위와 같이 DetailType element의 스키마를 보면 순서대로 base64Binary 타입의 element인 image가 정의되어 있고 이는 MTOM에서 관리할 수 있다. 또한 image 타입은 xmime:expectedContentTypes="image/jpeg"으로 속성이 선언되어 있어 MTOM에서 Attachment로 다루는 경우 그 Content-Type이 image/jpeg가 된다.

## 10.2.4. MTOM/XOP 예제 실행

본 절에서는 지금까지 구현한 클래스들 및 기타 설정 파일들을 사용해서 핸들러 프레임워크를 실행하는 방법에 대해서 설명한다.

다음과 같이 MTOM를 설정한 서비스를 생성하여 JEUS에 deploy한다.

```
$ ant deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 빌드한다. 클라이언트에서 wsimport의 과정을 거치므로 서비스의 deploy가 모두 완료되었을 때 클라이언트의 구성이 가능하다.

다음과 같이 MTOM을 설정한 클라이언트를 생성하고 서비스를 호출한다. 콘솔에 명령어를 입력하면 클라이언트와 서비스가 정상적으로 메시지를 주고받는 것을 확인할 수 있다.

```
$ ant run

...

Host: localhost:8088
Content-length: 4848
Content-type: multipart/related;type="application/xop+xml";boundary="uuid:23ba19
3c-bd1a-4323-abdd-339bf5c05cd1";start-info="text/xml"
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2,
*/*; q=.2
Connection: keep-alive
Soapaction:
User-agent: JAX-WS RI 3.0 - JEUS 9
--uuid:23ba193c-bd1a-4323-abdd-339bf5c05cd1
Content-Type: application/xop+xml;charset=utf-8;type="text/xml"
Content-Transfer-Encoding: binary

<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envel
ope/"><S:Body><Detail xmlns="http://tmaxsoft.com/mtom/data"><image><Include xmlns
="http://www.w3.org/2004/08/xop/include" href="cid:dc0fa852-3d46-4bac-9ad4-889d
59c6198a@example.jaxws.sun.com"/></image></Detail></S:Body></S:Envelope>
--uuid:23ba193c-bd1a-4323-abdd-339bf5c05cd1
Content-Type: image/jpeg
Content-Id: <dc0fa852-3d46-4bac-9ad4-889d59c6198a@example.jaxws.sun.com>
Content-Transfer-Encoding: binary

    JFIF

...
```

## 10.3. swaRef

JAX-WS 웹 서비스는 WSDL의 wsdl:type 안에 element를 wsi:swaRef라는 스키마 타입으로 정의하고 웹 서비스를 구성하면 전송되는 메시지는 element 값을 MIME Attachment의 형태로 포함시킨다.

실제 그 SOAP Body 메시지의 element 안에는 그 Attachment로의 레퍼런스를 취하는 형태를 지원한다. 이러한 wsi:swaRef 스키마의 element는 jakarta.activation.DataHandler의 형태의 Java 클래스로 매핑된다.

### 10.3.1. swaRef 사용법

wsi:swaRef 타입의 XML element는 DataHandler Java 클래스로 매핑되며 실제 네트워크에서는 Attachment의

형태로 전송된다.

예를 들어 다음과 같은 XML 스키마가 WSDL에 정의되어 있다고 가정한다.

```
<element name="claimForm" type="ws:swaRef"
  xmlns:ws="http://ws-i.org/profiles/basic/1.1/xsd"/>
```

XML 스키마가 정의된 WSDL 문서를 통해 wsimport 툴로 웹 서비스를 구성하여 네트워크로 메시지를 전송할 때 메시지는 다음과 같다.

```
Content-Type: Multipart/Related;
start-info="text/xml";
type="application/xop+xml";
boundary="-----_Part_4_32542424.1118953563492"
Content-Length: 1193
SOAPAction: ""

-----_Part_5_32550604.1118953563502

Content-Type: application/xop+xml;
type="text/xml"; charset=utf-8
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body>
<claimForm xmlns="http://example.org/mtom/data">
cid:b0a597fd-5ef7-4f0c-9d85-6666239f1d25@example.jaxws.sun.com
</claimForm>
</soapenv:Body>
</soapenv:Envelope>

-----_Part_5_32550604.1118953563502

Content-Type: application/xml
Content-ID:
<b0a597fd-5ef7-4f0c-9d85-6666239f1d25@example.jaxws.sun.com>
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=" https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/application_9.xsd"
  version="9">
<display-name>Simple example of application</display-name>
<description>Simple example</description>
<module>
  <ejb>ejb1.jar</ejb>
</module>
<module>
  <ejb>ejb2.jar</ejb>
</module>
<module>
  <web>
    <web-uri>web.war</web-uri>
    <context-root>web</context-root>
  </web>
</module>
```

### 10.3.2. swaRef 예제

swaRef가 적용된 WSDL 파일의 예제는 다음과 같다.

swaRef가 적용된 WSDL 파일의 예제 : <hello.wsdl>

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:types="http://tmaxsoft.com/swaref/data"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:tns="http://tmaxsoft.com/swaref"
  targetNamespace="http://tmaxsoft.com/swaref" name="swaref">
  <wsdl:types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://tmaxsoft.com/swaref/data"
      xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
      elementFormDefault="qualified"
      xmlns:ref="http://ws-i.org/profiles/basic/1.1/xsd">
      <import namespace="http://ws-i.org/profiles/basic/1.1/xsd"
        schemaLocation="wsi-swa.xsd" />
    <element name="claimForm" type="ref:swaRef" />
    <element name="claimFormResponse" type="ref:swaRef" />
  </schema>
</wsdl:types>
<wsdl:message name="claimFormIn">
  <wsdl:part name="data" element="types:claimForm" />
</wsdl:message>
<wsdl:message name="claimFormOut">
  <wsdl:part name="data" element="types:claimFormResponse" />
</wsdl:message>
<wsdl:portType name="Hello">
  <wsdl:operation name="claimForm">
    <wsdl:input message="tns:claimFormIn" />
    <wsdl:output message="tns:claimFormOut" />
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="HelloBinding" type="tns:Hello">
  <soap:binding style="document"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <wsdl:operation name="claimForm">
    <soap:operation />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="HelloService">
  <wsdl:port name="HelloPort" binding="tns:HelloBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL" />
  </wsdl:port>
</wsdl:service>
</definitions>
```

```
</wsdl:service>
</wsdl:definitions>
```

위와 같이 claimForm와 claimFormResponse element는 ref:swaRef로 외부 스키마인 wsi-swa.xsd를 이용하여 swaRef의 데이터 타입으로 정의되어 있으므로 이후 네트워크로 메시지가 전송될 때 Attachment의 형태로 전송된다.

### 10.3.3. swaRef 예제 실행

본 절에서는 지금까지 구현한 클래스들 및 기타 설정 파일들을 가지고 핸들러 프레임워크를 실행하는 방법에 대해서 설명한다.

다음과 같이 swaRef를 설정한 서비스를 생성하여 JEUS에 deploy한다.

```
$ ant build deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 빌드한다. 클라이언트에서 wsimport의 과정을 거치므로 서비스의 Deploy가 모두 완료되었을 때 클라이언트의 구성이 가능하다.

다음과 같이 swaRef를 설정한 클라이언트를 생성하고 서비스를 호출한다. 콘솔에서 명령어를 입력하면 클라이언트와 서비스가 정상적으로 메시지를 주고받는 것을 확인할 수 있다.

```
$ ant run
...

Host: localhost:8088
Content-length: 2672
Content-type: multipart/related; type="text/xml"; boundary="uuid:26973efe-2e29-436a-8fce-3fa58b6fd91f"
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Soapaction:
User-agent: JAX-WS RI 3.0 - JEUS 9
--uuid:26973efe-2e29-436a-8fce-3fa58b6fd91f
Content-Type: text/xml

<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><claimForm xmlns="http://tmaxsoft.com/swaref/data">cid:40b15647-3c0b-4449-90ad-29b1667e940b@example.jaxws.sun.com</claimForm></S:Body></S:Envelope>
--uuid:26973efe-2e29-436a-8fce-3fa58b6fd91f
Content-Id:<40b15647-3c0b-4449-90ad-29b1667e940b@example.jaxws.sun.com>
Content-Type: text/xml
Content-Transfer-Encoding: binary

<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:types="http://tmaxsoft.com/swaref/data"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
```



```
xmlns:tns="http://tmaxsoft.com/swaref"
targetNamespace="http://tmaxsoft.com/swaref" name="swaref">
...
```

## 10.4. 스트리밍 방식으로 첨부 파일을 처리하는 방법

JEUS 9 JAX-WS 웹 서비스는 첨부 파일을 포함하는 SOAP 메시지를 송수신할 때 첨부 파일을 메모리로 읽어 처리하기 보다는 스트리밍(Streaming) 방식으로 송수신할 수 있다. 이 기능은 첨부 파일이 대용량일 때 웹 서비스 Endpoint 호출 성능을 향상시킨다. 특히, 첨부 파일이 JVM Heap 크기를 넘는 경우 OutOfMemory Error 없이 첨부 파일 송수신을 가능하게 한다.

본 절에서는 웹 서비스 클라이언트가 Outbound SOAP 메시지에 포함된 첨부 파일을 스트리밍 방식으로 전송하는 방법에 대해서 설명한다.

웹 서비스 클라이언트가 Chunked transfer-encoding을 사용하여 Outbound 요청 메시지를 전송하도록 설정한다. 서비스 포트(port)에 대하여 웹 서비스 오퍼레이션(operation)을 호출하기 전에(즉, Outbound 요청 메시지를 전송하기 전에) 서비스 포트로부터 얻은 요청 MessageContext에 다음의 프로퍼티를 설정한다.

- Property Key : "com.sun.xml.ws.transport.http.client.streaming.chunk.size"
- Property Value : chunked data size

이 프로퍼티는 JAX-WS HTTP Transport가 HTTP 요청을 Chunked Streaming 방식으로 전송한다. JEUS 9 서버는 Chunked Streaming 방식을 지원한다.

스트리밍 방식으로 첨부 파일을 처리 : <AttachmentApp.java>

```
Hello port = new HelloService().getHelloPort(
    new jeus.webservices.jaxws.api.transport.http.AttachmentFeature());

Map<String, Object> reqCnt = ((BindingProvider)port).getRequestContext();
reqCnt.put("com.sun.xml.ws.transport.http.client.streaming.chunk.size",
    new Integer(4*1024));
```



모든 HTTP 서버가 이 방식을 지원하지는 않는다는 것에 유의한다.

# 11. Fast Infoset 웹 서비스

본 장에서는 JEUS 웹 서비스가 새롭게 지원하는 Fast Infoset 웹 서비스와 이를 구현하고 사용하는 방법에 대해 설명한다.

## 11.1. 개요

Fast Infoset 표준은 XML 대신 사용할 수 있는 효과적인 XML Infoset들의 바이너리 형태를 명시하고 있다.

Fast Infoset(ITU-T Rec. X.891 | ISO/IEC 24824-1 Fast Infoset) 스펙은 다음에 의해 표준화되고 있다.

- ITU-T(통신장비 및 시스템의 조합 표준을 육성하기 위한 조직체)

<http://www.itu.int/ITU-T/studygroups/com17/index.asp>

- ISO(International Standards Organization)

[http://www.iso.org/iso/home/standards\\_development/list\\_of\\_iso\\_technical\\_committees.htm](http://www.iso.org/iso/home/standards_development/list_of_iso_technical_committees.htm)

XML 문서와 마찬가지로 이러한 XML Infoset들의 바이너리 형태 인스턴스를 가리켜 **Fast Infoset 문서**라고 한다. XML 문서와 Fast Infoset 문서는 실질적인 형태를 가지며 내부적으로 XML Infoset을 지니고 있다. 이러한 Fast Infoset 문서는 직렬화 또는 파싱(Parsing)하는 속도가 더 빠르며 크기면에서도 XML 문서보다 작다. 따라서 Fast Infoset 문서는 XML 문서의 프로세싱 속도와 크기가 이슈가 되는 경우 사용한다.

### Fast Infoset 설계

Fast Infoset 문서는 XML 문서와 마찬가지로 직렬화되거나 파싱(Parsing)될 수 있다.

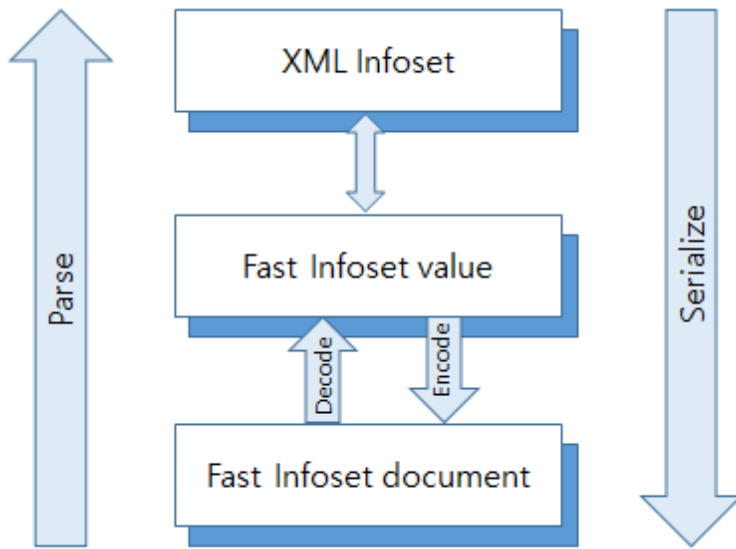
- 직렬화 과정

XML 문서로부터 얻은 DOM 문서나 SAX 이벤트에 해당하는 XML Infoset으로부터 Fast Infoset을 얻고 다시 이 Fast Infoset으로부터 Fast Infoset 문서를 얻는 과정이다.

- 파싱 과정

Fast Infoset 문서로부터 Fast Infoset을 얻고 다시 이 Fast Infoset으로부터 XML Infoset을 얻는 과정이다. Fast Infoset 스펙은 이러한 문서들의 효과적인 직렬화 혹은 파싱 과정 그리고 문서 크기의 축소에 대한 최적화를 보장한다.

다음은 이러한 직렬화 또는 파싱 과정을 나타낸 것이다.



Fast Infoset의 직렬화/파싱

## Fast Infoset 장점

Fast Infoset 문서는 XML 문서에 비해 파싱하는 속도와 직렬화하는 속도가 빠르며 문서의 크기가 더 작아진다는 장점이 있다.

이 외에도 다음과 같은 여러 가지 장점을 갖는다.

- XML 문서에서 사용되는 것과 같은 end 태그가 존재하지 않는다.
- escaping 문자 데이터가 존재하지 않는다.
- Fast Infoset 문서를 Fast Infoset으로 변환하는 디코더(decoder)는 길이를 미리 알고 있다는 장점이 있다.
- 반복되는 문자열을 인덱싱한다.
- 문자 공간(Namespace)과 같은 정보 또한 인덱싱한다.
- 바이너리 콘텐츠를 내포할 수 있다.
- 비슷한 어휘(Vocabulary)를 가진 문서들의 상태를 보존한다.

또한 바이너리 형태의 Fast Infoset 문서는 다음과 같은 장점을 갖는다.

- Element 아이템 또는 문자 정보들이 더 작은 비트들로 인코딩된다.
- Fast Infoset 인코더와 디코더는 잘 정의된 경계선(Boundary)에 힘입어 구현이 쉽고 효율이 뛰어나다.
- 인덱싱 정보를 통해 직렬화 또는 파싱하는 데 있어서 대상물의 크기가 작아지기 때문에 속도가 빠르다.
- 스트리밍 기능을 활용하기 적합하다.

## 11.2. Fast Infoset 사용

본 절에서는 Fast Infoset 기능을 사용하기 위한 방법인 Content Negotiation 기법에 대해 설명한다.

## 11.2.1. Content Negotiation

JAX-WS 웹 서비스에서 Fast Infoset 기능을 사용하기 위해서는 표준 HTTP Header인 Accept 또는 Content-Type에 Fast Infoset 기능의 사용 여부를 등록시키는 방법을 사용한다. 이러한 Fast Infoset의 사용 여부가 항상 웹 서비스를 호출하는 클라이언트에 의해 결정된다는 점에서 그리고 HTTP Header를 통해 이러한 결정이 이루어진다는 점에서 **Content Negotiation 기법**이라고 한다.

클라이언트가 HTTP Header에 정보를 삽입하는가에 따라 결정되는 웹 서비스의 Fast Infoset에서 클라이언트는 첫 번째 메시지를 전달할 때에는 XML로 인코딩된 SOAP 메시지를 전달한다. 이때 클라이언트가 HTTP Accept Header에 MIME 타입 application/fastinfoset의 정보를 포함시키면 그 다음부터 이루어지는 메시지 전달은 (서버가 Fast Infoset을 지원한다면) 모두 Fast Infoset으로 인코딩된 SOAP 메시지를 전송한다. 즉, 클라이언트가 XML로 인코딩된 SOAP 메시지를 HTTP Accept Header에 MIME 타입 application/fastinfoset로 전달하면 Fast Infoset이 가능한 서버는 Fast Infoset으로 인코딩된 메시지를 응답 메시지로써 전달한다. 만약 이후에 클라이언트가 같은 Stub Object를 통해 메시지를 전송한다면 클라이언트와 서버는 계속 이러한 Fast Infoset으로 인코딩된 메시지를 주고받게 된다. 이러한 방법을 **Negotiation Pessimistic 방식**이라고 한다.

JAX-WS 웹 서비스에서 Fast Infoset 기능을 사용하기 위해서는 다음의 방법으로 설정한다.

- 클라이언트의 속성을 설정한다.

```
((BindingProvider) stubOrDispatch).getRequestContext().put(
    com.sun.xml.ws.client.ContentNegotiation.PROPERTY, "pessimistic");
```

- 클라이언트가 동작하는 VM의 시스템 변수를 설정한다.

```
java -Dcom.sun.xml.ws.client.ContentNegotiation=pessimistic
```

## 11.3. Fast Infoset 예제

JAX-WS 웹 서비스에서 Fast Infoset 기능을 사용하기 위해서는 클라이언트 애플리케이션에서 클라이언트의 속성을 설정하는 방법과 클라이언트가 동작하는 VM의 시스템 변수를 설정하는 방법이 있다.

본 절에서는 클라이언트 애플리케이션을 통해 클라이언트의 속성을 설정하는 방법에 대해 설명한다.



서버에서는 다른 옵션을 통한 기능은 제공하지 않는다. Fast Infoset 속성은 모두 클라이언트에 의해서 결정된다.

다음은 Fast Infoset 속성을 위한 클라이언트 애플리케이션의 예제이다.

Fast Infoset 예제 : <AddNumbersClient.java>

```
public class AddNumbersClient {
    public static void main(String[] args) {
        AddNumbersImpl port = new AddNumbersImplService().getAddNumbersImplPort();
        ((BindingProvider) port).getRequestContext().put(
            ContentNegotiation.PROPERTY, "pessimistic");
    }
}
```

```

    int number1 = 10;
    int number2 = 20;

    System.out.println("#####");
    System.out.println("### JAX-WS Webservises examples - fastinfofet ###");
    System.out.println("#####");
    System.out.println("Testing Fast Infofet webservises...");
    int result = port.addNumbers(number1, number2);
    if (result == 30) {
        System.out.println("Success!");
    }
}
}
}

```

위와 같이 Fast Infofet 기능을 사용하기 위해서는 서비스 인터페이스로부터 얻은 프록시(Stub) 객체를 통해 pessimistic 속성을 Request 컨텍스트의 ContentNegotiation으로 설정한다.

## 11.4. Fast Infofet 웹 서비스 실행

본 절에서는 이전 절에서 구현한 클래스들 및 기타 설정 파일들을 이용하여 Fast Infofet 웹 서비스를 실행하는 방법을 설명한다. 기타 SEI의 구현 클래스 및 기타 설정 파일들은 앞 장에서 설명한 예제의 내용과 동일하다.

다음과 같이 Fast Infofet 웹 서비스를 설정한 서비스를 생성하여 JEUS에 deploy한다.

```
$ ant build deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 빌드하고 호출한다. 클라이언트에서 wsimport의 과정을 거치므로 서비스의 deploy가 모두 완료되었을 때 클라이언트의 구성이 가능하다.

다음과 같이 Fast Infofet 웹 서비스를 설정한 클라이언트를 생성하고 서비스에 호출한다.

```

$ ant run

...

run:
[java] #####
[java] ### JAX-WS Webservises examples - fastinfofet ###
[java] #####
[java] Testing Fast Infofet webservises...
[java] Success!

...

BUILD SUCCESSFUL

```

```
$ ant run
```

```
---[HTTP request]---
Host: localhost:8088
Content-length: 218
Content-type: text/xml; charset=utf-8
Accept: application/fastinfoset, text/xml, multipart/related, text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Soapaction: ""
User-agent: JAX-WS RI 3.0 - JEUS 9
<?xml version="1.0" ?><S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"><S:Body><ns2:addNumbers xmlns:ns2="http://server.fastinfoset/"><arg0>10</arg0><arg1>20</arg1></ns2:addNumbers></S:Body></S:Envelope>
-----
---[HTTP response 200]---
?
```

## 12. JAX-WS JMS 기반 전송

본 장에서는 웹 서비스에서 JAX-WS JMS 기반 전송(JMS Transport)을 사용하는 방법에 대해 설명한다.

### 12.1. 개요

기본적으로 웹 서비스 클라이언트 애플리케이션은 JEUS 웹 서비스를 호출할 때 전송 프로토콜로서 HTTP를 사용하지만, JEUS JAX-WS에서는 JMS 기반 전송을 사용하여 웹 서비스를 호출할 수 있다.

JAX-WS JMS 기반 전송은 JEUS MQ 서버의 ConnectionFactory와 Destination 설정을 사용한다. JAX-WS JMS 기반 전송을 사용하는 웹 서비스는 @jeus.webservices.jaxws.api.JMSWebService Annotation을 웹 서비스 Endpoint에 추가한다.

JAX-WS JMS 기반 전송을 사용하는 웹 서비스가 배치되면 공개된 WSDL에는 2개의 wsdl:port가 정의된다(HTTP 기반의 Port와 JMS 기반의 Port). 웹 서비스 클라이언트 애플리케이션은 사용하기 원하는 타입의 Port를 선택하여 웹 서비스를 호출할 수 있다.

### 12.2. JAX-WS JMS 기반 전송 설정

본 절에서는 개발자가 JAX-WS 웹 서비스와 JMS에 익숙하다고 가정하고, JAX-WS JMS 기반 전송을 설정하는 방법을 설명한다.

#### 12.2.1. JMS 서버 설정

JAX-WS JMS 기반 전송은 JMS 서버가 제공하는 ConnectionFactory와 Destination을 사용해서 동작한다. Destination으로 Queue 타입을 사용한다. JEUS 도메인 서버의 **domain.xml** 파일에 <connection-factory>, <destination>을 설정한다. 자세한 설정 방법은 "JEUS MQ 안내서"를 참고한다.

다음은 JAX-WS JMS 기반 전송 설정에 대한 예제이다.

