Copyright Notice

Copyright © 2013 TmaxSoft Co., Ltd. All Rights Reserved.

대한민국 경기도 성남시 분당구 서현동 272-6 우) 463-824

Restricted Rights Legend

All TmaxSoft Software (JEUS®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and may only be used or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd.

이 소프트웨어(JEUS®) 사용설명서의 내용과 프로그램은 저작권법과 국제 조약에 의해서 보호받고 있습니다. 사용설명서의 내용과 여기에 설명된 프로그램은 TmaxSoft Co., Ltd.의 사용권 계약 하에서만 사용이 가능하며, 사용권 계약을 준수하는 경우에만 사용 또는 복제할 수 있습니다. 이 사용설명서의 전부 또는 일부분을 TmaxSoft의 사전 서면 동의 없이 전자, 기계, 녹음 등의 수단을 사용하여 전송, 복제, 배포, 2차적 저작물작성 등의 행위를 하여서는 안 됩니다.

Trademarks

JEUS® is registered trademark of TmaxSoftCo., Ltd. Other products, titles or services may be registered trademarks of their respective companies.

JEUS®는 TmaxSoft Co., Ltd.의 등록 상표입니다. 기타 모든 제품들과 회사 이름은 각각 해당 소유주의 상표로 참조용으로만 사용됩니다.

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : APACHE2.0, CDDL1.0, EDL1.0, OPEN SYMPHONY SOFTWARE1.1, TRILEAD-SSH2, Bouncy Castle, BSD, MIT, SIL OPEN FONT1.1

Detailed Information related to the license can be found in the following directory : ${INSTALL_PATH}/lib/licenses

관련 상세한 정보는 제품의 다음의 디렉터리에 기재된 사항을 참고해 주십시오. : ${INSTALL_PATH}/lib/licenses

안내서 정보

안내서 제목: JEUS Web Service 안내서
발행일: 2013-04-30
소프트웨어 버전: JEUS v7.0 Fix#1
안내서 버전: v2.1.2
내용 목차

제1장  웹 서비스 ........................................................................................................ 1
  1.2.  웹 서비스 표준 ............................................................................................... 2
      1.2.1.  SOAP 표준 ......................................................................................... 3
      1.2.2.  WSDL 표준 .......................................................................................... 4
      1.2.3.  UDDI 표준 ............................................................................................ 4
      1.2.4.  JAX-WS, JAXB, SIAX 표준 ................................................................. 4
      1.2.5.  JAX-RPC 표준 ....................................................................................... 4
      1.2.6.  SAAJ 표준 ............................................................................................. 5
  1.3.  SOAP 메시지 교환과 SOAP 메시지 인코딩 ................................................. 5

제2장  Jeus 웹 서비스 ............................................................................................ 7
  2.1.  기본 구조 ........................................................................................................ 7
  2.2.  웹 서비스의 설계 ............................................................................................ 8
      2.2.1.  웹 서비스 Back-end의 선택 ................................................................ 8
      2.2.2.  RPC 방식과 문서 방식의 선택 ............................................................. 9
      2.2.3.  웹 서비스 구현 방식 선택 .................................................................... 9
      2.2.4.  SOAP 메시지 핸들러의 생성 ............................................................... 10

I.  Jeus 7 웹 서비스 ................................................................................................. 13

제3장  Jeus 웹 서비스 구현 ................................................................................... 15
  3.1.  개요 ............................................................................................................... 15
  3.2.  Java 클래스 웹 서비스의 구현 .................................................................... 15
  3.3.  EJB 웹 서비스의 구현 .................................................................................. 20
  3.4.  WSDL로부터 웹 서비스 구현 ................................................................. 25

제4장  웹 서비스 생성과 배치 ................................................................................. 33
  4.1.  개요 ............................................................................................................... 33
  4.2.  Java 클래스 웹 서비스 생성과 배치 ............................................................. 33
  4.3.  EJB 웹 서비스 생성과 배치 ........................................................................ 34
  4.4.  WSDL 웹 서비스 생성과 배치 ..................................................................... 34
  4.5.  Endpoint 주소 결정 방식 ............................................................................ 35
      4.5.1.  서블릿 Endpoint .................................................................................. 35
      4.5.2.  EJB Endpoint ...................................................................................... 37

제5장  웹 서비스 호출 ............................................................................................. 41
  5.1.  개요 ............................................................................................................... 41
  5.2.  동적 프록시 방식의 웹 서비스 호출 ............................................................ 41
      5.2.1.  Client Artifact 생성 ........................................................................... 41
      5.2.2.  Java SE 클라이언트 방식 ................................................................. 48
      5.2.3.  Java EE 클라이언트 방식 ................................................................. 49
5.3. 디스패치 방식의 웹 서비스 호출 ................................................................. 51

제6장 표준 바인딩 선언 및 사용자화 ................................................................. 53
6.1. 개요 ........................................................................................................... 53
6.2. 표준 바인딩 선언 ..................................................................................... 53
6.2.1. 외부 문서(파일)에서 직접 선언 ......................................................... 54
6.2.2. WSDL 문서 내에서 직접 선언 ........................................................... 54
6.3. 표준 바인딩의 사용자화 .......................................................................... 55
6.3.1. 전체적인 바인딩 ................................................................................. 55
6.3.2. 패키지명의 사용자화 ......................................................................... 56
6.3.3. Wrapped 스타일 .................................................................................. 57
6.3.4. 비동기화 ............................................................................................. 57
6.3.5. 프로바이더 인터페이스 ..................................................................... 58
6.3.6. 클래스명의 사용자화 ......................................................................... 58
6.3.7. Java 메소드의 사용자화 ..................................................................... 59
6.3.8. Java 파라미터의 사용자화 .................................................................. 60
6.3.9. XML 스키마의 사용자화 ..................................................................... 60
6.3.10. 핸들러 체인의 사용자화 .................................................................... 61

제7장 핸들러 프레임워크 ................................................................................. 63
7.1. 개요 ........................................................................................................... 63
7.2. 핸들러 체인의 우선순위 .......................................................................... 64
7.3. 핸들러 클래스 구성 .................................................................................. 64
7.3.1. 핸들러 클래스의 선언 ........................................................................ 64
7.4. 핸들러 클래스 설정 .................................................................................. 65
7.4.1. Java 클래스로부터 웹 서비스 구성 ................................................ 65
7.4.2. WSDL로부터 웹 서비스 구성 ............................................................ 66
7.4.3. 클라이언트의 구성 .............................................................................. 67
7.5. 핸들러 체인을 사용하는 웹 서비스의 예제 .............................................. 67
7.6. 웹 서비스의 핸들러 프레임워크 실행 ...................................................... 70

제8장 프로바이더와 디스패치 인터페이스 ......................................................... 73
8.1. 개요 ........................................................................................................... 73
8.2. 서비스 Endpoint 프로바이더 인터페이스 ............................................ 73
8.2.1. 프로바이더 인터페이스 .................................................................... 73
8.2.2. 프로바이더 인터페이스 예제 .............................................................. 75
8.2.3. 프로바이더 인터페이스 예제 실행 ...................................................... 76
8.3. 클라이언트 디스패치 인터페이스 ............................................................. 77
8.3.1. 디스패치 인터페이스 .......................................................................... 77
8.3.2. 디스패치 인터페이스 예제 ................................................................. 78
8.3.3. 디스패치 예제 실행 ............................................................................. 79
8.4. XML/HTTP 바인딩 .................................................................................. 80
8.4.1. RESTful 웹 서비스 .......................................................................... 81
8.4.2. RESTful 웹 서비스 예제 ................................................................. 82
8.4.3. RESTful 웹 서비스 예제 실행 ........................................................... 84
제9장 비동기 웹 서비스 ..................................................................................................................... 87
  9.1. 개요 ............................................................................................................................................. 87
  9.2. 클라이언트 비동기 오퍼레이션 ................................................................................................. 87
     9.2.1. 비동기 메소드를 가진 SEI Stub 이용 방법 ............................................................................ 87
     9.2.2. 디스패치 인터페이스 이용하는 방법 ...................................................................................... 92
  9.3. 비동기 웹 서비스 ..................................................................................................................... 93
     9.3.1. 비동기 웹 서비스 설정 ............................................................................................................ 93

제10장 MIME Attachment 메시지 전송 ............................................................................................. 95
  10.1. 개요 ......................................................................................................................................... 95
  10.2. MTOM/XOP .............................................................................................................................. 95
     10.2.1. 기본 동작 ............................................................................................................................ 96
     10.2.2. Attachment 바이너리 데이터 크기 설정 ......................................................................... 97
     10.2.3. MTOM/XOP 예제 ............................................................................................................... 98
     10.2.4. MTOM/XOP 예제 실행 ...................................................................................................... 100
  10.3. swaRef ...................................................................................................................................... 101
     10.3.1. swaRef 사용법 ................................................................................................................... 101
     10.3.2. swaRef 예제 ....................................................................................................................... 102
     10.3.3. swaRef 예제 실행 ............................................................................................................... 104
  10.4. 스트리밍 방식으로 첨부 파일을 처리하는 방법 ......................................................................... 105