JMS 서버 설정 : <domain.xml>

```
<jms-engine xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  ...
  <connection-factory>
    <type>queue</type>
    <name>QueueConnectionFactory</name>
    <service>jmstest</service>
  </connection-factory>
  ...
</jms-engine>
<jms-resource>
  ...
  <destination>
    <type>queue</type>
    <name>ExamplesQueue</name>
    <multiple-receiver>true</multiple-receiver>
  </destination>
</jms-resource>
</jeus>
```

```
...
</jms-resource>
```

### 12.2.2. 웹 서비스 작성

HTTP 대신에 JMS 기반 전송을 사용하기 위해서는 `@jeus.webservices.jaxws.api.JMSWebService` Annotation을 웹 서비스 Endpoint에 추가한다.

`@JMSWebService` Annotation이 설정된 웹 서비스 Endpoint는 다음과 같다.

웹 서비스 Endpoint : `<AddNumbersImpl.java>`

```
@WebService
@jeus.webservices.jaxws.api.JMSWebService(
    jndiConnectionFactoryName = "QueueConnectionFactory",
    destinationName = "ExamplesQueue",
    targetService = "AddNumbersImplService")
public class AddNumbersImpl {
    ...
}
```

`JMSWebService`의 속성은 다음과 같다.

속성	설명
<code>jndiConnectionFactoryName</code>	JMS 서버의 <code>ConnectionFactory</code> 이름으로 <code>domain.xml#&lt;connection-factory&gt;/&lt;name&gt;</code> 값과 일치해야 한다.
<code>destinationName</code>	JMS 서버의 <code>Destination</code> 이름으로 <code>domain.xml#&lt;destination&gt;/&lt;name&gt;</code> 값과 일치해야 한다.
<code>targetService</code>	Destination에 도착한 메시지를 전달할 서비스 Endpoint 구현체를 나타낸다.  fromJava 모델에서 속성이 명시되지 않으면, <code>wsdl:service</code> 이름을 <code>targetService</code> 속성값으로 사용한다.  fromWSDL 모델에서 JMS URI의 <code>targetService</code> 속성값과 일치해야 한다.

### 12.2.3. WSDL 설정

웹 서비스 Endpoint를 WSDL로부터 생성한다면 배치할 애플리케이션(WAR 또는 JAR)에 포함되는 WSDL에는 HTTP를 사용하는 `wsdl:port`와 JMS 기반 전송을 위한 `wsdl:port`가 설정되어 있어야 한다.

다음은 WSDL에 JAX-WS JMS 기반 전송을 위한 `wsdl:port`의 `soap:address`에 JMS URI를 설정하는 예제이다.

WSDL 설정 : `<AddNumbers.wsdl>`

```
<definitions name="AddNumbers" targetNamespace="urn:AddNumbers" ... >
    ...
    <service name="AddNumbersService">
        <port name="AddNumbersPort" binding="impl:AddNumbersBinding">
            <soap:address location="http://<host>:<port>/<context>/AddNumbersService" />
        </port>
    </service>
</definitions>
```



```

    </port>
    <port name="AddNumbersJMSPort" binding="impl:AddNumbersBinding">
        <soap:address
location="jms:jndi:ExamplesQueue?targetService=AddNumbersImplService&jndiConnectionFactoryName=QueueC
onnectionFactory" />
    </port>
    </service>
</definitions>

```

SOAP/JMS의 JMS URI은 'SOAP over Java Message Service 1.0' 표준에 따라 설정해야 한다.

JMS URI는 "jms" scheme을 사용하고, JMS destination을 JNDI로 조회할 destination 이름을 명시한다. JMS URI의 targetService 속성은 웹 서비스 Endpoint의 @JMSWebService#targetService 속성값과 일치해야 한다.

## 12.2.4. 웹 서비스 클라이언트 작성

JAX-WS JMS 기반 전송을 사용하려는 웹 서비스 클라이언트는 기존의 HTTP 기반의 포트 대신에 JMS 기반의 포트를 사용하여 웹 서비스를 호출할 수 있다.

웹 서비스의 공개된(Published) WSDL은 JMS 기반 전송을 위한 wsdl:port을 가지고 있다. 이 WSDL로부터 JEUS 웹 서비스의 wsimport 도구로 생성된 Service(extends jakarta.xml.ws.Service) 클래스는 기존의 HTTP 기반의 포트를 얻기 위한 메소드뿐만 아니라 JMS 기반의 포트를 얻기 위한 메소드도 가지고 있다.

웹 서비스 클라이언트 애플리케이션은 다음과 같이 사용할 타입의 포트를 선택하여 웹 서비스를 호출할 수 있다.

웹 서비스 클라이언트 : <AddNumbersClient.java>

```

//AddNumbersPortType port = new AddNumbersService().getAddNumbersPort();
AddNumbersPortType port = new AddNumbersService().getAddNumbersJMSPort();
...

```

## 13. 웹 서비스 정책

본 장에서는 웹 서비스의 정책 설정에 대한 기본적인 개념과 함께 간단한 시나리오에 대해 설명한다. 각각의 세부적인 웹 서비스 정책 설정에 관한 내용은 이후 각 장에서 자세히 설명한다.

### 13.1. 개요

JEUS 웹 서비스는 웹 서비스 정책(WS-Policy)을 지원한다. 웹 서비스 정책이란 어떤 웹 서비스가 가지고 있는 여러 가지 기능들(JEUS 웹 서비스에서는 WS-Addressing, WS-RM, WS-TX, WS-Security)의 정책을 노출시키기 위한 표준 명세이다.

JEUS 웹 서비스에서의 웹 서비스 정책 시나리오는 크게 서버 정책과 클라이언트 정책으로 구분할 수 있다. 서버는 WSDL을 통해 웹 서비스정책을 노출시킬 수 있고, 클라이언트는 웹 서비스의 정책에 맞는 기능을 자동으로 구성한다.

웹 서비스의 각각 정책 설정은 [웹 서비스 Addressing](#), [신뢰성 메시징 기술](#), [웹 서비스 트랜잭션](#), [웹 서비스 보안](#)의 내용을 참고한다.

### 13.2. 웹 서비스의 정책(WS-Policy)

본 절에서는 일반적인 웹 서비스의 정책(WS-Policy)에 대해 설명한다.

일반적인 웹 서비스 정책의 특징은 다음과 같다.

- 웹 서비스 정책 명세는 표현하기가 매우 유연하고 확장성이 있도록 설계되어 있다.
- 웹 서비스 정책은 하나 이상의 정책 전제(policy assertion)를 통해 표현된다.



자세한 표준 웹 서비스의 정책에 관한 스키마의 내용은 <http://schemas.xmlsoap.org/ws/2004/09/policy/ws-policy.xsd>를 참고한다.

- 웹 서비스 정책 프레임워크(Framework)\*

다음은 웹 서비스 정책 프레임워크에 대한 설명이다.

- 정책 컨테이너(Policy Container)

웹 서비스 정책 프레임워크에서 가장 핵심이 되는 주요 컴퍼넌트는 "Policy"라는 element로 표현된 정책 컨테이너이다. 이 element는 ID 값을 부여받을 수 있어 다른 곳에서 이를 참조하거나 재사용할 수 있다. 또한 이 element는 전제(assertion) 또는 전제들의 조합들로 구성된다. 이러한 전제들은 정책 연산자(Operator)들로 이루어진다.

- 정책 연산자(Operator)

웹 서비스 정책 명세는 2개의 연산자와 하나의 속성을 정의하고 있다.

- ExactlyOne Operator

하위 element에 전체 또는 연산자들이 여러 개인 경우 이 중 어느 하나만을 선택하여 정책으로 가지고 있겠다는 것을 나타내기 위한 연산자이다.

다음은 연산자 사용에 대한 예이다.

```
<wsp:Policy>
  <wsp:ExactlyOne>
    <wsse:SecurityToken>
      <wsse:Token>
        ...
      <wsse:
        ...
      </wsse:Token>
    </wsse:SecurityToken>
  </wsp:ExactlyOne>
</wsp:Policy>
```

- All Operator

하위 element에 전체 또는 연산자들이 여러 개인 경우 이 중 모든 것을 취합하여 정책으로 가지고 있겠다는 것을 나타내기 위한 연산자이다.

다음은 연산자 사용에 대한 예이다.

```
<wsp:Policy>
  <wsp:All>
    <wsse:SecurityToken>
      <wsse:Token>
        ...
      </wsse:Token>
    </wsse:SecurityToken>
  </wsp:All>
</wsp:Policy>
```

- Optional Operator

하위 element에 전체 또는 연산자들이 이 속성으로 선언되어 있을 때 선택적으로 취할 수 있음을 나타내기 위한 연산자이다.

다음은 연산자 사용에 대한 예이다.

```
<wsp:Policy>
  <wsse:Integrity wsp:optional="true">
    ...
  </wsse:Integrity>
</wsp:Policy>
```

## 13.3. 서버 정책 설정

서버의 웹 서비스 정책 설정 시나리오는 WSDL로부터 웹 서비스를 구성하는 시나리오와 Java 클래스로부터 웹 서비스를 구성하는 시나리오로 나눌 수 있다.

### 13.3.1. WSDL로부터 웹 서비스 구성

WSDL 문서로부터 웹 서비스 정책 설정이 적용된 웹 서비스를 구성하는 시나리오는 다음과 같다.

1. WSDL 문서를 작성한다.
2. WSDL 문서에 웹 서비스 정책 설정을 한다.
3. wsimport 툴을 통해 Java Bean 객체들을 생성한다.
4. 서비스 구현 클래스를 작성한다.
5. 패키징된 서비스를 JEUS 서버에 deploy한다.

#### 디렉터리 구조

WSDL 문서로부터 웹 서비스 정책 설정이 적용된 JEUS 웹 서비스를 패키징한다. 서버에 해당하는 웹 서비스를 WSDL 문서로부터 구성할 때의 디렉터리는 다음과 같다.

```
war_root
|- WEB-INF
    |- classes
    |   |- ... (SEI, JAX-WS artifacts, Handler, Validator)
    |- wsdl
        |- addnumbers.wsdl
```

### 13.3.2. Java 클래스로부터 웹 서비스 구성

Java 클래스로부터 웹 서비스의 정책이 적용된 웹 서비스를 생성하기 위해서는 다음과 같이 **wsgen** 툴에 추가적으로 **-policy** 기능을 이용하여 **wsit-endpoint.xml** 파일을 생성해야 한다.

```
$ wsgen fromjava.server.AddNumbersImpl -d web/WEB-INF -policy service-config.xml
```

다음은 service-config.xml의 실제 모습이다. 자세한 설정 내용에 관한 것은 이어지는 각각의 여러 가지 웹 서비스 기능을 위한 장에서 자세히 살펴보기로 한다. 여기에서는 아래에 강조한 부분들을 중점적으로 살펴보기로 한다.

Java 클래스로부터 웹 서비스 구성 : <service-config.xml>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <!-- Endpoint 전체에 적용하고자 하는 설정 -->
    <endpoint-policy-subject>
      <addressing-policy>
        <using-addressing>true</using-addressing>
      </addressing-policy>
    <!-- Endpoint의 어느 오퍼레이션(메소드)에 적용하고자 하는 설정 -->
    <operation-policy-subject>
      <!-- 이 설정은 아래의 오퍼레이션(메소드)에 대해 적용된다 -->
      <operation-java-name>addNumbers</operation-java-name>
      <!-- 클라이언트 요청 메시지에 적용하고자 하는 설정 -->
      <input-message-policy-subject>
```

```

        .....
        </input-message-policy-subject>
        <!-- 서버의 응답 메시지에 적용하고자 하는 설정 -->
        <output-message-policy-subject>
        .....
        </output-message-policy-subject>
    </operation-policy-subject>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

wsgen의 -policy 기능을 이용하여 웹 서비스를 위한 Java 클래스 및 wsit-endpoint.xml 파일을 얻었으면 다음과 같은 시나리오로 웹 서비스 정책이 적용된 웹 서비스를 생성한다.

1. 서비스 구현 클래스를 작성한다.
2. jeus-webservices-config.xsd 스키마를 통해 service-config.xml 파일을 구성한다.
3. 구성한 서비스 구현 클래스를 wsgen 툴을 통해 웹 서비스를 생성할 때 -policy 옵션을 이용하여 wsit-endpoint.xml 파일을 생성한다.
4. wsit-endpoint.xml 파일을 패키징될 WEB-INF 폴더 아래에 위치시킨다.
5. 패키징된 서비스를 JEUS 서버에 deploy한다.

## 디렉터리 구조

Java 클래스로부터 웹 서비스의 정책이 적용된 웹 서비스를 패키징한다. 서버에 해당하는 웹 서비스를 Java 클래스로부터 구성할 때의 디렉터리는 다음과 같다.

```

war_root
|- WEB-INF
    |- classes
    |   |- ... (SEI, JAX-WS artifacts, Handler, Validator)
    |- wsit-Endpoint.xml

```

## 13.4. 클라이언트 정책 설정

클라이언트의 웹 서비스 정책 설정은 웹 서비스 보안과 같은 시나리오 외에는 보통 필요하지 않다. JEUS 웹 서비스는 기본적으로 런타임에 원격 웹 서비스의 WSDL에 포함된 웹 서비스 정책 설정을 이해하고 자동으로 그 정책에 적합한 환경을 제공해주기 때문이다. 그러나 웹 서비스 보안과 같은 특정의 시나리오에서 추가적인 설정이 필요한 경우가 존재할 수 있다.

웹 서비스 정책이 설정되어 있는 웹 서비스에 대해 추가적인 설정이 필요한 경우 클라이언트를 구성하는 시나리오는 다음과 같다.

1. wsimport 툴을 통해 클라이언트 Java Bean 객체들을 구성한다.
2. 원격의 WSDL 문서를 접근 가능한 저장소에 wsit-client.xml이라는 이름으로 저장한다.



JEUS 웹 서비스는 런타임에 원격의 WSDL에 설정되어 있는 웹 서비스 정책 설정을 통해

클라이언트 환경을 제공해주기 때문에 원격의 WSDL에 설정되어 있는 웹 서비스 정책 설정에 대한 내용은 삭제해도 된다.

3. wsit-client.xml에 클라이언트에서 필요한 추가적인 웹 서비스 정책을 설정한다.
4. JAR 패키징할 경우 wsit-client.xml을 패키지될 classes/META-INF 디렉터리에 위치시킨다.  
  
WAR 패키징할 경우 wsit-client.xml 파일을 패키지될 WEB-INF 디렉터리 아래에 위치시킨다.
5. 패키징된 서비스를 JEUS 서버에 deploy한다.

## 디렉터리 구조

웹 서비스 정책 설정을 적용한 JEUS 웹 서비스를 패키징한다. 일반적으로 컨테이너에서 실행되는 웹 서비스 클라이언트는 다음과 같다.

```
war_root
|- WEB-INF
|   |- classes
|       |- ... (client classes, JAX-WS artifacts, Handler, Validator)
|       |- META-INF
|           |- wsit-client.xml
|- index.jsp
```

EJB 컨테이너에서 실행되는 웹 서비스 클라이언트 또는 독립 애플리케이션으로 실행되는 웹 서비스 클라이언트는 다음과 같다.

```
jar_root
|- classes
|   |- ... (client classes, JAX-WS artifacts, Handler, Validator)
|   |- META-INF
|       |- wsit-client.xml
```

## 14. 웹 서비스 Addressing

본 장에서는 트랜스포트에 독립적인 웹 서비스 Addressing과 그에 대한 간단한 예제를 통해 사용법을 설명한다.

### 14.1. 개요

웹 서비스 Addressing은 트랜스포트(transport)에 독립적으로 웹 서비스 메시지에 주소와 같은 정보를 표기하는 방식을 의미한다. 기본적인 웹 서비스는 Stateless의 메시지들의 교환이기 때문에 웹 서비스의 Addressing을 이용하면 상태를 알 수 있는 웹 서비스의 지원이 가능해진다. 실제로 웹 서비스 Addressing은 이후 장에서 설명할 WS-RM, WS-Security, WS-secure Conversation, WS-Trust 등 상당수 "WS-\*" 스펙들의 기반 기술이다.

웹 서비스 Addressing은 웹 서비스가 실행되고 있는 트랜스포트가 HTTP이든 SMTP이든 상관없이 동작하게 된다. 웹 서비스 Addressing을 설정하지 않고 애플리케이션 레벨에서 어떤 메시지에 정보를 표기하려면 메시지는 다음과 같은 모습을 나타낸다.

```
POST /AddNumbers/addnumbers HTTP 1.1/POST
Host: tmaxsoft.com
SOAPAction: http://tmaxsoft.com/AddNumbers/addnumbers

<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:tmax="http://tmaxsoft.com/">
  <S:Header>
    <tmax:MessageID>
      uuid:e197db59-0982-4c9c-9702-4234d204f7f4
    </tmax:MessageID>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

위와 같이 웹 서비스 Addressing을 설정하지 않은 메시지의 모습을 통해 알 수 있듯이 우선 이 메시지에 애플리케이션 나름으로 부여한 ID는 이 애플리케이션 레벨에서 사용하기 위해 설정한 것이므로 다른 애플리케이션에서의 재사용이 어렵다. 또한 트랜스포트를 HTTP에서 SMTP와 같은 것으로 변환한다면 트랜스포트 Header에 있는 정보 또한 그에 따른 매핑 규칙에 따라 변환해야 하는 등 많은 어려움을 수반한다. 따라서 이를 위해 W3C에서는 웹 서비스 Addressing을 위해 MAP(Message Addressing Properties)을 정의해서 SOAP 메시지나 WSDL 문서로의 바인딩(binding)을 규정해 놓았다.

다음은 메시지 레벨에서 이러한 MAP의 정보가 담긴 웹 서비스 Addressing을 설정한 메시지의 예제이다.

```
POST /AddNumbers/addnumbers HTTP 1.1/POST
Host: tmaxsoft.com
SOAPAction: http://tmaxsoft.com/AddNumbers/addnumbers

<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing/">
  <S:Header>
    <wsa:MessageID>
      uuid:e197db59-0982-4c9c-9702-4234d204f7f4
    </wsa:MessageID>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

```

</wsa:MessageID>
<wsa:To>
    http://tmaxsoft.com/AddNumbers/addnumbers
</wsa:To>
<wsa:Action>
    http://tmaxsoft.com/AddNumbers/addnumbers
</wsa:Action>
</S:Header>
<S:Body>
    ...
</S:Body>
</S:Envelope>

```

웹 서비스 Addressing을 설정한 메시지의 SOAP 메시지를 보면 <wsa:MessageID>, <wsa:To>, <wsa:Action> 같은 웹 서비스 Addressing에 관련된 element로 구성되어 있는 것을 알 수 있다. 이들은 MAP(Message Addressing Properties)의 SOAP 메시지의 바인딩 형태이다.

다음은 각 element에 대한 설명이다.

element	설명
<wsa:MessageID>	메시지를 유일하게 구분할 수 있도록 하는 절대 URI로 설정된 값이다.
<wsa:To>	메시지를 전달받도록 의도된 곳의 주소를 나타내기 위한 절대 URI로 설정된 값이다.
<wsa:Action>	메시지가 의미하는 바를 나타내기 위한 절대 URI로 설정된 값이다.

이와 같은 웹 서비스 Addressing 메시지는 모든 정보들이 트랜스포트 또는 애플리케이션과 독립적으로 프로세싱될 수 있는 고유의 형태로 들어 있는 것을 알 수 있다. 예를 들어 이러한 메시지가 HTTP가 아닌 SMTP와 같은 다른 트랜스포트로 전달되어야 한다면 <wsa:To> Header의 값을 `mailto:purchasing@example.com`과 같은 값으로 변경한다.

본 절에서는 서버와 클라이언트에서 웹 서비스 Addressing을 설정하는 방법과 예제와 실행방법에 대해서 설명한다.

## 14.2. 서버 설정

서버에서 웹 서비스 Addressing 설정은 Java 클래스로부터 설정하는 방법과 WSDL로부터 설정하는 방법이 있다. 본 절에서는 각 방법에 대해서 설명한다.

### 14.2.1. Java 클래스로부터 설정

Java 클래스로부터 웹 서비스를 구성할 때 웹 서비스 Addressing을 설정하기 위해서는 다음과 같이 단순히 `jakarta.jws.WebService` Annotation과 더불어 `jakarta.xml.ws.soap.Addressing` Annotation을 추가하면 모든 웹 서비스 Addressing 설정은 완료된다.

서버 웹 서비스 Addressing 설정 (1) : <AddnumbersImpl.java>

```

@Addressing
@WebService

```



```
public class AddNumbersImpl {
    ...
}
```

위와 같이 jakarta.xml.ws.soap.Addressing Annotation을 통해 웹 서비스 Addressing을 설정하면 이를 통해 서비스 Endpoint는 다음과 같은 동작을 수행하게 된다.

- 생성되는 WSDL 문서의 <wsdl:binding> element에 표준 <wsaw:UsingAddressing> element가 생성된다.
- 받아들이는 메시지의 모든 웹 서비스 Addressing Header들을 해석해서 올바른 문법인지를 검사한다.
- 올바른 문법이 아닌 경우 오류 메시지를 전달한다.
- <wsa:Action> Header 값이 해당 오퍼레이션에 대해 기대한 값과 일치하지 않을 경우 오류 메시지를 전달한다.
- 모든 전달되는 메시지에는 웹 서비스 Addressing Header에 대한 정보를 포함하고 있다.

또한 웹 서비스 Addressing은 SEI의 메소드(WSDL 문서의 operation element)에 대해 다음과 같이 Action MAP(Message Addressing Property)을 직접 설정할 수 있다.

서버 웹 서비스 Addressing 설정 (2) : <AddnumbersImpl.java>

```
@Addressing
@WebService
public class AddNumbersImpl {

    @Action(
        input = "http://tmaxsoft.com/input",
        output = "http://tmaxsoft.com/output",
        fault = {
            @FaultAction(className = AddNumbersException.class,
                value = "http://tmaxsoft.com/fault")
        }
    )
    public int addNumbers(int number1, int number2)
        throws AddNumbersException {

        ...

    }
}
```

이와 같이 설정된 메소드는 WSDL로 변환될 때 다음과 같은 WSDL 문서의 operation element에 바인딩된다.

서버 웹 서비스 Addressing 설정 (3) : <Addnumbers.wsdl>

```
...

<operation name="addNumbers">
    <input wsaw:Action="http://tmaxsoft.com/input"
        message="tns:addNumbers"/>
    <output wsaw:Action="http://tmaxsoft.com/output"
        message="tns:addNumbersResponse"/>
    <fault wsaw:Action="http://tmaxsoft.com/fault"
        message="tns:AddNumbersException"
        name="AddNumbersException"/>
</operation>
```

```

...
<binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
<wsaw:UsingAddressing />
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="addNumbers">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
  <operation name="addNumbers2">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
...

```

### 14.2.2. WSDL로부터 설정

WSDL로부터 웹 서비스를 구성할 때는 위의 Java 클래스로부터 WSDL을 생성했을 때와 마찬가지로 표준 확장 element인 `<wsaw:UsingAddressing>`을 이용한다. 이렇게 구성된 웹 서비스는 JEUS에서 제공하는 `wsimport` 툴을 사용하여 간단히 Addressing을 지원하는 웹 서비스를 생성할 수 있다.

## 14.3. 클라이언트 설정

클라이언트의 웹 서비스 Addressing 설정은 클라이언트를 생성하기 위해 참조하는 WSDL 문서의 `<wsaw:UsingAddressing>`이 포함되어 있는지 여부에 따라 **wsimport** 툴에 의해 자동으로 결정된다.

클라이언트가 별도로 웹 서비스 Addressing 기능을 애플리케이션 레벨에서 수행하고 있는 경우 웹 서비스 Addressing이 설정된 서비스로부터 클라이언트를 구성할 때 웹 서비스 Addressing 기능을 동작시키지 않을 수 있다.

다음은 클라이언트에서 웹 서비스의 Addressing 기능을 동작시키고 싶지 않을 경우 사용할 수 있는 방법이다.

```

new AddNumbersImplService().getAddNumbersImplPort(
    new jakarta.xml.ws.AddressingFeature(false));

```

위와 같이 클라이언트로부터 프록시(Endpoint Interface)를 얻을 때 `jakarta.xml.ws.AddressingFeature`의 값을 `false`로 설정하면 클라이언트는 메시지를 전송할 때 서비스에서 WSDL에 규정하고 있는 웹 서비스 Addressing

기능을 동작하지 않은 상태로 메시지를 전송하게 된다.

서비스의 WSDL에 웹 서비스의 Addressing 기능을 나타내는 `wsaw:UsingAddressing`을 포함하고 있지 않을 때 **wsimport** 툴을 통해 얻은 Portable Artifact들은 순수한 웹 서비스의 기능만을 수행한다. 클라이언트에서 별도로 웹 서비스의 Addressing 기능을 동작하고 싶은 경우가 있는데 이런 경우에는 다음과 같이 `jakarta.xml.ws.Service`가 지원하는 메소드들을 사용하거나 `jakarta.xml.ws.soap.AddressingFeature`를 사용한다.

```
<T> Dispatch<T> createDispatch(javax.xml.namespace.QName,
    java.lang.Class<T>, Service.Mode, WebServiceFeature...)
Dispatch<java.lang.Object> createDispatch(javax.xml.namespace.QName,
    jakarta.xml.bind.JAXBContext, Service.Mode, WebServiceFeature...)
<T> T getPort(java.lang.Class<T>, WebServiceFeature...)
<T> T getPort(javax.xml.namespace.QName, java.lang.Class<T>,
    WebServiceFeature...)

new AddNumbersImplService().getAddNumbersImplPort(
    new jakarta.xml.ws.AddressingFeature());
```

## 14.4. 예제

다음은 예제에서 다루는 서비스 Endpoint 구현 클래스의 예이다.

웹 서비스 Addressing : <AddNumbersImpl.java>

```
@Addressing
@WebService
public class AddNumbersImpl {

    @Resource
    WebServiceContext wsc;

    @Action(input = "http://tmaxsoft.com/input",
        output = "http://tmaxsoft.com/output")

    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }

    public int addNumbers2(int number1, int number2) {
        return number1 + number2;
    }
}
```

위와 같이 Addressing Annotation과 WSDL 문서의 operation element에 바인딩될 Action Annotation으로 웹 서비스 Addressing을 구성하고 있는 것을 확인할 수 있다.

## 14.5. 예제 실행

위와 같이 구성한 웹 서비스 Addressing을 설정한 Endpoint 클래스들 및 기타 파일들을 사용해서 서비스를 구성하여 JEUS 9에 deploy하는 방법은 다음과 같다.

```
$ ant deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 실행한다.

클라이언트 콘솔과 서버 콘솔 화면으로부터 다음과 같이 정상적으로 메시지를 전달하는 모습을 확인할 수 있다.

```
$ ant run

...

run:
  [java] #####
  [java] ### JAX-WS Webservicess examples - addressing ###
  [java] #####
  [java] basic name mapping result: 20
  [java] default name mapping result: 20

BUILD SUCCESSFUL

...

---[HTTP request]---
Host: localhost:8088
Content-length: 626
Content-type: text/xml; charset=utf-8
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg,
*, q=.2, */*; q=.2
Connection: keep-alive
Soapaction: "http://tmaxsoft.com/input"
User-agent: JAX-WS RI 3.0 - JEUS 9
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:To>http://localhost:8088/AddNumbers/addnumbers</wsa:To>
    <wsa:Action>http://tmaxsoft.com/input</wsa:Action>
    <wsa:ReplyTo xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Address>
        http://www.w3.org/2005/08/addressing/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>uuid:880f3891-d07b-4ad1-bbc1-8dce8f1aedef</wsa:MessageID>
  </S:Header>
  <S:Body>
    <ns2:addNumbers xmlns:ns2="http://server.wsaddressing/">
      <arg0>10</arg0><arg1>10</arg1>
    </ns2:addNumbers>
  </S:Body>
</S:Envelope>
---[HTTP response 200]---
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
```

```

xmlns:wsa="http://www.w3.org/2005/08/addressing">
<S:Header>
  <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
  <wsa:Action>http://tmaxsoft.com/output</wsa:Action>
  <wsa:MessageID>uuid:815cb296-a2f3-45df-80c5-5a2c1ca836ca</wsa:MessageID>
  <wsa:RelatesTo>uuid:880f3891-d07b-4ad1-bbc1-8dce8f1aedef</wsa:RelatesTo>
</S:Header>
<S:Body>
  <ns2:addNumbersResponse xmlns:ns2="http://server.wsaddressing/">
    <return>20</return>
  </ns2:addNumbersResponse>
</S:Body>
</S:Envelope>
-----

...

---[HTTP request]---
Host: localhost:8088
Content-length: 663
Content-type: text/xml; charset=utf-8
Accept: text/xml, multipart/related, text/html, image/gif, image/jpeg,
*; q=.2, */*; q=.2
Connection: keep-alive
Soapaction: "http://server.wsaddressing/AddNumbersImpl/addNumbers2Request"
User-agent: JAX-WS RI 3.0 - JEUS 9
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:To>http://localhost:8088/AddNumbers/addnumbers</wsa:To>
    <wsa:Action>
      http://server.wsaddressing/AddNumbersImpl/addNumbers2Request
    </wsa:Action>
    <wsa:ReplyTo xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Address>
        http://www.w3.org/2005/08/addressing/anonymous
      </wsa:Address>
    </wsa:ReplyTo>
    <wsa:MessageID>uuid:bf65c920-9129-495a-b9dc-8cb8efb9c2a6</wsa:MessageID>
  </S:Header>
  <S:Body>
    <ns2:addNumbers2 xmlns:ns2="http://server.wsaddressing/">
      <arg0>10</arg0><arg1>10</arg1>
    </ns2:addNumbers2>
  </S:Body>
</S:Envelope>
---[HTTP response 200]---
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <S:Header>
    <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
    <wsa:Action>
      http://server.wsaddressing/AddNumbersImpl/addNumbers2Response
    </wsa:Action>
    <wsa:MessageID>uuid:41975002-46e8-4219-89a6-ed6e72899fa8</wsa:MessageID>
    <wsa:RelatesTo>uuid:bf65c920-9129-495a-b9dc-8cb8efb9c2a6</wsa:RelatesTo>
  </S:Header>

```

```
<S:Body>
  <ns2:addNumbers2Response xmlns:ns2="http://server.wsaddressing/">
    <return>20</return>
  </ns2:addNumbers2Response>
</S:Body>
</S:Envelope>
-----
...
```

# 15. 신뢰성 메시징 기술

본 장에서는 신뢰성 메시징 기술에 대한 개념과 설정 방법에 대해서 설명한다.

## 15.1. 개요

신뢰성 메시징은 더욱더 신뢰성 있는 웹 서비스를 위한 양질의 서비스(Quality of Service, QOS)를 위함이다. 신뢰성은 하나의 지점으로부터 다른 하나의 지점으로 메시지를 전달할 수 있는 능력에 의해 측정될 수 있다. 이러한 신뢰성 메시징은 웹 서비스의 양 끝단에서 애플리케이션 메시지들의 전달을 보장하는 것이 주요 목적이다.

신뢰성 메시징 기술(WS-Reliable Messaging)은 메시지가 전달되지 못한 경우에 이를 복구하는 기술로 어떤 메시지들이 오직 한 번만 부과적으로는 순서에 맞게 웹 서비스의 양 끝단에 전달되는 것을 보장한다. 메시지가 중간에서 사라졌다면 메시지를 보낸 시스템은 이를 받아들이는 시스템으로부터 확인(Acknowledge) 메시지를 받을 때까지 그 메시지를 재전송하고, 받아들이는 시스템에서 메시지들이 순서에 맞게 들어오지 않았다면 이를 순서에 맞게 다시 수정해서 웹 서비스 Endpoint에 전달하게 된다.

이러한 신뢰성 메시징에 관한 스펙으로는 다음과 같은 것들이 있다.

- WS-Reliable Messaging
- WS-Coordination
- WS-AtomicTransactions

다음과 같은 문제점들이 발생할 경우에는 신뢰성 메시징 기술의 사용을 고려해야 한다.

- 통신 실패(Communication failure)가 발생하여 네트워크(Unavailable network)가 연결되지 않거나 연결의 끊김(Connection drop)이 현상이 발생하는 경우
- 애플리케이션 메시지가 전송 도중 사라지는 경우
- 순서에 맞게 전송되는 애플리케이션 메시지를 요구하는 경우인데 순서에 맞게 전송되지 않는 경우

신뢰성 메시징 기술을 사용할 것인지를 고려할 때 다음과 같은 장단점들 비교해봐야 한다.

- 신뢰성 메시징은 소스로부터 목적지까지 단 한 번만 전송되는 것을 의미한다. 만약 순서에 맞게 전송되는 옵션을 선택한 경우 그 메시지는 순서에 맞게 전송된다.
- 신뢰성 메시징으로 웹 서비스의 메시지를 전송하는 경우 이는 전체 웹 서비스의 성능을 저하시킬 수 있다. 특히 순서에 맞게 전송되는 옵션일 경우에 더 심하다.
- 신뢰성 메시징을 사용하지 않는 클라이언트는 이를 사용하는 웹 서비스와 상호 운용할 수 없다.

## 15.2. 서버 설정

서버의 WS-Reliable Messaging 설정은 WSDL로부터 구현할 수도 있고, Java 클래스로부터 설정할 수도 있다.

## 15.2.1. WSDL로부터 설정

WS-Reliable Messaging을 WSDL로부터 구현하기 위해서는 웹 서비스 Addressing의 경우와 마찬가지로 WSDL 문서에 웹 서비스 정책을 설정하고 **wsimport** 툴을 사용하여 웹 서비스를 생성한다.

WSDL 파일에 WS-Reliable Messaging을 웹 서비스 정책을 설정하는 방법은 다음과 같다.

- 기본적으로 PolicyAssertion에 다음을 설정한다.

```
<wsrm:RMAssertion xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy" />
```

- 순서에 맞는 전송(inorder)을 보장받기 위해서는 PolicyAssertion에 다음을 설정한다.

```
<rm:Ordered xmlns:rm="http://sun.com/2006/03/rm" />
```

다음은 이와 같이 구성한 WSDL 파일의 예이다.

서버의 WS-Reliable Messaging 설정 : <AddNumbers.wSDL>

```
<?xml version="1.0" ?>
<definitions name="AddNumbers" targetNamespace="http://example.org"
  xmlns:tns="http://example.org" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2005/02/rm/policy"
  xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
  <wsp:UsingPolicy />
  <wsp:Policy wsu:Id="AddNumbers_policy">
    <wsp:ExactlyOne>
      <wsp:All>
        <wsaw:UsingAddressing />
        <wsrm:RMAssertion>
          <wsrm:InactivityTimeout Milliseconds="600" />
          <wsrm:AcknowledgementInterval Milliseconds="200" />
        </wsrm:RMAssertion>
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <types>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
      elementFormDefault="qualified"
      targetNamespace="http://example.org">
      <element name="addNumbersResponse" type="tns:addNumbersResponse" />
      <complexType name="addNumbersResponse">
        <sequence>
          <element name="return" type="xsd:int" />
        </sequence>
      </complexType>
      <element name="addNumbers" type="tns:addNumbers" />
      <complexType name="addNumbers">
        <sequence>
```



```

        <element name="arg0" type="xsd:int" />
        <element name="arg1" type="xsd:int" />
    </sequence>
</complexType>
</xsd:schema>
</types>
<message name="addNumbers">
    <part name="parameters" element="tns:addNumbers" />
</message>
<message name="addNumbersResponse">
    <part name="result" element="tns:addNumbersResponse" />
</message>
<portType name="AddNumbersPortType">
    <operation name="addNumbers">
        <input message="tns:addNumbers" name="add" />
        <output message="tns:addNumbersResponse" name="addResponse" />
    </operation>
</portType>
<binding name="AddNumbersBinding" type="tns:AddNumbersPortType">
<wsp:PolicyReference URI="#AddNumbers_policy" />
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="document" />
    <operation name="addNumbers">
        <soap:operation soapAction="" />
        <input>
            <soap:body use="literal" />
        </input>
        <output>
            <soap:body use="literal" />
        </output>
    </operation>
</binding>
<service name="AddNumbersService">
    <port name="AddNumbersPort" binding="tns:AddNumbersBinding">
        <soap:address location="REPLACE_WITH_ACTUAL_URL" />
    </port>
</service>
</definitions>

```

## 15.2.2. Java 클래스로부터 설정

Java 클래스로부터 WS-Reliable Messaging을 설정하기 위해서는 다음과 같이 **wsgen** 툴의 **-policy** 기능을 이용하여 **wsit-endpoint.xml**을 먼저 얻어야 한다.

```
$ wsgen fromjava.server.AddNumbersImpl -d web/WEB-INF -policy service-config.xml
```

다음은 **service-config.xml**의 내용이다.

Java 클래스의 WS-Reliable Messaging 설정 : <service-config.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>

```

```

        <addressing-policy/>
        <rm-policy>
            <inactivityTimeout>600000</inactivityTimeout>
            <acknowledgementInterval>1000</acknowledgementInterval>
        </rm-policy>
    </endpoint-policy-subject>
</policy>
</web-services-config>

```

## 15.3. 클라이언트 설정

WS-Reliable Messaging을 위한 클라이언트의 추가적인 설정은 필요하지 않다. JEUS 웹 서비스는 클라이언트의 런타임에 원격 웹 서비스 WSDL의 WS-Reliable Messaging 정책을 해석해서 자동으로 WS-Reliable Messaging을 위한 환경을 제공한다.

## 15.4. 예제

Java 클래스로부터의 구현은 wsit-endpoint.xml이라는 DD 파일이 WAR 또는 EAR 패키징에서 WEB-INF 폴더에 추가된다는 사실을 제외하고 기본적인 JEUS 9 웹 서비스와 동일하다.

다음은 service-config.xml 파일의 예이다.

WS-Reliable Messaging 설정 : <service-config.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <addressing-policy/>
            <rm-policy>
                <inactivityTimeout>600000</inactivityTimeout>
                <acknowledgementInterval>1000</acknowledgementInterval>
            </rm-policy>
        </endpoint-policy-subject>
    </policy>
</web-services-config>

```

## 15.5. 예제 실행

위의 service-config.xml 파일을 가지고 wsgen을 통해 웹 서비스를 생성하기 위해서는 다음과 같이 입력한다.

```
$ ant build
```

웹 서비스가 정상적으로 생성되었다면 다음과 같은 구조를 갖게 된다.

```
web
```

```

|
+-- WEB-INF
    |
    +-- wsit-fromjava.server.AddNumbersImpl.xml
    +-- classes
        |
        +-- fromjava.server.AddNumbersImpl

```

wsit-fromjava.server.AddNumbersImpl.xml 파일은 WAR 또는 EAR 패키징에서 WEB-INF 폴더에 추가된 것이다.

서비스를 deploy하기 위해 다음과 같이 실행한다.

```
$ ant deploy
```

웹 서비스가 정상적으로 deploy되면 이를 사용하는 클라이언트를 다음과 같이 생성하고 서비스를 호출한다.

```

$ ant run

...

run:
[java] #####
[java] ### JAX-WS Webservicess examples - wsit ###
[java] #####
[java] Testing wsit webservicess...
[java] Success!

...

BUILD SUCCESSFUL

```

## 16. 웹 서비스 트랜잭션

본 장에서는 웹 서비스 트랜잭션에 대한 개념과 설정 방법에 대해서 설명한다.

### 16.1. 개요

트랜잭션은 신뢰성있는 분산 애플리케이션을 구현할 때 기본이 되는 개념이다. 트랜잭션은 애플리케이션에 참여하는 모든 참여인들(participants)이 서로 동의한 결과를 얻을 수 있도록 하는 메커니즘이다.

전통적으로 트랜잭션은 일명 "ACID"라는 다음 속성들을 포함하고 있다.

구분	설명
원자성(Atomicity)	성공적일 경우 모든 작동이 발생하고, 실패라면 어떤 작동도 발생하지 않는다.
일관성(Consistency)	애플리케이션은 완료될 때 유효한 상태 변환을 수행한다.
고립성(Isolated)	작동 결과는 성공적으로 끝나기 전까지 트랜잭션 외부에서는 공유되지 않는다.
지속성(Durability, 영속성)	일단 트랜잭션이 성공적으로 완료되면 변경으로 실패를 다시 복구할 수 있다.

웹 서비스 환경은 애플리케이션의 작동과 결과를 제어하기 위해 전통적인 트랜잭션 메커니즘에 의해 제공되는 동일한 코디네이션 작동이 필요하다. 또한 유연한 방식으로 다중 서비스에서 산출된 결과들의 프로세싱 코디네이션을 핸들링할 기능도 필요하다. 이것에는 좀 더 자유로운 형식의 트랜잭션이 필요하다.

JEUS 9 웹 서비스에서 다음의 트랜잭션 스펙을 지원한다.

- WS-Coordination
- WS-AtomicTransaction

위 트랜잭션 스펙은 협동하고 있는 작동들에 대한 WSDL 정의를 제공한다.

### 16.2. 서버 설정

서버에서 웹 서비스 트랜잭션은 WSDL로부터 설정하거나 Java 클래스로부터 설정할 수 있다.

#### 16.2.1. WSDL로부터 설정

웹 서비스 트랜잭션을 WSDL로부터 구현하려면 웹 서비스 Addressing의 경우와 마찬가지로 WSDL 문서에 웹 서비스 정책 설정을 하여 wsimport 툴을 사용하여 웹 서비스를 생성한다.

웹 서비스 정책 설정에 따라 WSDL 파일에 웹 서비스 트랜잭션을 알맞게 설정하려면 기본적으로 PolicyAssertion에 다음을 설정한다.

```
<wsat200410:ATAssertion xmlns:ns1="http://schemas.xmlsoap.org/ws/2002/12/policy" />
```

웹 서비스 트랜잭션 정책을 설정한 WSDL 파일은 다음과 같다.

WSDL 웹 서비스 트랜잭션 설정 : <AddNumbers.wSDL>

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="AddNumbersService" targetNamespace="http://server.fromwsdl/"
  xmlns:tns="http://server.fromwsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
  oasis-200401-wss-wssecurity-utility-1.0.xsd"
  xmlns:wsp="http://www.w3.org/ns/ws-policy"
  xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsp:Policy xmlns:wsat200410="http://schemas.xmlsoap.org/ws/2004/10/wsat"
    wsu:Id="AddNumbersPortBinding_addNumbers_WSAT_Policy" wsp:Name="">
    <wsat200410:ATAssertion
      xmlns:ns1="http://schemas.xmlsoap.org/ws/2002/12/policy"
      wsp:Optional="true" ns1:Optional="true" />
    </wsp:Policy>
    <types>...</types>
    <message>...</message>
    <portType>...</portType>

    <binding name="AddNumbersPortBinding" type="tns:AddNumbers">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
      <operation name="addNumbers">
        <wsp:PolicyReference URI="#AddNumbersPortBinding_addNumbers_WSAT_Policy" />
        <soap:operation soapAction="addNumbers" />
        <input>
          <wsp:PolicyReference URI="#AddNumbersPortBinding_addNumbers_WSAT_Policy" />
          <soap:body use="literal" />
        </input>
        <output>
          <wsp:PolicyReference URI="#AddNumbersPortBinding_addNumbers_WSAT_Policy" />
          <soap:body use="literal" />
        </output>
      </operation>
    </binding>
    <service>...</service>
  </definitions>
```

## 16.2.2. Java 클래스로부터 설정

Java 클래스로부터 웹 서비스 트랜잭션을 설정하려면 웹 서비스 Endpoint 구현체 클래스에 다음의 Annotation을 붙인다.

```
@com.sun.xml.ws.api.tx.at.Transactional(version=com.sun.xml.ws.api.tx.at.Transactional.Version.WSAT10)
```

다음은 Java 클래스로부터 웹 서비스 트랜잭션을 설정한 예제이다.

```
@WebService
@com.sun.xml.ws.api.tx.at.Transactional(
    version=com.sun.xml.ws.api.tx.at.Transactional.Version.WSAT10)
public class AddNumbersImpl {
    ...
}
```

## 16.3. 클라이언트 설정

웹 서비스 트랜잭션을 위한 클라이언트 사이드의 추가적인 설정은 필요하지 않다. JEUS 웹 서비스는 클라이언트의 런타임에 원격 웹 서비스 WSDL의 웹 서비스 트랜잭션 정책을 해석해서 자동으로 웹 서비스 트랜잭션을 위한 환경을 제공한다.

## 16.4. 코디네이터 서비스

웹 서비스 트랜잭션을 사용하기 위해서는 트랜잭션 참여자 간의 트랜잭션 액티비티를 조율하는 코디네이터 서비스를 deploy해야 한다. 코디네이터 서비스는 서버와 클라이언트에 모두 필요하다. 웹 서비스와 웹 서비스 클라이언트가 같은 서버에서 동작하면 하나의 코디네이터 서비스만 deploy한다.

코디네이터 서비스를 위한 wstx-services.ear은 다음 경로의 디렉터리에 위치한다.

```
JEUS_HOME/lib/systemapps
```

## 16.5. 웹 서비스 트랜잭션 예제

Java 클래스로부터의 구현은 @com.sun.xml.ws.api.tx.at.Transactional annotaion을 붙이는 것을 제외한 나머지 설정은 기본적인 JEUS 9 웹 서비스와 동일하다.

웹 서비스 클라이언트에서는 JTA(Java Transaction API)를 사용하여 프로그래밍한다.

웹 서비스 트랜잭션 : <AddNumbersClient.jsp>

```
InitialContext ctx = new InitialContext();
UserTransaction utx = (UserTransaction) ctx.lookup("java:comp/UserTransaction");

AddNumbersImplService service = new AddNumbersImplService();
AddNumbersImpl port = service.getAddNumbersImplPort();

utx.begin();

int result = port.addNumbers(number1, number2);

utx.commit();
```

# 17. 웹 서비스 보안

본 장에서는 전송 수준과 메시지 수준의 웹 서비스 보안에 대해 자세히 설명하고, JEUS 웹 서비스에서 이러한 보안을 적용하는 방법에 대해 설명한다.

## 17.1. 개요

웹 서비스에 보안을 적용하기 위해서는 전통적으로 다음과 같은 2가지 방식이 존재한다.

- 전송 수준 보안

SSL을 사용하여 클라이언트와 웹 서비스 사이의 연결 보안을 보장한다.

전송 수준 보안을 적용하려면 클라이언트와 JEUS 서버 간의 연결을 SSL을 사용하여 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. 또한 전송 수준 보안은 전체적인 메시지를 다루므로 메시지 일부의 보안 적용이 불가능하다.

- 메시지 수준 보안

SOAP 메시지를 전자 서명이나 암호화한다.

메시지 수준 보안은 SSL의 보안의 장점을 포함하면서 부가적인 유연성을 제공한다. 메시지 수준 보안은 메시지 전달 과정에서 하나 이상의 매개체가 존재하더라도 보안이 유지되는 End-to-End 보안이다. 연결 자체의 보안 유지보다는 SOAP 메시지 자체의 서명과 암호화를 의미한다. 또한 부분적인 서명과 암호화가 가능하다는 장점이 있다.

## 17.2. 전송 수준 보안

웹 서비스의 전송 수준 보안(Transport-level Security)은 웹 서비스 클라이언트 응용 프로그램과 웹 서비스 간의 연결을 SSL을 사용하여 안전하게 하는 것을 의미한다.

전체적인 절차는 다음과 같다.

1. JEUS 서버의 SSL을 설정한다.

웹 서비스 개발 부분은 추가 작업이 필요하지 않다. JEUS 서버에서의 SSL 설정은 "JEUS Web Engine 안내서"를 참고한다.

2. 다음의 과정으로 클라이언트 응용 프로그램의 SSL을 설정한다.

- a. 인증서를 가져온다(Internet Explorer 등을 통해 인증서를 로컬 디렉터리에 저장한다).
- b. 가져온 인증서를 Keystore에 저장한다.
- c. wsimport를 사용하여 WSDL로부터 Stub을 생성하려고 하거나 클라이언트에서 웹 서비스를 호출하기 위해 클라이언트를 실행할 때 시스템 프로퍼티 값을 다음과 같이 설정한다.

```
-Djavax.net.ssl.trustStore=keystore_name
```

```
-Djavax.net.ssl.trustStorePassword=keystore_password
```

d. wsimport 툴을 사용하기 위해 추가적으로 다음과 같은 환경변수 설정이 필요하다.

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

## 17.3. 메시지 수준 보안

다음은 JEUS 웹 서비스가 구현하고 있는 웹 서비스 보안에 관한 명세서이다.

- 웹 서비스 보안 정책(WS-Security Policy)
- 웹 서비스 보안(WS-Security)
- 웹 서비스 보안 대화(WS-SecureConversation)
- 웹 서비스 신뢰(WS-Trust)

본 절에서는 각각의 명세서들에 대해 그 의미와 시나리오에 대해 설명한다.

### 17.3.1. 웹 서비스 보안 정책

웹 서비스 보안 정책(WS-Security Policy) 명세는 어떻게 메시지가 보안화되어 이동되는지를 설명하는 전제(assertion)들을 정의한다. 이러한 전제들은 토큰, 암호화 기법, 사용되는 메커니즘, 전송 보안 등의 정책들을 포괄하는 유연성을 따른다. 이러한 웹 서비스 보안 정책은 웹 서비스 메시지 수준 보안을 구현할 때 핵심적인 요소이다.

JEUS 웹 서비스에서는 여러 가지 웹 서비스 메시지 수준 보안을 위한 시나리오별 예제들을 제공한다. 이러한 시나리오별 예제의 웹 서비스 보안 정책을 적용한 정책 파일들은 여러 가지 웹 서비스의 메시지 보안을 적용하는 데 도움이 될 것이다.

### 17.3.2. 웹 서비스 보안

웹 서비스 보안(WS-Security) 명세는 메시지의 XML 서명을 사용한 데이터 무결성과 XML 암호화를 사용한 데이터 비밀성 2가지를 실현하여 안전한 웹 서비스를 구현할 때 사용되는 확장된 SOAPElement들을 정의하고 있다.

다음은 웹 서비스 보안 명세를 사용한 웹 서비스 보안 시나리오이다.

- 사용자명 인증(Username Authentication)을 통한 대칭 바인딩의 인증 기능 강화
- 상호 인증 보안(Mutual Certificates Security)



- SSL을 통한 SAML(Security Assertions Mark-up Language) 인증

### 17.3.2.1. 사용자명 인증을 통한 대칭 바인딩의 인증 기능 강화

이 예제는 대칭 바인딩(Symmetric Binding)을 통한 메시지 보안 예제이다. 앞서 설명한 것과 마찬가지로 대칭 바인딩에서는 클라이언트와 서버가 암호화 키와 서명에 동일한 인증서 정보를 사용한다. 이 예제에서는 그 인증서 정보로써 서버의 인증서를 사용한다. 이때 클라이언트는 자신이 누구인지를 서버에 인증받을 방법이 별도로 존재하지 않기 때문에 부가적으로 사용자명 토큰을 사용한다.

다음은 사용자명 인증(Username Authentication)의 시나리오와 Keystore 설정에 대한 설명이다.

#### • 시나리오

다음은 사용자명 인증의 시나리오이다.

1. 클라이언트는 대칭 키를 생성한 후 이를 사용해 요청 메시지를 암호화 및 서명을 한다. 이때 서버의 공개 키를 사용하여 대칭 키를 함께 메시지에 담아 보낸다. 추가적으로 사용자명과 비밀번호 정보를 함께 보낸다.
2. 서버는 서버의 개인 키를 사용하여 대칭 키를 해독한 뒤 이를 사용하여 클라이언트의 요청 메시지를 복호화하고 서명을 검증한다. 추가적으로 사용자명과 비밀번호를 통해 클라이언트를 인증한다.
3. 서버는 다시 그 대칭 키를 통해 응답 메시지를 서명 및 암호화한다. 이때 대칭 키를 별도로 클라이언트로 전송하지는 않는다.
4. 클라이언트는 가지고 있던 대칭 키를 사용하여 서버의 응답 메시지를 해석하고 서명을 확인한다.

#### • Keystore 설정

다음은 Keystore 설정 정보이다.

구분	설명
클라이언트	서버의 공개 키가 담긴 Keystore이다.
서버	서버의 개인 키가 담긴 Keystore이다.

### 17.3.2.2. 상호 인증 보안

이 예제는 비대칭 바인딩, 상호 인증서를 통한 메시지 보안 예제이다. 대칭 바인딩과 달리 비대칭 바인딩에서는 클라이언트와 서버가 암호화 키와 서명에 서로 다른 인증서 정보를 사용한다. 서로 자신의 개인 키로 서명을 하고 인증서 정보를 넘겨줌으로써 서로 서명을 검증할 때 상대방이 누구인지를 인증할 수 있다.

다음은 상호 인증 보안 시나리오와 Keystore 설정에 대한 설명이다.

#### • 시나리오

다음은 상호 인증 보안(Mutual Certificates Security)의 시나리오이다.

1. 클라이언트는 클라이언트의 개인 키로 메시지에 서명을 하고 하나의 대칭 키를 생성해서 메시지를 암호화한 후, 그 대칭 키를 서버의 공개 키를 사용하여 암호화하여 메시지와 함께 서버에게 요청 메시지를 보낸다. 이때 인증 역할을 하게 될 클라이언트의 인증서를 함께 보낸다.
2. 서버는 서버의 개인 키를 사용하여 대칭 키를 꺼낸 후 암호화된 메시지를 해석한다. 인증서를 통해 인증을

하고 클라이언트의 공개 키로 서명을 검증한다.

3. 서버는 서버의 개인 키로 메시지에 서명을 하고 하나의 대칭 키를 생성해서 메시지를 암호화한 후 그 대칭 키를 클라이언트의 공개 키를 사용하여 암호화하며 메시지와 함께 클라이언트에게 응답 메시지를 보낸다. 이때 인증 역할을 하게 될 서버의 인증서는 함께 보내지 않는다.
4. 클라이언트는 클라이언트의 개인 키를 사용하여 대칭 키를 꺼낸후 암호화된 메시지를 해석한다. 서버의 공개 키로 서명을 확인한다.

#### • Keystore 설정

다음은 Keystore 설정 정보이다.

구분	설명
클라이언트	서버의 공개 키가 담긴 Keystore이다.
서버	서버의 개인 키가 담긴 Keystore이다.

### 17.3.2.3. SSL을 통한 SAML 인증

본 절에서는 SAML(Security Assertions Mark-up Language) 토큰을 SOAP 메시지 헤더에 포함시켜 보내는 방법에 대해 설명한다.

실제 웹 서비스에서 SAML 토큰을 사용한 서비스를 하려면 이러한 SAML 토큰을 생성하는 SAML 토큰 프레임워크가 필요하다. 여기서는 단순히 핸들러를 통해 SAML 토큰을 텍스트로 생성해서 보내는 예를 통해 어떻게 SAML 토큰을 SOAP 메시지 헤더에 포함시키는지에 대해 알 수 있다.

웹 서비스의 발전과 사용자의 활용 빈도가 높아짐에 따라 다양한 웹 애플리케이션 간에 보안 및 인증 정보를 교환할 필요성이 높아지고 이에 따른 교환 표준이 필요하다. 이와 같은 표준을 제공하기 위해 탄생한 언어가 SAML이다.

SAML은 다음과 같은 기능을 제공한다.

- 사용자 보안 정보에 대한 XML 포맷을 제공하고 이러한 정보를 요청 및 전송하기 위한 포맷을 제공한다.
- SOAP과 같은 프로토콜에서 이러한 메시지를 사용하는 방법을 정의한다.
- 웹 SSO와 같이 일반적인 특정 사용 사례에 대해 자세한 메시지 교환 방법을 지정한다.
- 사용자의 신원을 Export시키지 않고 사용자 속성을 결정하는 기능을 비롯하여 여러 가지 개인 정보 보호 메커니즘을 지원한다.
- UNIX, Microsoft Windows, X.509, LDAP, DCE, XCMML 등 널리 사용되는 기술에서 제공하는 포맷으로 ID 정보를 처리하는 방법을 제공한다.
- 메타 데이터 스키마를 수식화하여 참여하는 시스템에서 지원하는 SAML 옵션과 통신할 수 있는 기능을 제공한다.

다음은 SSL을 통한 SAML 인증을 사용하기 위한 시나리오이다.

#### • 시나리오

1. 클라이언트는 SAML 토큰을 SOAP 메시지 헤더에 포함하여 SSL 설정을 통해 메시지를 보낸다.
2. 서버는 SOAP 메시지 헤더에 포함된 SAML 토큰을 해석하고 응답 메시지를 보낸다.



모든 클라이언트와 서버의 메시지 교환은 JEUS 서버의 SSL 설정을 통해 이루어진다.

### 17.3.3. 웹 서비스 보안 대화

웹 서비스 보안 대화(Web Service Secure Conversation) 명세는 서비스 제공자와 클라이언트 간의 보안 컨텍스트 생성 및 공유를 정의한다. 보안 컨텍스트는 메시지를 교환할 때 생기는 간접비용(overhead)을 줄이기 위해 사용된다. 이 명세는 보다 나은 메시지 수준의 보안과 보다 효율적인 다중 메시지 교환을 제공하기 위한 표준을 정의한다.

이 명세를 따르고 있는 JEUS 웹 서비스는 다중 메시지 교환을 위한 보안 방식이 정의되는 기본 메커니즘을 제공하고 더욱 더 효율적인 키나 새로운 키 재료와 함께 컨텍스트가 수립되도록 한다. 이러한 접근은 전체적인 성능과 다음에 따르는 메시지 교환의 보안성을 향상시킨다.

대칭 바인딩에서는 메시지를 여러 번 주고받을 때마다 암호화 키를 매번 생성해야 한다. 따라서 메시지를 여러 번 주고받는 시나리오에서는 적합하지 않다. 이 예제에서는 보안 대화(Secure Conversation)를 통해 첫 메시지를 주고받기 전 Handshaking 과정을 통해 비대칭 바인딩으로 서로 SecureContext를 수립하고 이후 실제 메시지를 여러 번 주고받을 때는 수립한 SecureContext를 사용하여 대칭 바인딩과 같이 서명과 암호화에 사용하게 된다. 인증 과정은 처음 비대칭 바인딩을 통한 SecureContext 수립 과정에서 거치게 된다.

다음은 웹 서비스 보안 대화의 시나리오와 Keystore 설정에 대한 설명이다.

#### • 시나리오

다음은 웹 서비스 보안 대화의 시나리오이다.

1. 클라이언트는 자신의 인증서와 SecureContext 정보를 자신의 인증서를 사용해서 서명하고, 서버의 공개 키로 암호화하여 서버에 전송한다(비대칭 바인딩).
2. 서버는 자신의 개인 키로 클라이언트의 메시지를 복호화하고 클라이언트 서명을 검증하면서 인증 절차를 거친후 SecureContext 수립 메시지로 응답한다(비대칭 바인딩).
3. 이후 클라이언트와 서버는 메시지를 주고받을 때 이 SecureContext 정보를 통해 암호화 및 서명을 한다(대칭 바인딩).

#### • Keystore 설정

다음은 Keystore 설정 정보이다.

구분	설명
클라이언트	클라이언트의 개인 키와 서버의 공개 키가 담긴 Keystore이다.
서버	서버의 개인 키와 클라이언트의 공개 키가 담긴 Keystore이다.

### 17.3.4. 웹 서비스 신뢰

본 절에서는 WS-Trust를 인증에 관해 사용하는 예에 대해 살펴본다. 서로 다른 토큰, 즉 신뢰할 수 없는 관계의 인증서는 STS를 통해 SAML 토큰이라는 다른 형태로 변환되어 요청자를 신뢰할 수 없는 응답자가 STS를 통해 인증할 수 있다.

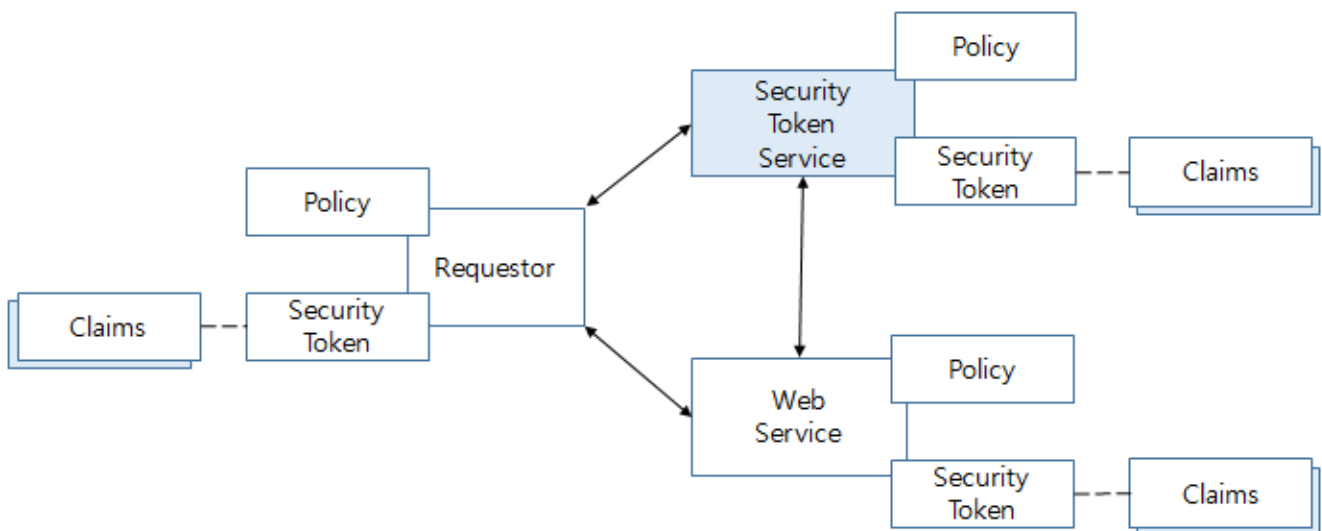
웹 서비스 보안을 사용하는 메시지 교환 방식에서는 서비스와 클라이언트 간에 보안 정보 공유를 위해 사용되는 보안 토큰에 대해 이미 합의되어야 하지만, 이러한 사전 합의가 이루어지지 않는 경우 메시지 교환 이전에 신뢰 관계(Trust Relationship)가 성립되어야 한다. 이러한 것을 웹 서비스 신뢰(Web Service Trust)라고 하며 JEUS 웹 서비스는 이를 지원하고 있다.

웹 서비스 신뢰는 서버와 클라이언트가 서로 다른 형태의 보안 토큰 또는 서로 다른 도메인의 보안 토큰을 사용해서 서로를 인증하지 못하는 상황일 때 사용할 수 있다. 이러한 클라이언트와 서버의 상이한 보안 토큰때문에 서로를 인증하지 못하는 상황은 클라이언트와 서버 사이에 각각의 인증 역할을 중계해 주는 또 다른 웹 서비스가 존재하여 해결한다. 이러한 웹 서비스를 통상적으로 **STS(Security Token Service)**라고 한다.

웹 서비스 신뢰는 다음의 관계를 갖는다.

- 클라이언트와 서버는 서로 직접적인 신뢰 관계가 없다.
- 클라이언트와 STS는 서로 신뢰 관계가 있다.
- STS와 서버는 서로 신뢰 관계가 있다.

다음 그림과 같이 Requestor는 웹 서비스와 통신하기 전에 웹 서비스가 자신을 인증할 수 있도록 하는 보안 토큰을 STS로부터 동적으로 부여받는다. 부여받은 보안 토큰을 통해 Requestor는 웹 서비스에게 실제로 보내려는 메시지에 STS로 부여받은 보안 토큰을 함께 실어 보내게 되며 웹 서비스는 Requestor를 인증(신뢰)할 수 있다.



WS-Trust

(출처: <http://docs.oasis-open.org/ws-sx/ws-trust/200512/ws-trust-1.3-os.html>)

다음은 웹 서비스 신뢰 시나리오와 Keystore 설정에 대한 설명이다.

#### • 시나리오

다음은 웹 서비스 신뢰의 시나리오이다.

1. 클라이언트는 자신의 인증서, 사용자명, 암호와 함께 실제 메시지를 전달할 서버에 대한 SAML 인증서 발급을 STS에 요청한다.
2. STS는 클라이언트를 인증한 후 SAML 인증서를 발급한다.
3. 클라이언트는 SAML 인증서와 함께 서버에 메시지를 보낸다.
4. 서버는 SAML 인증서를 통해 클라이언트를 신뢰하고 비즈니스 로직을 수행한다.

## • Keystore 설정

다음은 Keystore 설정 정보이다.

구분	설명
클라이언트	<ul style="list-style-type: none"><li>클라이언트의 개인 키와 서버의 공개 키가 담긴 Keystore이다.</li><li>클라이언트의 개인 키와 STS 서버의 공개 키가 담긴 Keystore이다.</li></ul>
서버	서버의 개인 키와 클라이언트의 공개 키가 담긴 Keystore이다.
STS 서버	서버의 개인 키와 클라이언트의 공개 키가 담긴 Keystore이다.

## 17.4. 메시지 수준 보안 설정

본 절에서는 JEUS 웹 서비스 메시지 수준 보안의 여러 가지 시나리오 예제에서 공통적으로 적용되는 설정법과 각각의 시나리오에 대한 설정법을 자세히 살펴볼 것이다. 실제 여러 가지 시나리오 예제에서 동작하는 것에 대해서는 샘플을 참고한다.



각 시나리오별 메시지 보안 설정 예제의 웹 서비스와 클라이언트의 비즈니스 로직은 모두 동일하다.

### 17.4.1. 공통 설정

JEUS 웹 서비스 메시지 수준 보안의 여러 가지 시나리오 예제에서 공통적으로 적용되는 설정은 서버와 클라이언트로 구분하여 설정한다.

#### 17.4.1.1. 서버 설정

다음은 'endpoint-policy-subject', 'operation-policy-subject', 'input(output)-message-policy-subject'에 공통적으로 들어가는 보안 설정들에 대한 설명이다.

##### • 'keystore', 'truststore' element의 설정

'security-policy' 하위의 element 중 하나인 'keystore', 'truststore' element는 그 'security-policy' element의 'security-binding'의 보안 설정에 따라 능동적으로 설정해야 한다.

'keystore-filename'에 들어가는 Keystore 파일에 대한 설정은 절대 경로 혹은 상대 경로를 포함한 실제 Keystore 파일의 위치를 지정한다. 상대 경로로 지정할 때는 서비스 구현 클래스가 포함된 classes 디렉터리 하위의 META-INF 디렉터리에 Keystore 파일이 위치해야 한다.

서버 메시지 수준 보안 설정 (1) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
```

```

.....
<xs:complexType name="keyTruststoreType">
  <xs:choice>
    <xs:element name="keystore-file" type="keyTruststoreFileType" />
    <xs:element name="keystore-callbackhandler" type="xs:string" />
  </xs:choice>
</xs:complexType>
<xs:complexType name="keyTruststoreFileType">
  <xs:sequence>
    <xs:element name="alias" type="xs:string" />
    <xs:element name="key-type" type="xs:string" minOccurs="0"
      default="JKS" />
    <xs:element name="keystore-password" type="xs:string" />
    <xs:element name="keystore-filename" type="xs:string" />
  </xs:sequence>
</xs:complexType>
.....
</xs:schema>

```

### • 'include-token' 속성 설정

'include-token' 속성은 메시지에 토큰을 포함할 것인지의 여부를 설정한다.

X509 토큰에서 사용되는 예는 다음과 같다.

서버 메시지 수준 보안 설정 (2) : <jeus-webservices-config.xsd>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
  targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
  xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
  attributeFormDefault="unqualified" version="7.0">
  .....
  <xs:complexType name="x509TokenType">
    <xs:sequence>
      <xs:element name="include-token" type="xs:boolean" minOccurs="0"
        default="false" />
    </xs:sequence>
  </xs:complexType>
  .....
</xs:schema>

```



그 밖의 다른 element 설정에 대한 자세한 설명은 'jeus-webservices-config.xsd' 스키마 및 "JEUS Reference 안내서"나 샘플 예제를 참고한다.

## Java 클래스로부터 웹 서비스 구현

Java 클래스로부터 웹 서비스 메시지 보안을 구현하기 위해서는 다음과 같이 **wsgen** 툴의 -policy 기능을 사용하여 웹 서비스를 구성한다.

```
$ wsgen fromjava.server.AddNumbersImpl -d web/WEB-INF -policy service-config.xml
```

-policy의 인자로 사용되는 service-config.xml을 구성하려면 service-config.xml을 위한 JEUS 보안 정책 스키마인 jeus-webservices-config.xsd에 대한 지식이 필요하다.

JEUS 보안 정책 스키마는 다음과 같이 'endpoint-policy-subject', 'operation-policy-subject', 'input(output)-message-policy-subject' element에 보안 설정을 할 수 있다.

서버 Java 클래스 메시지 수준 보안 설정 (1) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:element name="web-services-config" type="web-services-configType" />
    <xs:complexType name="web-services-configType">
        <xs:choice>
            .....
            <xs:element name="policy" type="policy-configType"
maxOccurs="unbounded" />
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="policy-configType">
        <xs:sequence>
            <xs:element name="endpoint-policy-subject"
type="endpointPolicySubjectType" maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="endpointPolicySubjectType">
        <xs:sequence>
            .....
            <xs:element name="security-policy"
type="endpointSecurityPolicyType" minOccurs="0" />
            <xs:element name="operation-policy-subject"
type="operationPolicySubjectType" minOccurs="0"
maxOccurs="unbounded" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="operationPolicySubjectType">
        <xs:sequence>
            <xs:element name="security-policy"
type="operationSecurityPolicyType" minOccurs="0" />
            .....
            <xs:element name="input-message-policy-subject"
type="messagePolicySubjectType" minOccurs="0" />
            <xs:element name="output-message-policy-subject"
type="messagePolicySubjectType" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="messagePolicySubjectType">
        <xs:sequence>
            <xs:element name="security-policy" type="messageSecurityPolicyType"
minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

이 3가지 element들의 보안 설정 내용은 다음과 같다.

- 'endpoint-policy-subject' element 보안 설정

서비스 구현 클래스를 단위로 보안 설정을 할 수 있다. 보안 바인딩(비대칭, 대칭) 설정 및 서명 및 암호화, 보안 토큰, WS-Security 버전 등 메시지 보안에 관련된 여러 가지를 설정할 수 있다.

서버 Java 클래스 메시지 수준 보안 설정 (2) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="endpointSecurityPolicyType">
        <xs:sequence>
            <xs:element name="security-binding" type="securityBindingType"
minOccurs="0" />
            <xs:element name="token" type="supportingTokenType" minOccurs="0" />
            <xs:element name="protection" type="protectionType" minOccurs="0" />
            .....
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

'endpoint-policy-subject' element 보안 설정은 서비스 구현 클래스 단위의 보안 설정으로 다음과 같이 크게 3가지로 구분할 수 있다.

- security-binding

다음과 같이 security-binding element 보안 설정은 'transport-binding', 'symmetric-binding', 'asymmetric-binding' 중 하나를 선택할 수 있다.

서버 Java 클래스 메시지 수준 보안 설정 (3) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="securityBindingType">
        <xs:sequence>
            <xs:choice>
                <xs:element name="transport-binding"
type="transportBindingType" />
                <xs:element name="symmetric-binding"
type="symmetricBindingType" />
                <xs:element name="asymmetric-binding"
type="asymmetricBindingType" />
            </xs:choice>
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```



```
</xs:schema>
```

#### ▪ 'transport-binding' element 보안 설정

transport-binding element의 보안 설정은 웹 서비스 보안(WS-Security) 명세 외에 다른 전송단, 예를 들어 HTTPS와 같은 것을 사용하여 메시지를 보호하겠다는 것을 나타낸다.

서버 Java 클래스 메시지 수준 보안 설정 (4) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0"> .....
  <xs:complexType name="transportBindingType">
    <xs:sequence>
      <xs:element name="transport-token" type="tokenType" />
      <xs:element name="algorithm-suite" type="algorithmSuiteType"
minOccurs="0" default="Basic128" />
      <xs:element name="layout" type="layoutType" minOccurs="0"
default="Lax" />
      <xs:element name="timestamp" type="xs:boolean" minOccurs="0"
default="true" />
    </xs:sequence>
  </xs:complexType>
  .....
</xs:schema>
```

#### ▪ 'symmetric-binding' element 보안 설정

symmetric-binding element의 보안 설정은 웹 서비스 보안(WS-Security) 명세를 사용하여 대칭 바인딩(Symmetric Binding)으로 메시지를 보호할 때 설정한다.

대칭 바인딩(Symmetric Binding)은 메시지를 암호화하거나 복호화할 때 사용되는 대칭 키(Symmetric Key)는 SOAP 메시지의 헤더 부분에 추가적인 element(EncryptedKey) 하위에 암호화된 상태로 메시지와 함께 전송되는데, 이 대칭 키를 암호화하거나 복호화할 때 사용되는 토큰이 같다는 것을 의미한다. 서명할 때도 서명을 검증할 때와 같은 토큰을 통하여 이루어진다는 것을 의미한다. 하위의 'protection-token' element는 추가적으로 메시지 암호화 및 서명에 사용되는 토큰이 동일함을 의미한다.

서버 Java 클래스 메시지 수준 보안 설정 (5) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0"> .....
  <xs:complexType name="symmetricBindingType">
    <xs:sequence>
      <xs:element name="protection-token" type="tokenType" />
      <xs:element name="algorithm-suite" type="algorithmSuiteType"
minOccurs="0" default="Basic128" />
      <xs:element name="layout" type="layoutType" minOccurs="0"
default="Lax" />
    </xs:sequence>
  </xs:complexType>
  .....
</xs:schema>
```

```

        default="Lax" />
        <xs:element name="timestamp" type="xs:boolean" minOccurs="0"
        default="true" />
        <xs:element name="encrypt-signature" type="xs:boolean"
        minOccurs="0" default="false" />
        <xs:element name="encrypt-before-siging" type="xs:boolean"
        minOccurs="0" default="false" />
    </xs:sequence>
</xs:complexType>
.....
</xs:schema>

```

#### ▪ 'asymmetric-binding' element 보안 설정

asymmetric-binding element의 보안 설정은 웹 서비스 보안(WS-Security) 명세를 사용하여 비대칭 바인딩(Asymmetric Binding)으로 메시지를 보호할 때 설정한다.

비대칭 바인딩(Asymmetric Binding)은 메시지를 암호화하거나 복호화하는 데 사용되는 대칭 키(symmetric key)는 SOAP 메시지의 헤더 부분에 추가적인 element(EncryptedKey) 하위에 암호화된 상태로 메시지와 함께 전송되는데, 이 대칭 키를 암호화하거나 복호화할 때 사용되는 토큰이 서로 다르다는 것을 의미한다. 서명을 할 때와 서명을 검증할 때 다른 토큰을 통해 이루어진다는 것을 의미한다.

서버 Java 클래스 메시지 수준 보안 설정 (6) : <jeus-webservices-config.xsd>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="asymmetricBindingType">
        <xs:sequence>
            <xs:element name="initiator-token" type="tokenType" />
            <xs:element name="recipient-token" type="tokenType" />
            <xs:element name="algorithm-suite" type="algorithmSuiteType"
            minOccurs="0" default="Basic128" />
            <xs:element name="layout" type="layoutType" minOccurs="0"
            default="Strict" />
            <xs:element name="timestamp" type="xs:boolean" minOccurs="0"
            default="true" />
            <xs:element name="encrypt-before-siging" type="xs:boolean"
            minOccurs="0" default="false" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>

```

다음은 하위의 element에 대한 설명이다.

element	설명
initiator-token	클라이언트가 메시지를 서명할 때 사용하는 토큰이자 서버로부터 받은 암호화된 메시지를 복호화할 때 사용하는 토큰임을 의미한다.  서버가 메시지를 암호화할 때 사용하는 토큰이자 클라이언트로부터 받은 메시지의 서명을 검증할 때 사용하는 토큰임을 의미한다.
recipient-token	서버가 메시지를 서명할 때 사용하는 토큰이자 클라이언트로부터 받은 암호화된 메시지를 복호화할 때 사용하는 토큰임을 의미한다.  클라이언트가 메시지를 암호화할 때 사용하는 토큰이자 서버로부터 받은 메시지의 서명을 검증할 때 사용하는 토큰임을 의미한다.

#### ◦ token

다음은 token element 보안 설정에 대한 예로 하위의 element 중 하나를 선택할 수 있다.

서버 Java 클래스 메시지 수준 보안 설정 (7) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="supportingTokenType">
        <xs:choice>
            <xs:element name="supporting-token" type="tokenType" />
            <xs:element name="signed-supporting-token" type="tokenType" />
            <xs:element name="endorsing-supporting-token" type="tokenType" />
            <xs:element name="signed-endorsing-supporting-token"
                type="tokenType" />
        </xs:choice>
    </xs:complexType>
    .....
</xs:schema>
```

다음은 하위의 element에 대한 설명이다.

element	설명
supporting-token	이곳에 설정한 토큰 값은 실제 메시지의 헤더 부분에 포함되며 다른 메시지 부분을 암호화하거나 서명할 때 사용할 수 있다.
signed-supporting-token	'supporting-token'과 같다. 추가적으로 이 토큰 또한 서명이 된다.
endorsing-supporting-token	서명을 다시 서명하는 데 사용되는 토큰을 설정할 때 사용한다. 'Signature' element를 전체를 서명한다.
signed-endorsing-supporting-token	'endorsing-supporting-token'과 같다. 추가적으로 이 토큰 또한 원래 서명을 통해 서명한다. 상호 서명하는 것을 의미한다.

#### ◦ protection

다음은 protection element 보안 설정에 대한 예로 하위의 element 중 하나를 선택할 수 있다.

서버 Java 클래스 메시지 수준 보안 설정 (8) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="protectionType">
        <xs:sequence>
            <xs:element name="signed-part" type="protectionPartType"
minOccurs="0" />
            <xs:element name="encrypted-part" type="protectionPartType"
minOccurs="0" />
            <xs:element name="signed-element" type="xs:string"
minOccurs="0" />
            <xs:element name="encrypted-element" type="xs:string"
minOccurs="0">
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

다음은 하위의 element에 대한 설명이다.

element	설명
signed-part	서명(검증)할 부분을 설정할 때 사용한다.
encrypted-part	암호화 혹은 복호화할 부분을 설정할 때 사용한다.
signed-element	서명(검증)할 부분을 설정할 때 사용한다. 특정 element를 부분 서명할 때 사용하면 유용하다.  'signed-element' element 설정하는 경우 'disable-streaming-security' element의 값을 'true'로 함께 설정해야 한다. 자세한 사항은 샘플을 참고한다.
encrypted-element	암호화 혹은 복호화할 부분을 설정할 때 사용한다. 특정 element를 부분 암호화할 때 사용하면 유용하다.  'encrypted-element' element를 설정하는 경우 'disable-streaming-security' element의 값을 'true'로 함께 설정해야 한다. 자세한 사항은 샘플을 참고한다.

#### • 'operation-policy-subject' element의 보안 설정

서비스 구현 클래스의 메소드 단위로 보안 설정을 할 수 있다. 서명 및 암호화, 이를 뒷받침하는 보안 토큰들을 추가로 설정할 수 있다.

'operation-policy-subject' element 보안 설정은 서비스 구현 클래스의 메소드 단위의 보안 설정으로 다음과 같이 크게 2가지로 구분할 수 있다.

서버 Java 클래스 메시지 수준 보안 설정 (9) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="operationSecurityPolicyType">
        <xs:sequence>
            <xs:element name="supporting-token" type="supportingTokenType"
minOccurs="0" />
            <xs:element name="protection" type="protectionType" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

다음은 하위의 element에 대한 설명이다.

element	설명
supporting-token	이 설정은 'endpoint-policy-subject' element의 'token' element와 같다. 단, 이 설정은 이 메소드에만 설정되는 값이다.
protection	이 설정은 'endpoint-policy-subject' element의 'protection' element와 같다. 단, 이 설정은 이 메소드에만 설정되는 값이다.

#### • 'input(output)-message-policy-subject' element의 보안 설정

서비스 구현 클래스 메소드의 매개변수 및 반환값 단위로 보안 설정을 한다. 서명 및 암호화와 이를 뒷받침하는 보안 토큰들을 추가로 설정할 수 있다.

'input(output)-message-policy-subject' element 보안 설정은 서비스 구현 클래스 메소드의 매개변수 및 반환값 단위 보안 설정으로 다음과 같이 크게 2가지로 구분할 수 있다.

서버 Java 클래스 메시지 수준 보안 설정 (10) : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns="http://www.tmaxsoft.com/xml/ns/jeus"
targetNamespace="http://www.tmaxsoft.com/xml/ns/jeus"
xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified" version="7.0">
    .....
    <xs:complexType name="messageSecurityPolicyType">
        <xs:sequence>
            <xs:element name="supporting-token" type="supportingTokenType"
minOccurs="0" />
            <xs:element name="protection" type="protectionType" minOccurs="0" />
        </xs:sequence>
    </xs:complexType>
    .....
</xs:schema>
```

다음은 하위의 element에 대한 설명이다.

element	설명
supporting-token	이 설정은 'endpoint-policy-subject' element의 'token' element와 같다. 단, 이 설정은 이 메소드의 매개변수 및 반환값에만 설정되는 값이다.
protection	이 설정은 'endpoint-policy-subject' element의 'protection' element와 같다. 단, 이 설정은 이 메소드의 매개변수 및 반환값에만 설정되는 값이다.

## WSDL로부터 웹 서비스 구현

WSDL로부터 구현하기 위해서는 WSDL 문서에 직접 메시지 보안 정책 설정을 하고 **wsimport** 툴을 사용하여 웹 서비스를 구성한다. WSDL로부터 구현하려면 WS-Policy의 하부 스펙인 WS-SecurityPolicy의 내용을 정확하게 이해하여 WSDL에 적용하면 간단히 웹 서비스 메시지 보안을 구현할 수 있다.



WS-SecurityPolicy의 내용은 [WS-SecurityPolicy 1.2 스펙 문서](#)에서 확인한다.

다음은 메시지 보안이 설정된 웹 서비스를 구현하기 위한 WSDL 예이다. Keystore에 대한 설정은 절대 경로 또는 상대 경로를 포함한 실제 Keystore 파일의 위치를 지정한다. 상대 경로로 지정할 때는 서비스 구현 클래스가 포함된 classes 디렉터리 하위의 META-INF 디렉터리에 Keystore 파일이 위치해야 한다.

WSDL 메시지 수준 보안 설정 : <jeus-webservices-config.xsd>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions
...
targetNamespace="http://tmax.com/" name="NewWebServiceService">
<ns1:Policy xmlns:ns1="http://schemas.xmlsoap.org/ws/2004/09/policy"
wsu:Id="NewWebServicePortBindingPolicy">
  <ns1:ExactlyOne>
    <ns1:All>
      <ns7:SymmetricBindingxmlns:ns7="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <ns1:Policy>
          <ns1:ExactlyOne>
            <ns1:All>
              <ns7:AlgorithmSuite>
                <ns1:Policy>
                  <ns1:ExactlyOne>
                    <ns1:All>
                      <ns7:TripleDes />
                    </ns1:All>
                  </ns1:ExactlyOne>
                </ns1:Policy>
              </ns7:AlgorithmSuite>
              <ns7:IncludeTimestamp />
              <ns7:Layout>
                <ns1:Policy>
                  <ns1:ExactlyOne>
                    <ns1:All>
                      <ns7:Strict />
                    </ns1:All>
                  </ns1:ExactlyOne>
                </ns1:Policy>
              </ns7:Layout>
            </ns1:All>
          </ns1:ExactlyOne>
        </ns1:Policy>
      </ns7:SymmetricBinding>
    </ns1:All>
  </ns1:ExactlyOne>
</ns1:Policy>
```

```

</ns7:Layout>
<ns7:OnlySignEntireHeadersAndBody />
<ns7:ProtectionToken>
  <ns1:Policy>
    <ns1:ExactlyOne>
      <ns1:All>
        <ns7:X509Tokenns7:IncludeToken=
"http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/Never">
          <ns1:Policy>
            <ns1:ExactlyOne>
              <ns1:All>
                <ns7:WssX509V3Token10/>
              </ns1:All>
            </ns1:ExactlyOne>
          </ns1:Policy>
        </ns7:X509Token>
      </ns1:All>
    </ns1:ExactlyOne>
  </ns1:Policy>
</ns7:ProtectionToken>
</ns1:All>
</ns1:ExactlyOne>
</ns1:Policy>
</ns7:SymmetricBinding>
<ns8:Wss11xmlns:ns8="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
  <ns1:Policy>
    <ns1:ExactlyOne>
      <ns1:All>
        <ns8:MustSupportRefEncryptedKey />
        <ns8:MustSupportRefIssuerSerial />
        <ns8:MustSupportRefThumbprint />
      </ns1:All>
    </ns1:ExactlyOne>
  </ns1:Policy>
</ns8:Wss11>
<sc:KeyStore wspp:visibility="private"
alias="xws-security-server"
storepass="changeit" type="JKS"
location="keystore.jks" />
<sc:TrustStore wspp:visibility="private"
peeralias="xws-security-client"
storepass="changeit" type="JKS"
location="cacerts.jks" />
<sc:ValidatorConfigurationxmlns:sc=
"http://schemas.sun.com/2006/03/wss/server">
  <sc:Validator name="usernameValidator"
classname="com.tmax.UsernamePasswordValidator" />
</sc:ValidatorConfiguration>
<ns9:UsingAddressingxmlns:ns9="http://www.w3.org/2006/05/addressing/wsd1" />
</ns1:All>
</ns1:ExactlyOne>
</ns1:Policy>
<ns10:Policy xmlns:ns10="http://schemas.xmlsoap.org/ws/2004/09/policy"
wsu:Id="NewWebServicePortBinding_add_Input_Policy">
  <ns10:ExactlyOne>
    <ns10:All>
      <ns11:EncryptedPartsxmlns:ns11=
"http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <ns11:Body />
      </ns11:EncryptedParts>
    </ns10:All>
  </ns10:ExactlyOne>
</ns10:Policy>

```

```

</ns11:EncryptedParts>
<ns12:SignedPartsxm1ns:ns12=
  "http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
  <ns12:Body />
  <ns12:Header Name="ReplyTo"
    Namespace="http://www.w3.org/2005/08/addressing" />
  <ns12:HeaderNamespace="http://www.w3.org/2005/08/addressing" Name="To"/>
  <ns12:Header Name="From"
    Namespace="http://www.w3.org/2005/08/addressing"/>
  <ns12:Header Name="MessageId"
    Namespace="http://www.w3.org/2005/08/addressing"/>
  <ns12:Header Name="FaultTo"
    Namespace="http://www.w3.org/2005/08/addressing"/>
  <ns12:Header Name="Action"
    Namespace="http://www.w3.org/2005/08/addressing"/>
  <ns12:Header Name="RelatesTo"
    Namespace="http://www.w3.org/2005/08/addressing"/>
</ns12:SignedParts>
<ns13:SignedSupportingTokensxm1ns:ns13=
  "http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
  <ns10:Policy>
    <ns10:ExactlyOne>
      <ns10:All>
        <ns13:UsernameToken ns13:
          IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/
          IncludeToken/AlwaysToRecipient">
          <ns10:Policy>
            <ns10:ExactlyOne>
              <ns10:All>
                <ns13:WssUsernameToken10/>
              </ns10:All>
            </ns10:ExactlyOne>
          </ns10:Policy>
        </ns13:UsernameToken>
      </ns10:All>
    </ns10:ExactlyOne>
  </ns10:Policy>
</ns13:SignedSupportingTokens>
</ns10:All>
</ns10:ExactlyOne>
</ns10:Policy>
<ns14:Policy xm1ns:ns14="http://schemas.xmlsoap.org/ws/2004/09/policy"
  wsu:Id="NewWebServicePortBinding_add_Output_Policy">
  <ns14:ExactlyOne>
    <ns14:All>
      <ns15:EncryptedPartsxm1ns:ns15="http://schemas.xmlsoap.org/ws/
        2005/07/securitypolicy">
        <ns15:Body />
      </ns15:EncryptedParts>
      <ns16:SignedPartsxm1ns:ns16="http://schemas.xmlsoap.org/ws/
        2005/07/securitypolicy">
        <ns16:Body />
        <ns16:Header Name="ReplyTo"
          Namespace="http://www.w3.org/2005/08/addressing"/>
        <ns16:HeaderNamespace="http://www.w3.org/2005/08/addressing"
          Name="To"/>
        <ns16:Header Name="From"
          Namespace="http://www.w3.org/2005/08/addressing"/>
        <ns16:Header Name="MessageId"

```



```

        Namespace="http://www.w3.org/2005/08/addressing"/>
    <ns16:Header Name="FaultTo"
        Namespace="http://www.w3.org/2005/08/addressing"/>
    <ns16:Header Name="Action"
        Namespace="http://www.w3.org/2005/08/addressing"/>
    <ns16:Header Name="RelatesTo"
        Namespace="http://www.w3.org/2005/08/addressing"/>
    </ns16:SignedParts>
</ns14:All>
</ns14:ExactlyOne>
</ns14:Policy>
<types>
    ...
</types>
<message name="add">
    ...
</message>
<portType name="NewWebService">
    ...
</portType>
<binding name="NewWebServicePortBinding" type="tns:NewWebService">
    <ns17:PolicyReference xmlns:ns17="http://schemas.xmlsoap.org/ws/2004/09/policy"
        URI="#NewWebServicePortBindingPolicy" />
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
    <operation name="add">
        <soap:operation soapAction="" />
        <input>
            <ns18:PolicyReference xmlns:ns18="http://schemas.xmlsoap.org/ws/2004/09/policy"
                URI="#NewWebServicePortBinding_add_Input_Policy" />
            <soap:body use="literal" />
        </input>
        <output>
            <ns19:PolicyReference xmlns:ns19="http://schemas.xmlsoap.org/ws/2004/09/policy"
                URI="#NewWebServicePortBinding_add_Output_Policy" />
            <soap:body use="literal" />
        </output>
    </operation>
</binding>
<service name="NewWebServiceService">
    <port name="NewWebServicePort" binding="tns:NewWebServicePortBinding">
        <soap:address location="http://localhost:8088/usernameAuthentication_war/
            NewWebServiceService"/>
    </port>
</service>
</definitions>

```

### 17.4.1.2. 클라이언트 설정

메시지 보안이 설정된 웹 서비스의 클라이언트는 wsit-client.xml 파일을 통해 다음과 같이 부가적으로 Keystore 및 Callback 핸들러 등을 설정해야 한다. 그 밖의 다른 메시지 보안 설정은 서버의 WSDL에 설정된 메시지 보안 정책을 런타임에 해석하여 메시지 보안 환경을 자동으로 구성한다.

Keystore에 대한 설정은 절대 경로 또는 상대 경로를 포함한 실제 Keystore 파일의 위치를 지정한다. 상대 경로로 지정할 때는 서비스 구현 클래스가 포함된 classes 디렉터리 하위의 META-INF 디렉터리에 Keystore 파일이 위치해야 한다.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
...
targetNamespace="http://server.fromjava/"
  <wsp:UsingPolicy />
  <wsp:Policy wsu:Id="TmaxBP0">
    <wsp:ExactlyOne>
      <wsp:All>
        <CallbackHandlerConfiguration
          xmlns="http://schemas.sun.com/2006/03/wss/client">
          <CallbackHandler default="user_jeus" name="usernameHandler" />
          <CallbackHandler default="password_jeus"
            name="passwordHandler" />
        </CallbackHandlerConfiguration>
        <TrustStore location="cacerts.jks" peeralias="xws-security-server"
          storepass="changeit" type="JKS"
          xmlns:ns0="http://java.sun.com/xml/ns/wsit/policy"
          ns0:visibility="private"
          xmlns="http://schemas.sun.com/2006/03/wss/client" />
      </wsp:All>
    </wsp:ExactlyOne>
  </wsp:Policy>
  <types>
    ...
  </types>
  <message name="addNumbers">
    ...
  </message>
  <portType name="AddNumbersImpl">
    ...
  </portType>
  <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    <wsp:PolicyReference URI="#TmaxBP0" />
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http" />
    <operation name="addNumbers">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <service name="AddNumbersImplService">
    <port binding="tns:AddNumbersImplPortBinding" name="AddNumbersImplPort">
      <soap:address
        location="http://localhost:8088/DocLitEchoService/AddNumbersImplService" />
    </port>
  </service>
</definitions>

```

## 17.4.2. 사용자명 인증을 통한 대칭 바인딩의 인증 기능 강화

본 절에서는 서버와 클라이언트에서 사용자명 인증을 통한 대칭 바인딩의 인증 기능 강화 방법에 대해서 설명한다.

### 17.4.2.1. 서버 설정

서버 웹 서비스의 설정은 service-config.xml 파일을 작성하고 사용자명의 유효자(Validator) 클래스를 작성한다.

#### 웹 서비스 설정 파일 작성

다음은 웹 서비스 설정 파일 작성에 대한 예이다.

웹 서비스 설정 파일 : <service-config.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <endpoint-policy-subject>
      <security-policy>
        <security-binding>
          <symmetric-binding>
            <protection-token>
              <x509-token>
                <include-token>true</include-token>
              </x509-token>
            </protection-token>
          </symmetric-binding>
        </security-binding>
      </security-policy>
    </endpoint-policy-subject>
  </policy>
  <token>
    <signed-supporting-token>
      <username-token>
        <username-password-validator>
          fromjava.server.UsernamePasswordValidator
        </username-password-validator>
        <include-token>true</include-token>
      </username-token>
    </signed-supporting-token>
  </token>
  <protection>
    <signed-part>...</signed-part>
    <encrypted-part>...</encrypted-part>
  </protection>
  <wss-version>11</wss-version>
  <keystore>...</keystore>
</web-services-config>
```

#### 사용자명의 유효자(Validator) 클래스 작성

com.sun.xml.wss.impl.callback.PasswordValidationCallback.PasswordValidator을 구현하여 username token을 사용해서 접근한 사용자의 username과 password를 validate한다.

사용자명의 유효자 클래스 : <UsernamePasswordValidator.java>

```
public class UsernamePasswordValidator implements
    PasswordValidationCallback.PasswordValidator {

    public boolean validate(PasswordValidationCallback.Request request)
        throws PasswordValidationCallback.PasswordValidationException {
        ...
        return false;
    }

    private boolean validateUserFromDB(String username, String password) {
        ...
        return false;
    }
}
```

#### 17.4.2.2. 클라이언트 설정

클라이언트 웹 서비스의 설정은 추가적으로 wsit-client.xml에 Keystore(Truststore) 설정과 함께 사용자명 핸들러를 설정해야 한다.

#### Keystore(Truststore) 설정 및 사용자명 핸들러의 설정

wsit-client.xml 파일에 다음과 같이 설정한다.

Keystore(Truststore) 설정 및 사용자명 핸들러의 설정 : <wsit-client.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    ...
    targetNamespace="http://server.fromjava/"
    <wsp:UsingPolicy />
    <wsp:Policy wsu:Id="TmaxBP0">
        <wsp:ExactlyOne>
            <wsp:All>
                <CallbackHandlerConfiguration
                    xmlns="http://schemas.sun.com/2006/03/wss/client">
                    <CallbackHandler default="user_jeus" name="usernameHandler" />
                    <CallbackHandler default="password_jeus"
                        name="passwordHandler" />
                </CallbackHandlerConfiguration>
                <TrustStore location="cacerts.jks" peeralias="xws-security-server"
                    storepass="changeit" type="JKS"
                    xmlns:ns0="http://java.sun.com/xml/ns/wsdl/policy"
                    ns0:visibility="private"
                    xmlns="http://schemas.sun.com/2006/03/wss/client" />
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    ...
    <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
        <wsp:PolicyReference URI="#TmaxBP0" />
    </binding>
    ...
</definitions>
```

```
...
</definitions>
```

## 사용자명 핸들러 클래스 작성

javax.security.auth.callback.CallbackHandler을 구현하여 각각의 Callback을 CallbackHandler에 전달하면 여러 가지 설정을 해 줄 수 있다. 전달받은 Callback의 종류에 맞추어 정책에 따른 설정을 한다.

위 예제에서는 NameCallback과 PasswordCallback을 전달받아 Callback에 username과 password를 설정하는 CallbackHandler를 구성한다.

다음의 예제에서는 클래스 파일 내에서 username과 password를 미리 입력해 놓았지만 경우에 따라서 다양한 reader객체를 사용하여 사용자로부터 직접 입력받을 수도 있다.

사용자명 핸들러 클래스 : <UsernamePasswordCallbackHandler.java>

```
public class UsernamePasswordCallbackHandler implements CallbackHandler {

    public void handle(Callback[] callbacks) throws IOException,
        UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            Callback callback = callbacks[i];
            if (callback instanceof NameCallback) {
                ((NameCallback) callback).setName("user_jeus");
            } else if (callback instanceof PasswordCallback) {
                ((PasswordCallback) callback).setPassword((
                    new String("password_jeus")).toCharArray());
            } else {
                throw new UnsupportedCallbackException(callback,
                    "Unknow callback for username or password");
            }
        }
    }
}
```

## 17.4.3. 상호 인증 보안

본 절에서는 서버와 클라이언트에서 상호 인증 보안(Mutual Certificates Security)을 설정하는 방법에 대해 설명한다.

### 17.4.3.1. 서버 설정

서버의 웹 서비스 설정은 service-config.xml 설정 파일로 구성한다.

서버 상호 인증 보안 설정 : <service-config.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <security-policy>
```

```

        <security-binding>
            <asymmetric-binding>
                <initiator-token>
                    <x509-token>
                        <include-token>true</include-token>
                    </x509-token>
                </initiator-token>
                <recipient-token>
                    <x509-token>
                        <include-token>false</include-token>
                    </x509-token>
                </recipient-token>
            </asymmetric-binding>
        </security-binding>
    </protection>
        <signed-part>...</signed-part>
        <encrypted-part>...</encrypted-part>
    </protection>
    <keystore>...</keystore>
    <truststore>...</truststore>
</security-policy>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

### 17.4.3.2. 클라이언트 설정

클라이언트의 웹 서비스 설정은 추가적으로 wsit-client.xml에 Keystore(Truststore) 설정을 해야 한다.

클라이언트 상호 인증 보안 설정 : <wsit-client.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
...
targetNamespace="http://server.fromjava/">
    <wsp:UsingPolicy/>
    <wsp:Policy wsu:Id="TmaxBP0">
        <wsp:ExactlyOne>
            <wsp:All>
                <KeyStore
                    alias="xws-security-client"
                    location="keystore.jks"
                    storepass="changeit" type="JKS"
                    xmlns:ns0="http://java.sun.com/xml/ns/wsit/policy"
                    ns0:visibility="private"
                    xmlns="http://schemas.sun.com/2006/03/wss/client"/>
                <TrustStore
                    location="cacerts.jks"
                    peeralias="xws-security-server"
                    storepass="changeit"
                    type="JKS"
                    xmlns:ns0="http://java.sun.com/xml/ns/wsit/policy"
                    ns0:visibility="private"
                    xmlns="http://schemas.sun.com/2006/03/wss/client"/>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>

```

```

...
<binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
  <wsp:PolicyReference URI="#TmaxBP0"/>
  ...
</binding>
...
</definitions>

```

## 17.4.4. SSL을 통한 SAML 인증

본 절에서는 서버와 클라이언트에서 SSL을 통한 SAML 인증을 설정하는 방법에 대해서 설명한다.

### 17.4.4.1. 서버 설정

서버의 웹 서비스 설정은 service-config.xml 설정 파일로 구성한다.

서버 SSL을 통한 SAML 인증 설정 : <service-config.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <endpoint-policy-subject>
      <security-policy>
        <security-binding>
          <transport-binding>
            <transport-token>
              <https-token/>
            </transport-token>
          </transport-binding>
        </security-binding>
      </security-policy>
    </endpoint-policy-subject>
    <operation-policy-subject>
      <operation-java-name>addNumbers</operation-java-name>
      <input-message-policy-subject>
        <security-policy>
          <supporting-token>
            <signed-supporting-token>
              <saml-token>
                </saml-token>
            </signed-supporting-token>
          </supporting-token>
        </security-policy>
      </input-message-policy-subject>
    </operation-policy-subject>
  </endpoint-policy-subject>
</policy>
</web-services-config>

```

### 17.4.4.2. 클라이언트 설정

클라이언트의 웹 서비스 설정은 추가적으로 wsit-client.xml에 SAML 메시지 처리를 위한 Callback 핸들러 클래스 설정을 해야 한다.

## SAML 메시지 처리를 위한 Callback 핸들러 클래스 설정

wsit-client.xml 파일에 다음과 SAML 메시지 처리를 위한 Callback 핸들러 클래스를 설정한다.

Callback 핸들러 클래스 설정 : <wsit-client.xml>

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
...
targetNamespace="http://server.fromjava/"
  <ns5:Policy xmlns:ns5="http://schemas.xmlsoap.org/ws/2004/09/policy"
    wsu:Id="TmaxB01TmaxIn1">
    <ns5:ExactlyOne>
      <ns5:All>
        <CallbackHandlerConfiguration
          xmlns="http://schemas.sun.com/2006/03/wss/client">
          <CallbackHandler classname="xwss.saml.SamlCallbackHandler"
            name="samlHandler" />
        </CallbackHandlerConfiguration>
      </ns5:All>
    </ns5:ExactlyOne>
  </ns5:Policy>
  ...
  <binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
    ...
    <operation name="addNumbers">
      <soap:operation soapAction="" />
      <input>
        <ns10:PolicyReference
          xmlns:ns10="http://schemas.xmlsoap.org/ws/2004/09/policy"
          URI="#TmaxB01TmaxIn1" />
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  ...
</definitions>
```

## SAML 메시지 처리를 위한 Callback 핸들러 클래스 작성

SAML 표준에 따라 SOAP 요청 메시지에 정보를 담기 위해 예제에서는 javax.security.auth.callback.CallbackHandler를 구현하는 SamlCallbackHandler를 사용한다. SamlCallbackHandler에서는 Callback의 종류에 따라 필요한 assertion을 저장하는 역할을 한다.

다음은 이 예제에서 사용되는 SamlCallbackHandler이다.

Callback 핸들러 클래스 작성 : <SamlCallbackHandler.java>

```
public class SamlCallbackHandler implements CallbackHandler {
  ...
  public void handle(Callback[] callbacks) throws IOException,
    UnsupportedCallbackException {
    for (int i = 0; i < callbacks.length; i++) {
```



```

        if (callbacks[i] instanceof SAMLCallback) {
            try {
                SAMLCallback samlCallback = (SAMLCallback) callbacks[i];
                if (samlCallback.getConfirmationMethod().equals(
                    samlCallback.SV_ASSERTION_TYPE)) {
                    samlCallback.setAssertionElement(createSVSAMLAssertion());
                    svAssertion = samlCallback.getAssertionElement();
                } else if (samlCallback.getConfirmationMethod().equals(
                    samlCallback.HOK_ASSERTION_TYPE)) {
                    samlCallback.setAssertionElement(createHOKSAMLAssertion());
                    hokAssertion = samlCallback.getAssertionElement();
                } else {
                    throw new Exception("SAML Assertion Type is not matched.");
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        } else {
            throw unsupported;
        }
    }
}

private static Element createSVSAMLAssertion() {...}
private static Element createSVSAMLAssertion20() {...}
private Element createHOKSAMLAssertion() {...}
private Element createHOKSAMLAssertion20() {...}
...
}

```

## 17.4.5. 보안 대화

본 절에서는 서버와 클라이언트에서 보안 대화(Secure Conversation)를 설정하는 방법에 대해서 설명한다.

### 17.4.5.1. 서버 설정

웹 서비스의 서버 설정은 service-config.xml 설정 파일로 구성한다.

서버 보안 대화 설정 : <service-config.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <portcomponent-wsdl-name>AddNumbersPort</portcomponent-wsdl-name>
            <addressing-policy>
                <www-w3-org />
            </addressing-policy>
            <security-policy>
                <security-binding>
                    <symmetric-binding>
                        <protection-token>
                            <secure-conversation-token>
                                <asymmetric-binding-initiator-token>
                                    <include-token>true</include-token>

```

```

        </asymmetric-binding-initiator-token>
        <asymmetric-binding-recipient-token>
        </asymmetric-binding-recipient-token>
        <include-token>true</include-token>
    </secure-conversation-token>
</protection-token>
    <encrypt-signature>true</encrypt-signature>
</symmetric-binding>
</security-binding>
<trust>true</trust>
<keystore>...</keystore>
<truststore>...</truststore>
</security-policy>
<operation-policy-subject>
    <operation-wsdl-name>addNumbers</operation-wsdl-name>
    <input-message-policy-subject>
        <security-policy>
            <protection>
                <signed-part>...</signed-part>
                <encrypted-part>...</encrypted-part>
            </protection>
        </security-policy>
    </input-message-policy-subject>
</operation-policy-subject>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

#### 17.4.5.2. 클라이언트 설정

클라이언트의 웹 서비스 설정은 추가적으로 wsit-client.xml에 Keystore(Truststore) 설정을 해야 한다.

클라이언트 보안 대화 설정 : <wsit-client.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
...
xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    <message name="add" />
    <message name="addResponse" />
    <portType name="NewWebService">
        <wsdl:operation name="add">
            <wsdl:input message="tns:add" />
            <wsdl:output message="tns:addResponse" />
        </wsdl:operation>
    </portType>
    <binding name="NewWebServicePortBinding" type="tns:NewWebService">
        <wsp:PolicyReference URI="#NewWebServicePortBindingPolicy" />
        <soap12:binding style="document"
            transport="http://schemas.xmlsoap.org/soap/http" />
        <wsdl:operation name="add">
            <soap12:operation soapAction="http://xmlsoap.org/Ping"
                style="document" />
            <wsdl:input>
                <soap12:body use="literal" />
            </wsdl:input>
            <wsdl:output>

```

```

        <soap12:body use="literal" />
    </wsdl:output>
</wsdl:operation>
</binding>
<service name="NewWebServiceService">
    <wsdl:port name="NewWebServicePort" binding="tns:NewWebServicePortBinding">
        <soap12:address location="REPLACE_WITH_ACTUAL_ADDRESS" />
    </wsdl:port>
</service>
<wsp:Policy wsu:Id="NewWebServicePortBindingPolicy"
xmlns:scc="http://schemas.sun.com/ws/2006/05/sc/client">
    <wsp:ExactlyOne>
        <wsp:All>
            <sc:KeyStore wspp:visibility="private"
                location="./keystore.jks" type="JKS"
                alias="xws-security-client" storepass="changeit">
            </sc:KeyStore>
            <sc:TrustStore wspp:visibility="private"
                location="./cacerts.jks" type="JKS"
                storepass="changeit" peeralias="xws-security-server"
                stsalias="wssip">
            </sc:TrustStore>
            <scc:SCClientConfiguration wspp:visibility="private">
                <scc:LifeTime>36000</scc:LifeTime>
            </scc:SCClientConfiguration>
        </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
</definitions>

```

## 17.4.6. 웹 서비스 신뢰

본 절에서는 웹 서비스 신뢰(WS-Trust)를 위한 서버와 STS, 클라이언트의 설정 방법에 대해서 설명한다.

### 17.4.6.1. 서버 설정

웹 서비스의 서버 설정은 service-config.xml 설정 파일로 구성한다.

서버 웹 서비스 신뢰 설정 : <service-config.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
<!--
웹 서비스 신뢰는 WS-Addressing 프레임워크 위에서 동작한다
-->
            <addressing-policy>
                <www-w3-org />
            </addressing-policy>
            <security-policy>
<!--
웹 서비스는 x509 토큰을 보안 토큰으로 사용할 것을 명시한다
-->
        </security-binding>
    </policy>
</web-services-config>

```

```

        <symmetric-binding>
            <protection-token>
                <x509-token />
            </protection-token>
            <layout>Strict</layout>
        </symmetric-binding>
    </security-binding>

<!--
웹 서비스는 추가적으로 다음 주소의 STS로부터 보안 토큰을 발급 받을 것을 명시한다
-->

        <token>
            <endorsing-supporting-token>
                <issued-token>
                    <issuer-address>
                        http://localhost:8088/trust_sts/SecurityTokenService
                    </issuer-address>
                </issued-token>
            </endorsing-supporting-token>
        </token>
        <wss-version>11</wss-version>
        <trust>true</trust>
        ...
    </security-policy>
    ...
</endpoint-policy-subject>
</policy>
</web-services-config>

```

#### 17.4.6.2. STS 설정

STS를 위해 WS-SecurityPolicy 설정을 WSDL의 WS-Policy 프레임워크를 통해 설정해야 한다.

다음은 WS-SecurityPolicy 설정에 대한 예이다.

WS-SecurityPolicy 설정 : <sts.wSDL>

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions ...>
    <wsp:Policy wsu:Id="TmaxSTSServerPolicy">
        <wsp:ExactlyOne>
            <wsp:All>
                <sp:SymmetricBinding
                    xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
                    <wsp:Policy>
                        <sp:ProtectionToken>
                            <wsp:Policy>
                                <sp:X509Token
                                    sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/Never">
                                    <wsp:Policy>
                                        <sp:RequireDerivedKeys />
                                        <sp:RequireThumbprintReference />
                                        <sp:WssX509V3Token10 />
                                    </wsp:Policy>
                                </sp:X509Token>
                            </wsp:Policy>
                        </sp:ProtectionToken>
                    </wsp:Policy>
                </sp:SymmetricBinding>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>

```

```

        <wsp:Policy>
            <sp:Basic128 />
        </wsp:Policy>
    </sp:AlgorithmSuite>
    <sp:Layout>
        <wsp:Policy>
            <sp:Lax />
        </wsp:Policy>
    </sp:Layout>
    <sp:IncludeTimestamp />
    <sp:EncryptSignature />
    <sp:OnlySignEntireHeadersAndBody />
</wsp:Policy>
</sp:SymmetricBinding>
<sp:SignedSupportingTokens
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
    <wsp:Policy>
        <sp:UsernameToken
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient"
>
        <wsp:Policy>
            <sp:WssUsernameToken10 />
        </wsp:Policy>
    </sp:UsernameToken>
</wsp:Policy>
</sp:SignedSupportingTokens>
<sp:Wss11
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
    <wsp:Policy>
        <sp:MustSupportRefKeyIdentifier />
        <sp:MustSupportRefIssuerSerial />
        <sp:MustSupportRefThumbprint />
        <sp:MustSupportRefEncryptedKey />
    </wsp:Policy>
</sp:Wss11>
<sp:Trust10
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
    <wsp:Policy>
        <sp:MustSupportIssuedTokens />
        <sp:RequireClientEntropy />
        <sp:RequireServerEntropy />
    </wsp:Policy>
</sp:Trust10>
...
<sc:ValidatorConfiguration
xmlns:sc="http://schemas.sun.com/2006/03/wss/server">
    <sc:Validator name="usernameValidator"
        classname="trust.sts.UsernamePasswordValidator" />
</sc:ValidatorConfiguration>
<tc:STSConfiguration
xmlns:tc="http://schemas.sun.com/ws/2006/05/trust/server">
    encryptIssuedKey="true" encryptIssuedToken="false">
    <tc:LifeTime>36000</tc:LifeTime>
    <tc:Contract>
        com.sun.xml.ws.security.trust.impl.IssueSamlTokenContractImpl
    </tc:Contract>
    <tc:Issuer>TmaxsoftSTS</tc:Issuer>
    <tc:ServiceProviders>

```

```

<tc:ServiceProvider endPoint="http://localhost:8088/trust_server/FinancialService">
    <tc:CertAlias>bob</tc:CertAlias>
    <tc:TokenType>
        http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV1.1
    </tc:TokenType>
</tc:ServiceProvider>
<tc:ServiceProvider endPoint="default">
    <tc:CertAlias>bob</tc:CertAlias>
    <tc:TokenType>
        http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-
1.1#SAMLV1.1
    </tc:TokenType>
</tc:ServiceProvider>
</tc:ServiceProviders>
</tc:STSTConfiguration>
<wsap10:UsingAddressing />
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
...
<wsdl:binding name="ISecurityTokenService_Binding"
    type="tns:ISecurityTokenService">
    <wsp:PolicyReference URI="#TmaxSTSServerPolicy" />
    ...
</wsdl:binding>
...
</wsdl:definitions>

```

### 17.4.6.3. 클라이언트 설정

클라이언트의 웹 서비스 설정은 wsit-client.xml에 서버, STS 각각을 위한 Keystore(Truststore) 설정을 해야 한다.

클라이언트 웹 서비스 신뢰 설정 : <wsit-client.xml>

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://tempuri.org/" ...>
    <!-- FinancialService WSDL -->
    <wsp:Policy wsu:Id="TmaxFSClientPolicy" ...>
        <wsp:ExactlyOne>
            <wsp:All>
                <sc:KeyStore ... />
                <sc:TrustStore ... />
                <scc:SCClientConfiguration wspp:visibility="private">
                    <scc:LifeTime>36000</scc:LifeTime>
                </scc:SCClientConfiguration>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    ...
    <wsdl:binding name="IFinancialService_Binding"
        type="tns:IFinancialService">
        <wsp:PolicyReference URI="#TmaxFSClientPolicy" />
        ...
    </wsdl:binding>
    <wsdl:service name="FinancialService">
        <wsdl:port name="IFinancialService_Port"

```

```

        binding="tns:IFinancialService_Binding">
        <soap:address
            location="http://localhost:8088/trust_server/FinancialService" />
    </wsdl:port>
</wsdl:service>

<!-- STSService WSDL -->
<wsp:Policy wsu:Id="TmaxSTSCClientPolicy" ...>
    <wsp:ExactlyOne>
        <wsp:All>
            <sc:KeyStore ... />
            <sc:TrustStore ... />
            <sc:CallbackHandlerConfiguration
                xmlns:sc="http://schemas.sun.com/2006/03/wss/client">
                <sc:CallbackHandler default="alice" name="usernameHandler" />
                <sc:CallbackHandler default="alice" name="passwordHandler" />
            </sc:CallbackHandlerConfiguration>
        </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
...
<wsdl:binding name="ISecurityTokenService_Binding"
    type="tns:ISecurityTokenService">
    <wsp:PolicyReference URI="#TmaxSTSCClientPolicy" />
    ...
</wsdl:binding>
<wsdl:service name="SecurityTokenService">
    <wsdl:port name="ISecurityTokenService_Port"
        binding="tns:ISecurityTokenService_Binding">
        <soap:address
            location="http://localhost:8088/trust_sts/STSImpService" />
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>

```

## 17.4.7. 실행

다음은 실행 화면이다.

```

<<_Exception_>>
com.sun.xml.ws.server.UnsupportedMediaException: Unsupported Content-Type:
application/soap+xml Supported ones are: [text/xml]
    at com.sun.xml.ws.encoding.StreamSOAPCodec.decode(StreamSOAPCodec.java:295)
    at com.sun.xml.ws.encoding.StreamSOAPCodec.decode(StreamSOAPCodec.java:129)
    at com.sun.xml.ws.encoding.SOAPBindingCodec.decode(SOAPBindingCodec.java:287)
    at jeus.webservices.jaxws.transport.http.HttpAdapter.decodePacket
(HttpAdapter.java:322)
    at jeus.webservices.jaxws.transport.http.HttpAdapter.access$3
(HttpAdapter.java:287)
    at jeus.webservices.jaxws.transport.http.HttpAdapter$HttpToolkit.handle
(HttpAdapter.java:517)
    at jeus.webservices.jaxws.transport.http.HttpAdapter.handle
(HttpAdapter.java:272)
    at jeus.webservices.jaxws.transport.http.EndpointAdapter.handle
(EndpointAdapter.java:149)
    at jeus.webservices.jaxws.transport.http.servlet.WSServletDelegate.doGet

```

```

(WSServletDelegate.java:139)
  at jeus.webservices.jaxws.transport.http.servlet.WSServletDelegate.doPost
(WSServletDelegate.java:170)
  at jeus.webservices.jaxws.transport.http.servlet.WSServlet.doPost
(WSServlet.java:23)
  at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:725)
  at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:818)
  at jeus.servlet.engine.ServletWrapper.executeServlet(ServletWrapper.java:328)
  at jeus.servlet.engine.ServletWrapper.execute(ServletWrapper.java:222)
  at jeus.servlet.engine.HttpServletRequestProcessor.run(HttpServletRequestProcessor.java:278)
<<__!Exception__>>
[2999.01.01 00:00:00][0][bxxx] [TMAX-xx] Unsupported Content-Type:
application/soap+xml Supported ones are: [text/xml]
<<__Exception__>>
com.sun.xml.ws.server.UnsupportedMediaException: Unsupported Content-Type:
application/soap+xml Supported ones are: [text/xml]
  at com.sun.xml.ws.encoding.StreamSOAPCodec.decode(StreamSOAPCodec.java:295)
  at com.sun.xml.ws.encoding.StreamSOAPCodec.decode(StreamSOAPCodec.java:129)
  at com.sun.xml.ws.encoding.SOAPBindingCodec.decode(SOAPBindingCodec.java:287)
  at jeus.webservices.jaxws.transport.http.HttpAdapter.decodePacket
(HttpAdapter.java:322)
  at jeus.webservices.jaxws.transport.http.HttpAdapter.access$3
(HttpAdapter.java:287)
  at jeus.webservices.jaxws.transport.http.HttpAdapter$HttpToolkit.handle
(HttpAdapter.java:517)
  at jeus.webservices.jaxws.transport.http.HttpAdapter.handle(HttpAdapter.java:272)
  at jeus.webservices.jaxws.transport.http.EndpointAdapter.handle
(EndpointAdapter.java:149)
  at jeus.webservices.jaxws.transport.http.servlet.WSServletDelegate.doGet
(WSServletDelegate.java:139)
  at jeus.webservices.jaxws.transport.http.servlet.WSServletDelegate.doPost
(WSServletDelegate.java:170)
  at jeus.webservices.jaxws.transport.http.servlet.WSServlet.doPost
(WSServlet.java:23)
  at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:725)
  at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:818)
  at jeus.servlet.engine.ServletWrapper.executeServlet(ServletWrapper.java:328)
  at jeus.servlet.engine.ServletWrapper.execute(ServletWrapper.java:222)
  at jeus.servlet.engine.HttpServletRequestProcessor.run(HttpServletRequestProcessor.java:278)
<<__!Exception__>>
user alice is a validate user.
your company : Tmaxsoft
your department : Infra

```

클라이언트를 실행하면 클라이언트가 STS의 URL 정보를 얻기 위해 여러 가지 접미 URL을 통해 연결을 시도한다. 여기서 로그 레벨이 'FINE' 이상인 경우 위와 같은 Exception이 발생하는 것을 확인할 수 있다.

## 17.5. JAX-RPC(JEUS 5) 웹 서비스 보안 이전 방법

JEUS 웹 서비스의 메시지 수준 보안은 웹 서비스 보안 정책(Web Service Secure Policy)을 통해 동작한다. 즉, WSDL 문서 내에 직접적으로 웹 서비스 보안 정책을 설정하거나 wsit-endpoint.xml에 웹 서비스 보안 정책을 설정한다.

JEUS에서는 추가적으로 service-config.xml을 통해 웹 서비스 정책 설정을 지원하며 이는 웹 서비스 보안 정책을



지원한다. 따라서 사용자가 웹 서비스 정책 및 웹 서비스 정책에 대해 모르더라도 서버와 클라이언트 간에 이러한 service-config.xml 파일의 공유가 가능하다면 이 파일을 통해 양 끝단의 메시지 수준의 보안을 간단하게 처리할 수 있다. 이는 JAX-RPC(JEUS 5) 웹 서비스의 메시지 수준 보안 방식과 유사하다.

## 17.5.1. 암호화

웹 서비스 메시지 수준 보안에서 암호화(Encryption)에 대한 설정은 다음과 같이 구분된다.

- JAX-RPC 웹 서비스 설정
- JAX-WS 웹 서비스 설정

### JAX-RPC 웹 서비스 설정

JAX-RPC 웹 서비스 메시지 수준 보안에서 암호화를 설정하는 방식은 다음과 같이 jeus-webservices-dd.xml(서버)와 jeus-web-dd.xml(클라이언트)를 설정한다.

JAX-RPC 웹 서비스 암호화 설정 : <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>
      PingSecurityService
    </webservice-description-name>
    <port>
      <port-component-name>PingPort</port-component-name>
      <security>
        <request-receiver>
          <action-list>Encrypt</action-list>
          <password-callback-class>
            ping.PingPWCallback
          </password-callback-class>
          <decryption>
            <keystore>
              <key-type>jks</key-type>
              <keystore-password>changeit</keystore-password>
              <keystore-filename>
                server-keystore.jks
              </keystore-filename>
            </keystore>
          </decryption>
        </request-receiver>
        <response-sender>
          <action-list>Encrypt</action-list>
          <password-callback-class>
            ping.PingPWCallback
          </password-callback-class>
          <encryption-infos>
            <encryption-info>
              <encryptionUser>xws-security-client</encryptionUser>
              <keyIdentifier>DirectReference</keyIdentifier>
              <keystore>
                <key-type>jks</key-type>
                <keystore-password>changeit</keystore-password>
              </keystore>
            </encryption-info>
          </encryption-infos>
        </response-sender>
      </security>
    </port>
  </service>
</jeus-webservices-dd>
```

```

        <keystore-filename>
            server-truststore.jks
        </keystore-filename>
    </keystore>
</encryption-info>
</encryption-infos>
</response-sender>
</security>
</port>
</service>
</jeus-webservices-dd>

```

## JAX-WS 웹 서비스 설정

JAX-WS(JEUS 9) 웹 서비스의 메시지 수준 보안은 웹 서비스를 생성(wsgen 또는 wsimport 통해 웹 서비스를 생성)할 때 service-config.xml 파일을 추가적으로 -policy 인자와 함께 명령문에 설정한다.

JAX-WS(JEUS 9) 웹 서비스 암호화 설정 : <service-config.xml>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <portcomponent-wsdl-name>PingPort</portcomponent-wsdl-name>
            <security-policy>
                <security-binding>
                    <asymmetric-binding>
                        <initiator-token>
                            <x509-token/>
                        </initiator-token>
                        <recipient-token>
                            <x509-token/>
                        </recipient-token>
                    </asymmetric-binding>
                </security-binding>
                <keystore>
                    <keystore-file>
                        <alias>xws-security-client</alias>
                        <key-type>JKS</key-type>
                        <keystore-password>changeit</keystore-password>
                        <keystore-filename>
                            client-keystore.jks
                        </keystore-filename>
                    </keystore-file>
                </keystore>
                <truststore>
                    <keystore-file>
                        <alias>xws-security-server</alias>
                        <key-type>JKS</key-type>
                        <keystore-password>changeit</keystore-password>
                        <keystore-filename>
                            client-truststore.jks
                        </keystore-filename>
                    </keystore-file>
                </truststore>
            </security-policy>
            <operation-policy-subject>

```

```

        <operation-wsdl-name>ping</operation-wsdl-name>
        <input-message-policy-subject>
            <security-policy>
                <protection>
                    <encrypted-part>
                        <body/>
                    </encrypted-part>
                </protection>
            </security-policy>
        </input-message-policy-subject>
        <output-message-policy-subject>
            <security-policy>
                <protection>
                    <encrypted-part>
                        <body/>
                    </encrypted-part>
                </protection>
            </security-policy>
        </output-message-policy-subject>
    </operation-policy-subject>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

## 17.5.2. 서명

웹 서비스 메시지 수준 보안에서 서명(Signature)에 대한 설정은 다음과 같이 구분된다.

- JAX-RPC 웹 서비스 설정
- JAX-WS 웹 서비스 설정

### JAX-RPC 웹 서비스 설정

JAX-RPC 웹 서비스 메시지 수준 보안에서는 다음과 같이 jeus-webservices-dd.xml(서버) 또는 jeus-web-dd.xml(클라이언트)으로 서명을 설정한다.

JAX-RPC 웹 서비스 서명 설정 : <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>
            PingSecurityService
        </webservice-description-name>
        <port>
            <port-component-name>PingPort</port-component-name>
            <security>
                <request-receiver>
                    <action-list>Signature</action-list>
                    <password-callback-class>
                        ping.PingPWCallback
                    </password-callback-class>
                    <signature-verification>
                        <keystore>

```

```

        <key-type>jks</key-type>
        <keystore-password>changeit</keystore-password>
        <keystore-filename>
            server-truststore.jks
        </keystore-filename>
    </keystore>
</signature-verification>
</request-receiver>
<response-sender>
    <action-list>Signature</action-list>
    <password-callback-class>
        ping.PingPWCallback
    </password-callback-class>
    <user>xws-security-server</user>
    <signature-infos>
        <signature-info>
            <keyIdentifier>DirectReference</keyIdentifier>
            <keystore>
                <key-type>jks</key-type>
                <keystore-password>changeit</keystore-password>
                <keystore-filename>
                    server-keystore.jks
                </keystore-filename>
            </keystore>
        </signature-info>
    </signature-infos>
</response-sender>
</security>
</port>
</service>
</jeus-webservices-dd>

```

## JAX-WS 웹 서비스 설정

JAX-WS(JEUS 9) 웹 서비스의 메시지 수준 보안은 웹 서비스를 생성(wsgen 혹은 wsimport 통해 웹 서비스를 생성)할 때 service-config.xml 파일을 추가적으로 -policy 인자와 함께 명령문에 설정한다.

JAX-WS(JEUS 9) 웹 서비스 서명 설정 : <service-config.xml>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <portcomponent-wsdl-name>PingPort</portcomponent-wsdl-name>
            <security-policy>
                <security-binding>
                    <asymmetric-binding>
                        <initiator-token>
                            <x509-token/>
                        </initiator-token>
                        <recipient-token>
                            <x509-token/>
                        </recipient-token>
                    </asymmetric-binding>
                </security-binding>
            </security-policy>
            <keystore>
                <keystore-file>

```

```

        <alias>xws-security-client</alias>
        <key-type>JKS</key-type>
        <keystore-password>changeit</keystore-password>
        <keystore-filename>
            client-keystore.jks
        </keystore-filename>
    </keystore-file>
</keystore>
<truststore>
    <keystore-file>
        <alias>xws-security-server</alias>
        <key-type>JKS</key-type>
        <keystore-password>changeit</keystore-password>
        <keystore-filename>
            client-truststore.jks
        </keystore-filename>
    </keystore-file>
</truststore>
</security-policy>
<operation-policy-subject>
    <operation-wsdl-name>ping</operation-wsdl-name>
    <input-message-policy-subject>
        <security-policy>
            <protection>
                <signed-part>
                    <body/>
                </signed-part>
            </protection>
        </security-policy>
    </input-message-policy-subject>
    <output-message-policy-subject>
        <security-policy>
            <protection>
                <signed-part>
                    <body/>
                </signed-part>
            </protection>
        </security-policy>
    </output-message-policy-subject>
</operation-policy-subject>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

### 17.5.3. Timestamp

웹 서비스 메시지 수준 보안에서 Timestamp의 설정은 다음과 같이 구분된다.

#### JAX-WS 웹 서비스 설정

JAX-WS(JEUS 9) 웹 서비스의 메시지 수준 보안은 웹 서비스를 생성할 때, 즉 `wsgen` 또는 `wsimport`로 웹 서비스를 생성할 때 `service-config.xml` 파일을 추가적으로 `-policy` 인자와 함께 명령문에 설정한다.

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <policy>
    <endpoint-policy-subject>
      <portcomponent-wsdl-name>PingPort</portcomponent-wsdl-name>
      <security-policy>
        <security-binding>
          <asymmetric-binding>
            <initiator-token>
              <x509-token/>
            </initiator-token>
            <recipient-token>
              <x509-token/>
            </recipient-token>
          </asymmetric-binding>
          <timestamp>true</timestamp>
        </security-binding>
        <keystore>
          <keystore-file>
            <alias>xws-security-client</alias>
            <key-type>JKS</key-type>
            <keystore-password>changeit</keystore-password>
            <keystore-filename>
              client-keystore.jks
            </keystore-filename>
          </keystore-file>
        </keystore>
        <truststore>
          <keystore-file>
            <alias>xws-security-server</alias>
            <key-type>JKS</key-type>
            <keystore-password>changeit</keystore-password>
            <keystore-filename>
              client-truststore.jks
            </keystore-filename>
          </keystore-file>
        </truststore>
      </security-policy>
    <operation-policy-subject>
      <operation-wsdl-name>ping</operation-wsdl-name>
      <input-message-policy-subject>
        <security-policy>
          <protection>
            <signed-part>
              <body/>
            </signed-part>
          </protection>
        </security-policy>
      </input-message-policy-subject>
      <output-message-policy-subject>
        <security-policy>
          <protection>
            <signed-part>
              <body/>
            </signed-part>
          </protection>
        </security-policy>
      </output-message-policy-subject>
    </operation-policy-subject>
  </policy>
</web-services-config>

```

```

        </output-message-policy-subject>
    </operation-policy-subject>
</endpoint-policy-subject>
</policy>
</web-services-config>

```

## 17.5.4. 사용자명 토큰

웹 서비스 메시지 수준 보안에서 사용자명 토큰(Username Token)의 설정은 다음과 같이 구분된다.

### JAX-WS 웹 서비스 설정

JAX-WS(JEUS 9) 웹 서비스의 메시지 수준 보안은 웹 서비스를 생성(wsgen 또는 wsimport 통해 웹 서비스를 생성)할 때 service-config.xml 파일을 추가적으로 -policy 인자와 함께 명령문에 설정한다.

JAX-WS(JEUS 9) 웹 서비스 사용자명 토큰 설정 : <service-config.xml>

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <policy>
        <endpoint-policy-subject>
            <security-policy>
                <security-binding>
                    <symmetric-binding>
                        <protection-token>
                            <x509-token>
                                <include-token>true</include-token>
                            </x509-token>
                        </protection-token>
                    </symmetric-binding>
                </security-binding>
            </security-policy>
        </endpoint-policy-subject>
    </policy>
    <token>
        <signed-supporting-token>
            <username-token>
                <username-password-validator>
                    fromjava.server.UsernamePasswordValidator
                </username-password-validator>
                <include-token>true</include-token>
            </username-token>
        </signed-supporting-token>
    </token>
    <keystore>
        <keystore-file>
            <alias>xws-security-server</alias>
            <key-type>JKS</key-type>
            <keystore-password>changeit</keystore-password>
            <keystore-filename>
                keystore.jks
            </keystore-filename>
        </keystore-file>
    </keystore>
    <wss-version>11</wss-version>
    <protection>
        <signed-part>
            <body/>
        </signed-part>
    </protection>
</web-services-config>

```

```
</signed-part>
<encrypted-part>
  <body/>
</encrypted-part>
</protection>
</security-policy>
</endpoint-policy-subject>
</policy>
</web-services-config>
```

## 17.6. 접근 제어 설정된 웹 서비스 호출 방법

JEUS 웹 서비스는 접근이 허용된 사용자만이 웹 서비스를 호출할 수 있도록 접근 제어 설정을 할 수 있으며, 웹 서비스 Back-end에 따라 각각 설정법이 다르다. JEUS 웹 서비스의 접근 제어 설정 방법은 [접근 제어 설정](#)을 참고하도록 한다.

접근 제어(Basic Authentication)가 설정된 웹 서비스의 호출을 하기 위해서는 WSDL에 접근하여 그 내용을 바탕으로 Portable Artifact를 생성하고, 그것을 사용하여 웹 서비스를 호출해야 하는데, 이 경우에 사용자 이름과 암호를 요구할 경우 설정하는 작업이 필요하다.

### 17.6.1. Portable Artifact 생성

접근 제어가 설정된 웹 서비스의 클라이언트를 위한 Portable Artifact의 생성하는 과정이 필요하다.

**wsimport** 툴을 사용하여 Portable Artifact를 생성하기 위해서는 다음과 같이 콘솔 창에 입력한다.

```
$ wsimport -Xauthfile authorization.txt
http://localhost:8088/AddNumbers/AddNumbersImplService?wsdl
```

"authorization.txt" 파일은 실제 WSDL에 접근하기 위한 권한 설정 파일이다. 권한 설정 파일의 예는 다음과 같다.

권한 설정 파일 : <authorization.txt>

```
http://jeus:jeus@localhost:8088/AddNumbers/AddNumbersImplService?wsdl
```

여기에서 사용자명은 'jeus', 암호는 'jeus'이다.

### 17.6.2. 웹 서비스의 클라이언트 작성

JAX-WS 웹 서비스 클라이언트가 스스로 인증하기 위해서는 다음과 같은 2가지의 설정이 클라이언트 프로그램 내에 삽입되어야 한다.

- jakarta.xml.ws.BindingProvider.USERNAME\_PROPERTY
- jakarta.xml.ws.BindingProvider.PASSWORD\_PROPERTY



다음 예는 실제로 프로그램에서 위 설정을 구현한 예이다.

```
Echo port = // ... SEI(Service Endpoint Interface)의 획득
((jakarta.xml.ws.BindingProvider) port).getRequestContext().
    put(jakarta.xml.ws.BindingProvider.USERNAME_PROPERTY, "jeus");
((jakarta.xml.ws.BindingProvider) port).getRequestContext().
    put(jakarta.xml.ws.BindingProvider.PASSWORD_PROPERTY, "jeus");
String s = port.echoString("JEUS");
```

런타임에 JAX-WS 웹 서비스 클라이언트 엔진은 동적 Stub을 생성하기 위해 원격 WSDL에 한 번 접근하는데, 이때 부가적인 권한 설정이 필요하다. 이러한 권한 설정은 `java.net.Authenticator` 클래스를 사용하여 클라이언트 프로그램에 설치한다.

다음은 `java.net.Authenticator` 클래스를 사용한 클라이언트 프로그램의 예이다.

웹 서비스 클라이언트 : <AddNumberClient.java>

```
public class AddNumbersClient {
    ...
    public void execute() {
        Authenticator.setDefault(new MyAuthenticator());

        AddNumbersImpl port = new AddNumbersImplService().getAddNumbersImplPort();
        ((jakarta.xml.ws.BindingProvider) port).getRequestContext().
            put(jakarta.xml.ws.BindingProvider.USERNAME_PROPERTY, "jeus");
        ((jakarta.xml.ws.BindingProvider) port).getRequestContext().
            put(jakarta.xml.ws.BindingProvider.PASSWORD_PROPERTY, "jeus");

        int number1 = 10;
        int number2 = 20;

        port.addNumbers(number1, number2);
        ...
    }

    class MyAuthenticator extends Authenticator {
        protected PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication("jeus", "jeus".toCharArray());
        }
    }
}
```

## 17.7. 접근 제어 설정

JEUS 웹 서비스는 접근이 허용된 사용자만 호출할 수 있도록 접근 제어 설정이 가능하며, 설정 방법은 Back-end에 따라 다르다. 본 절에서는 웹 서비스 접근 제어 설정 방법과 접근 제어가 설정된 웹 서비스를 호출하는 방법에 대해 설명한다.

## 17.7.1. Java 클래스 웹 서비스 접근 제어 설정

특정 URL에 대한 접근을 제한하여 웹 서비스 접근을 제어할 수 있으며, 이를 위해 web.xml 및 jeus-web-dd.xml 같은 웹 응용 프로그램의 DD에 보안 정보를 추가해야 한다. 또한 보안 도메인 설정도 필요하다.

### 보안 도메인 설정

접근 제어 설정을 하기 위해서는 먼저 사용자 등록이 필요하다.

다음 경로의 accounts.xml에 사용자를 등록한다.

```
JEUS_HOME/config/{NODE_NAME}/security/{SECURITY_DOMAIN_NAME}/accounts.xml
```

다음은 사용자 등록의 예로 사용자의 이름은 'jeus'이고, 암호는 'jeus'이며, 암호는 base64 인코딩하여 등록한다.

접근 제어 설정 : <accounts.xml>

```
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <users>
    <user>
      <name>jeus</name>
      <password>{SHA}McbQ1yhI3yi0G1HGTg8DQVWkyhg=</password>
    </user>
  </users>
</accounts>
```

### DD 파일 작성

jeus-web-dd.xml에 다음과 같이 <role-mapping>을 추가한다.

접근 제어 설정 : <jeus-web-dd.xml>

```
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <docbase>BA_DocLitEchoService</docbase>
  <role-mapping>
    <role-permission>
      <principal>jeus</principal>
      <role>Administrator</role>
    </role-permission>
  </role-mapping>
</jeus-web-dd>
```

그리고 web.xml에 다음과 같이 보안 관련 설정을 추가한다.

접근 제어 설정 : <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 설정하면 URL이 /BA\_DocLitEchoService인 모든 요청에 대해 사용자가 'jeus'이고, 암호가 'jeus'일 경우에만 접근이 허용된다.

## 17.7.2. EJB 웹 서비스의 접근 제어 설정

EJB 웹 서비스 접근 제어는 Java 클래스 웹 서비스와 마찬가지로 특정 URL에 대한 접근을 제한하여 제어할 수 있다. 이를 위해 ejb-jar.xml, jeus-ejb-dd.xml 같은 EJB DD와 jeus-webservices-dd.xml 같은 웹 서비스 DD에 보안 정보를 추가해야 한다. 또한 보안 도메인 설정도 필요하다.

### 보안 도메인 설정

접근 제어 설정을 하기 위해서는 먼저 사용자의 등록이 필요하다.

다음 경로의 accounts.xml에 사용자를 등록한다.

```
JEUS_HOME/config/{NODE_NAME}/security/{SECURITY_DOMAIN_NAME}/accounts.xml
```

다음은 사용자 등록의 예로 사용자의 이름은 'jeus'이고, 암호는 'jeus'이며, 암호는 base64 인코딩하여 등록한다.

EJB 웹 서비스의 접근 제어 설정 : <accounts.xml>

```

<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <users>
        <user>
            <name>jeus</name>
            <password>{SHA}McbQ1yhI3yi0G1HGTg8DQVWkyhg=</password>
        </user>
    </users>
</accounts>

```

## DD 파일 작성

jeus-ejb-dd.xml에 다음과 같이 <role-permission>을 추가한다.

EJB 웹 서비스의 접근 제어 설정 : <jeus-ejb-dd.xml>

```
<jeus-ejb-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <module-info>
    <role-permission>
      <principal>jeus</principal>
      <role>Administrator</role>
    </role-permission>
  </module-info>
  <beanlist>
    ...
  </beanlist>
</jeus-ejb-dd>
```

그리고 ejb-jar.xml에 다음과 같이 보안 관련 설정을 추가한다.

EJB 웹 서비스의 접근 제어 설정 : <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>
```

추가적으로 웹 서비스 DD에 다음과 같은 설정을 할 수 있으며, 별도로 설정하지 않으면 기본값이 사용된다. SSL을 통해 데이터 무결성을 보장하려면 <ejb-transport-guarantee>를 'INTEGRAL'로 설정하고, 기밀성과 무결성을 모두 보장하려면 'CONFIDENTIAL'로 설정해야 한다. 이 경우 HTTPS 통신을 위해 별도의 HTTP 리스너 설정이 필요하다.

EJB 웹 서비스의 접근 제어 설정 : <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>
</jeus-webservices-dd>
```

```

</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>

```

### 17.7.3. 접근 제어가 설정된 웹 서비스 호출

접근 제어(Basic Authentication)가 설정된 웹 서비스를 호출하려면 WSDL에 접근해 그 내용을 바탕으로 Stub을 생성하고, 이를 통해 웹 서비스를 호출해야 한다. 이때 사용자 이름과 암호를 요구할 경우 해당 설정을 추가해야 한다.

#### WSDL로부터 Stub 생성

콘솔 툴을 이용할 경우 다음과 같이 설정한다.

```

wsdl2java -gen:client -username jeus -password jeus
http://localhost:8088/BA_DocLitEchoService/BA_DocLitEchoService?wsdl

```

# 18. Server-Sent Event

본 장에서는 JEUS에서 HTML5의 Server-Sent Event(SSE) 사용 방법에 대해서 설명한다.

## 18.1. 개요

Server-Sent Events는 서버가 단방향 클라이언트-서버 연결을 통해 표준 HTTP 또는 HTTPS를 사용하여 웹페이지에 데이터를 푸시할 수 있도록 해 준다. Server-Sent Events 통신 모델에서 클라이언트(예: 브라우저)는 초기 연결을 유지하고 서버는 데이터를 제공하여 클라이언트에 전송한다.

Server-Sent Events는 HTML5 스펙의 일부로 제안된 표준 기술이며 JEUS에서의 Server-Sent Event 지원은 JAX-RS(Java API for RESTful Web Services) 2.0을 통해서 제공된다.

## 18.2. JAX-RS 리소스에서 Server-Sent Events(SSE) 지원

SSE 지원을 위하여 리소스(Resource)에 SseFeature를 추가한다.

간단한 SSE 리소스 메소드

```
...
import jeus.webservices.jaxrs.media.sse.EventOutput;
import jeus.webservices.jaxrs.media.sse.OutboundEvent;
import jeus.webservices.jaxrs.media.sse.SseFeature;
...

@Path("events")
public class SseResource {

    @GET
    @Produces(SseFeature.SERVER_SENT_EVENTS)
    public EventOutput getServerSentEvents() {
        EventOutput eventOutput = new EventOutput();
        new Thread(new Runnable() {
            @Override
            public void run() {
                try {
                    for (int i = 0; i < 5; i++) {
                        // ... code that waits a few seconds
                        OutboundEvent.Builder eventBuilder = new OutboundEvent.Builder();
                        eventBuilder.name("example");
                        eventBuilder.data(String.class, "Hello world " + i + "!");
                        OutboundEvent event = eventBuilder.build();
                        eventOutput.write(event);
                    }
                } catch (IOException e) {
                    throw new RuntimeException(e);
                } finally {
                    try {
                        eventOutput.close();
                    } catch (IOException e) {
                        throw new RuntimeException(e);
                    }
                }
            }
        }).start();
    }
}
```

```

        }
    }
    }).start();
    return eventOutput;
}
}

```

위 코드는 URI "/events"로 배치(deployment)된 리소스를 정의한다. 리소스는 Output chunked 메시지 처리를 위한 EventOutput 엔티티를 반환하는 한개의 @GET 리소스 메소드를 가지고 있다.

EventOutput가 메소드로부터 반환된 후, JAX-RS 런타임은 EventOutput를 ChunkedOutput으로 인지하여 클라이언트 연결을 즉시 끊지 않고 응답 스트림에 HTTP 헤더를 쓰고 전송한 chunks (SSE events)를 기다린다. 이 시점에 클라이언트는 헤더를 읽고 개별 이벤트에 대한 리스닝을 시작한다.

예제에서 리소스 메소드는 일련의 5개 이벤트 전송을 위하여 스레드를 생성한다. 연속된 이벤트 사이에는 약간의 지연이 있다. 각각의 이벤트는 OutboundEvent 타입으로 표현된다. OutboundEvent는 SSE 메시지의 표준 형태를 반영하며 name, comment, id 를 표현하는 프로퍼티를 포함한다. data(Class) 메소드는 이벤트 데이터를 직렬화하기 위하여 사용된다.

SSE 지원 리소스에 연결한 클라이언트는 엔티티 스트림으로부터 다음 데이터를 받을 것이다.

```

event: example
data: Hello world 0!

event: example
data: Hello world 1!

event: example
data: Hello world 2!

event: example
data: Hello world 3!

event: example
data: Hello world 4!

```

## 18.3. JAX-RS 클라이언트에서 SSE 이벤트 처리

JEUS JAX-RS는 두 가지 프로그래밍 모델을 사용하여 SSE 이벤트 수신과 처리를 지원하기 위한 클라이언트 API를 제공한다.

- Pull 모델 : EventInput으로부터 이벤트를 폴링하는 모델
- Push 모델 : EventSource의 비동기 통지를 리스닝하는 모델

### 18.3.1. EventInput를 사용하여 SSE 이벤트 읽기

클라이언트에서 이벤트는 EventInput으로부터 읽을 수 있다. 다음 코드를 살펴 보자.

```

Client client = ClientBuilder.newBuilder().build();
WebTarget target = client.target("http://localhost:8088/sse_example/events");
EventInput eventInput = target.request().get(EventInput.class);
while (!eventInput.isClosed()) {
    final InboundEvent inboundEvent = eventInput.read();
    if (inboundEvent == null) {
        // connection has been closed
        break;
    }
    System.out.println(inboundEvent.getName() + ": " + inboundEvent.readData(String.class));
}

```

클라이언트는 [간단한 SSE 리소스 메소드](#)의 SseResource가 배치된 서버에 연결을 한다. 먼저 JAX-RS client 객체를 생성한다. 그 다음에 WebTarget 객체를 client 객체로부터 얻어 HTTP 요청 호출을 위해 사용한다. 반환된 응답 엔티티는 EventInput으로 직접 읽을 수 있다. 예제 코드는 eventInput 응답 스트림으로부터 inbound SSE 이벤트를 읽기 처리를 위하여 루프(loop)를 시작한다. eventInput으로부터 읽은 개별 chunk는 InboundEvent 이다. InboundEvent.readData(Class) 메소드는 이벤트 데이터의 역직렬화를 위하여 사용하는 메소드이다.

클라이언트 코드는 다음 결과를 보인다.

```

example: Hello world 0!
example: Hello world 1!
example: Hello world 2!
example: Hello world 3!
example: Hello world 4!

```

### 18.3.2. EventSource를 사용하여 비동기 SSE 처리

비동기로 SSE 이벤트를 읽기 위해 사용되는 클라이언트 API는 EventSource이다. EventSource의 사용법은 다음 예제에서 살펴 보자.

```

Client client = ClientBuilder.newBuilder().build();
WebTarget target = client.target("http://localhost:8088/sse_example/events");
EventSource eventSource = new EventSource(target);
EventListener listener = new EventListener() {
    @Override
    public void onEvent(InboundEvent inboundEvent) {
        System.out.println(inboundEvent.getName() + ": " + inboundEvent.readData(String.class));
    }
};
eventSource.addEventListener("example", listener);
eventSource.open();
...
eventSource.close();

```

클라이언트는 [간단한 SSE 리소스 메소드](#)의 SseResource가 배치된 서버에 연결을 한다. 먼저 JAX-RS client 객체를 생성한다. 그 다음에 WebTarget 객체를 client 객체로부터 얻는다. WebTarget에 대한 HTTP 요청 호출은



WebTarget 객체에서 직접 만들지 않는다. EventSource 객체를 target 객체로 초기하여 생성하자. 생성된 EventSource 객체는 자동적으로 target 연결을 하지 않는다. 연결은 나중에 eventSource.open() 메소드를 사용하여 수동적으로 수행한다.

EventListener 구현체는 incoming SSE 이벤트를 Listen하고 처리하기 위하여 사용된다.

InboundEvent.readData(Class) 메소드가 inboundEvent 객체로부터 이벤트 데이터를 역직렬화하기 위하여 사용된다.

예제로부터 리스너는 다음 결과를 보인다.

```
example: Hello world 0!  
example: Hello world 1!  
example: Hello world 2!  
example: Hello world 3!  
example: Hello world 4!
```

# 19. JEUS 웹 서비스 XML

본 장에서는 JEUS 9 웹 서비스가 지원하는 XML에 관련된 다양한 기술들에 대해 설명한다.

## 19.1. 개요

XML 문서를 스키마로부터 컴파일된 Java 클래스로 Java 오브젝트화(바인딩)하여 XML 문서의 정보들을 프로그래밍하듯이 다룰 수 있도록(programmatic) 해주는 JAXB(Java Architecture for XML Binding)에 대해 알아본다. 이후, JAXP(Java API for XML Processing)에 새롭게 API를 도입하기 시작한 StAX(Streaming APIs For XML)에 대해서도 알아보기로 한다.

- JAXB(Java Architecture for XML Binding)
- JAXP(Java API for XML Processing)
- SJSP(Sun Java Streaming XML Parser)

본 장을 진행하기에 앞서 XML 문서 혹은 문서 내의 내용물과 Java 오브젝트의 변환에서 일반적으로 사용되는 용어에 대해 설명한다.

구분	의미
언마셜링(Unmarshalling)	XML 문서 혹은 XML 콘텐츠를 Java 클래스로 오브젝트화(Java Content Tree)하는 과정이다.
마셜링(Marshalling)	Java 오브젝트(Java Content Tree)를 XML 문서 혹은 XML 콘텐츠로 변환하는 과정이다.

## 19.2. JAXB(XML 바인딩을 위한 Java 아키텍처)

어떤 스키마를 준수하는 XML 문서에 관련된 애플리케이션은 그 문서를 생성하거나 수정하거나 혹은 읽어들이 수 있어야 한다. 이와 같은 애플리케이션에서 사용되는 XML에 관련된 데이터 바인딩은 SAX나 DOM API를 이용할 수도 있지만 유지보수(스키마 변화에 따른)의 면에서 쉽지 않다. 그에 따라 JAXB는 XML 문서와 Java 오브젝트 사이의 연결(mapping)을 자동화시켜주는 API 및 툴들을 제공한다.

JAXB의 기능들은 다음과 같다.

- XML 콘텐츠들을 Java 오브젝트로 변환하는 과정인 언마셜링 기능
- Java 오브젝트로 표현된 객체들에 대하여 접근하거나 수정
- Java로 표현된 객체들을 다시 XML 콘텐츠로 변환하는 과정인 마셜링 기능

이와 같이 JAXB는 XML과 Java 코드들 사이의 표준화된, 그리고 효과적인 연결(mapping)을 개발자에게 제공한다. 결국 JAXB는 XML과 웹 서비스 기술을 사용하는 애플리케이션을 더욱 쉽게 개발할 수 있도록 한다. 계속되는 장에서는 JAXB에서 제공해주는 툴과 함께 예제 프로그램을 통해 JAXB에 관해 보다 자세히 설명한다.

## 19.2.1. 바인딩 컴파일러(XJC) 관련 프로그래밍 기법

XML 콘텐츠들을 Java 오브젝트로 변환하기 위해서는 변환하기 전에 미리 Java 클래스들을 가지고 있어야 한다. 이러한 Java 클래스는 스키마로부터 생성되며 이러한 과정을 바인딩 컴파일이라 한다.

본 절에서는 JEUS 9 웹 서비스가 기본으로 제공하는 바인딩 컴파일러 툴인 XJC에 대해 설명한다.

### XJC

기본적인 동작방법은 커맨드 라인(Command Line)에서 다음과 같이 입력하여 명령어를 수행한다.

```
JEUS_HOME/bin$ xjc.cmd -help
```

사용 방법은 다음과 같다.

```
Usage: xjc [-options ...] <schema file/URL/dir/jar> ...
        [-b <bindinfo>] ...
If dir is specified, all schema files in it will be compiled.
If jar is specified, /META-INF/sun-jaxb.episode binding file will be
compiled.
Options:
  -nv                : do not perform strict validation of the input
                      schema(s)
  -extension         : allow vendor extensions - do not strictly
                      follow the Compatibility Rules and App E.2
                      from the JAXB Spec
  -b <file/dir>      : specify external bindings files (each <file>
                      must have its own -b)
                      If a directory is given, **/*.xjb is searched
  -d <dir>           : generated files will go into this directory
  -p <pkg>           : specifies the target package
  -httpproxy <proxy> : set HTTP/HTTPS proxy. Format is
                      [user[:password]@]proxyHost:proxyPort
  -httpproxyfile <f> : Works like -httpproxy but takes the argument
                      in a file to protect password
  -classpath <arg>   : specify where to find user class files
  -catalog <file>    : specify catalog files to resolve external
                      entity references support TR9401, XCatalog,
                      and OASIS XML Catalog format.
  -readOnly          : generated files will be in read-only mode
  -npa               : suppress generation of package level
                      annotations(**/package-info.java)
  -no-header          : suppress generation of a file header with
                      timestamp
  -target 2.0        : behave like XJC 2.0 and generate code that
                      doesnt use any 2.1 features.
  -xmlschema         : treat input as W3C XML Schema (default)
  -relaxng            : treat input as RELAX NG (experimental,
                      unsupported)
  -relaxng-compact   : treat input as RELAX NG compact syntax
                      (experimental, unsupported)
  -dtd                : treat input as XML DTD (experimental,
                      unsupported)
  -wsdl              : treat input as WSDL and compile schemas inside
                      it(experimental,unsupported)
```

```
-verbose      : be extra verbose
-quiet       : suppress compiler output
-help        : display this help message
-version     : display version information
```

#### Extensions:

```
-Xlocator      : enable source location support for generated
                 code
-Xsync-methods : generate accessor methods with the
                 'synchronized' keyword
-mark-generated : mark the generated code as @jakarta.annotation.
                 Generated
-episode <FILE> : generate the episode file for separate
                 compilation
```



JEUS 9 웹 서비스는 XJC의 Ant Task도 지원하는데 보다 자세한 설명은 JEUS Reference 안내서의 "xjc" 콘솔 툴 및 "xjc" Ant Task를 참고한다.

## XJC Ant Task를 이용한 프로그래밍

다음은 하나의 스키마와 XML 문서를 JAXB를 이용하여 메모리상의 Java 객체로 바꿔서 XML 문서의 콘텐츠들을 프로그래밍적(programmatic)으로 다루는 예이다.

전체적인 흐름은 다음과 같다.

1. 하나의 XML 문서를 언마셜링하여 Java 객체로 변환한다.
2. 변환된 Java 객체에 대해 프로그래밍적으로 가공을 한다.
3. Java 객체를 다시 XML 문서로 마셜링(이 예제에서는 출력)한다.

다음은 이 예제의 build.xml의 일부분이다.

XJC Ant Task를 이용한 프로그래밍 : <build.xml>

```
...
<project basedir="." default="run">
...
  <taskdef name="xjc" classname="com.sun.tools.xjc.XJCTask">
    <classpath refid="classpath" />
  </taskdef>
  <target name="compile" description="Compile all Java source
    files">
    <echo message="Compiling the schema..." />
    <mkdir dir="gen-src" />
    <xjc extension="true" schema="po.xsd" package="primer.myPo"
      destdir="gen-src">
      <produces dir="gen-src/primer.myPo" includes="**/*.java" />
    </xjc>
    <echo message="Compiling the java source files..." />
    <mkdir dir="classes" />
    <javac destdir="classes" debug="on">
      <src path="src" />
    </javac>
  </target>
</project>
```

```

        <src path="gen-src" />
        <classpath refid="classpath" />
    </javac>
</target>
<target name="run" depends="compile" description="Run the sample
app">
    <echo message="Running the sample application..." />
    <java classname="Main" fork="true">
        <classpath refid="classpath" />
    </java>
</target>
...
</project>

```

다음은 이 예제의 Main.java에 대한 설명이다.

XJC Ant Task를 이용한 프로그래밍 (1) : <Main.java>

```

Schema schema = SchemaFactory.newInstance(W3C_XML_SCHEMA_NS_URI).
    newSchema(new File("src/conf/ts.xsd"));

JAXBContext jc = JAXBContext.newInstance("com.tmaxsoft");

//
Unmarshaller unmarshaller = jc.createUnmarshaller();
unmarshaller.setSchema(schema);

...

//
Marshaller marshaller = jc.createMarshaller();
marshaller.setSchema(schema);
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

...

```

이전에 build.xml의 XJC Ant Task를 사용하여 생성한 Java 클래스들의 패키지 이름을 이용하여 JAXB 컨텍스트 객체를 생성하고 이를 이용하여 Unmarshaller와 Marshaller를 생성한다. 그리고 스키마에 맞게 구현된 XML 문서인지를 판단하기 위한 Schema 객체를 생성하여 Unmarshaller와 Marshaller에 등록한다.

XJC Ant Task를 이용한 프로그래밍 (2) : <Main.java>

```

Object ts = unmarshaller.unmarshal(new File("src/conf/tsInput.xml"));
TmaxSoftType tst = (TmaxSoftType) ((JAXBElement) ts).getValue();
Address address = tst.getAddress1();
address.setName("John Bob");
address.setStreet("242 Main Street");
address.setCity("Beverly Hills");

```

poInput.xml이라는 XML 문서를 언마셜링한다. 그리고 언마셜링된 Java 객체를 수정하면 화면에 출력(마셜링)한다.

XJC Ant Task를 이용한 프로그래밍 (3) : <Main.java>

```

marshaller.marshal(ts, System.out);

```

## 19.2.2. 스키마 생성기(Schemagen) 관련 프로그래밍 기법

JEUS 9 웹 서비스는 스키마로부터 Java 클래스를 생성하는 스키마 컴파일러인 XJC와 함께 사용자가 미리 작성한 Java 클래스들로부터 특정 XML 스키마를 작성할 수 있는 툴을 제공한다. 이러한 툴을 스키마 생성기(generator)라 하는데 본 절에서는 JEUS 9 웹 서비스가 기본으로 제공하는 스키마 생성기인 Schemagen에 대해 설명한다.

### Schemagen

기본적인 동작 방법은 커맨드 라인(Command Line)에서 다음과 같이 입력하여 명령을 수행한다.

```
JEUS_HOME/bin$ schemagen.cmd -help
```

사용 방법은 다음과 같다.

```
Usage: schemagen [-options ...] <java files>
Options:
  -d <path>          : specify where to place processor and javac
                        generated class files
  -cp <path>          : specify where to find user specified files
  -classpath <path>   : specify where to find user specified files
  -episode <file>     : generate episode file for separate
                        compilation
  -version            : display version information
  -help               : display this usage message
```



JEUS 9 웹 서비스는 Schemagen의 Ant Task도 지원하는데 보다 자세한 설명은 JEUS Reference 안내서의 "schemagen" 콘솔 툴 및 "schemagen" Ant Task를 참고한다.

### Schemagen Ant Task를 이용한 프로그래밍

전체적인 흐름은 다음과 같다.

1. 소스 레벨에서 Schemagen 툴을 사용하여 Java 소스들로 스키마를 생성한다.
2. Java 소스들을 컴파일한다.
3. 컴파일된 Java 소스에 대해 프로그래밍적으로 데이터를 입력하여 Java 오브젝트를 생성한다.
4. Java 오브젝트를 앞에서 얻은 스키마를 이용하여 마셜링(이 예제에서는 출력)한다.

다음은 이 예제 build.xml의 한 부분이다.

Schemagen Ant Task를 이용한 프로그래밍 : <build.xml>

```
...
<project basedir="." default="run">
...
  <taskdef name="schemagen"
    classname="com.sun.tools.jxc.SchemaGenTask">
    <classpath refid="classpath" />
  </taskdef>
```

```

</taskdef>
<target name="compile"
  description="Compile all Java source files">
  <echo message="Generating schemas..." />
  <mkdir dir="schemas" />
  <schemagen destdir="schemas">
    <src path="src" />
    <classpath refid="classpath" />
  </schemagen>
  <echo message="Compiling the java source files..." />
  <mkdir dir="classes" />
  <javac destdir="classes" debug="on">
    <src path="src" />
    <classpath refid="classpath" />
  </javac>
</target>
<target name="run" depends="compile"
  description="Run the sample app">
  <echo message="Running the sample application..." />
  <java classname="Main" fork="true">
    <classpath refid="classpath" />
  </java>
</target>
...
</project>

```

다음은 이 예제에서 하나의 스키마를 나타내는 Java 클래스들이다.

- BusinessCard.java
- Address.java
- jaxb.index
- package-info.java
- Main.java

이 중 Main.java에 대해서만 간략히 알아본다.

Schemagen Ant Task를 이용한 프로그래밍 (1) : <Main.java>

```

BusinessCard card =
    new BusinessCard("John Doe", "Sr. Widget Designer", "Acme, Inc.",
        new Address(null, "123 Widget Way", "Anytown", "MA", (short) 12345),
        "123.456.7890", null, "123.456.7891", "John.Doe@Acme.ORG");

JAXBContext context = JAXBContext.newInstance(BusinessCard.class);
Marshaller m = context.createMarshaller();
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
m.marshal(card, new FileOutputStream(new File("src/conf/bcard.xml")));

...

```

컴파일된 Java 소스를 이용하여 Java 오브젝트를 하나 생성하고 Marshaller를 하나 등록하여 bcard.xml이라는 파일에 마셜링한다.

```
Unmarshaller um = context.createUnmarshaller();
Schema schema = SchemaFactory.newInstance(W3C_XML_SCHEMA_NS_URI)
    .newSchema(Main.class.getResource("schema1.xsd"));
um.setSchema(schema);
Object bce = um.unmarshal(new File("src/conf/bcard.xml"));
m.marshal(bce, System.out);
```

생성된 스키마 파일을 이용하여 Unmarshaller를 하나 생성하고 앞에서 마셜링한 bcard.xml 파일을 Unmarshaller한다. 이를 다시 화면에 출력(Marshaller)한다.

## 19.3. JAXP(XML을 다루기 위한 Java 표준 API)

XML과 Java는 그 성격상 플랫폼에 독립적이라는 면에서 여러 방면에서 함께 사용되었다. JEUS 9 웹 서비스는 XML 문서를 다루기 위한 Java 표준의 API를 지원한다. 이 표준은 Java의 표준 단체인 JCP가 명시한 XML 문서를 다루는 방식에 대한 표준(<http://jcp.org/en/jsr/detail?id=206>)을 따르고 있다.

JAXP의 주요 API는 다음과 같다.

- SAX
- DOM
- TrAX
- DOM
- StAX

### 19.3.1. StAX(Java 스트리밍 XML 파서)

JEUS 9 웹 서비스는 XML을 파싱할 때 사용하는 Java 스트리밍 방식 또한 지원한다. 이 표준은 Java의 표준 단체인 JCP가 명시한 Java 스트리밍 XML 파서를 위한 API에 관한 표준(<http://www.jcp.org/en/jsr/detail?id=173>)을 따르고 있다.