제11장 Fast Infoset 이용한 웹 서비스 ......................................................................................... 107
  11.1. 개요 ......................................................................................................................................... 107
  11.2. Fast Infoset 사용 ......................................................................................................................... 109
     11.2.1. Content Negotiation .......................................................................................................... 109
  11.3. Fast Infoset 예제 ....................................................................................................................... 109
  11.4. Fast Infoset 웹 서비스 실행 ..................................................................................................... 110

제12장 JAX-WS JMS 기반 전송 ...................................................................................................... 113
  12.1. 개요 ......................................................................................................................................... 113
  12.2. JAX-WS JMS 기반 전송 설정 .................................................................................................... 113
     12.2.1. JMS 서버 설정 .................................................................................................................... 113
     12.2.2. 웹 서비스 작성 .................................................................................................................. 114
     12.2.3. WSDL 설정 ........................................................................................................................ 115
     12.2.4. 웹 서비스 클라이언트 작성 ............................................................................................... 115

제13장 웹 서비스 정책 ..................................................................................................................... 117
  13.1. 개요 ......................................................................................................................................... 117
  13.2. 웹 서비스의 정책(WS-Policy) ................................................................................................. 117
  13.3. 서버 정책 설정 ........................................................................................................................ 119
     13.3.1. WSDL로부터 웹 서비스 구성 ......................................................................................... 119
     13.3.2. Java 클래스로부터 웹 서비스 구성 .................................................................................... 119
  13.4. 클라이언트의 정책 설정 .......................................................................................................... 121

제14장 웹 서비스 Addressing ....................................................................................................... 123
  14.1. 개요 ......................................................................................................................................... 123
<table>
<thead>
<tr>
<th>장</th>
<th>제목</th>
</tr>
</thead>
<tbody>
<tr>
<td>14.2</td>
<td>서버 설정</td>
</tr>
<tr>
<td>14.2.1</td>
<td>Java 클래스로부터 설정</td>
</tr>
<tr>
<td>14.2.2</td>
<td>WSDL로부터 설정</td>
</tr>
<tr>
<td>14.3</td>
<td>클라이언트 설정</td>
</tr>
<tr>
<td>14.4</td>
<td>예제</td>
</tr>
<tr>
<td>14.5</td>
<td>예제 실행</td>
</tr>
<tr>
<td>15</td>
<td>신뢰성 메시징 기술</td>
</tr>
<tr>
<td>15.1</td>
<td>개요</td>
</tr>
<tr>
<td>15.2</td>
<td>서버 설정</td>
</tr>
<tr>
<td>15.2.1</td>
<td>WSDL로부터 설정</td>
</tr>
<tr>
<td>15.2.2</td>
<td>Java 클래스로부터 설정</td>
</tr>
<tr>
<td>15.3</td>
<td>클라이언트 설정</td>
</tr>
<tr>
<td>15.4</td>
<td>예제</td>
</tr>
<tr>
<td>15.5</td>
<td>예제 실행</td>
</tr>
<tr>
<td>16</td>
<td>웹 서비스 트랜잭션</td>
</tr>
<tr>
<td>16.1</td>
<td>개요</td>
</tr>
<tr>
<td>16.2</td>
<td>서버 설정</td>
</tr>
<tr>
<td>16.2.1</td>
<td>WSDL로부터 설정</td>
</tr>
<tr>
<td>16.2.2</td>
<td>Java 클래스로부터 설정</td>
</tr>
<tr>
<td>16.3</td>
<td>클라이언트 설정</td>
</tr>
<tr>
<td>16.4</td>
<td>코디네이터 서비스</td>
</tr>
<tr>
<td>16.5</td>
<td>웹 서비스 트랜잭션 예제</td>
</tr>
<tr>
<td>17</td>
<td>웹 서비스 보안</td>
</tr>
<tr>
<td>17.1</td>
<td>개요</td>
</tr>
<tr>
<td>17.2</td>
<td>전송 수준 보안</td>
</tr>
<tr>
<td>17.3</td>
<td>메시지 수준 보안</td>
</tr>
<tr>
<td>17.3.1</td>
<td>웹 서비스 보안 정책</td>
</tr>
<tr>
<td>17.3.2</td>
<td>웹 서비스 보안</td>
</tr>
<tr>
<td>17.3.3</td>
<td>웹 서비스 보안 대화</td>
</tr>
<tr>
<td>17.3.4</td>
<td>웹 서비스 신뢰</td>
</tr>
<tr>
<td>17.4</td>
<td>메시지 수준 보안 설정</td>
</tr>
<tr>
<td>17.4.1</td>
<td>공통 설정</td>
</tr>
<tr>
<td>17.4.2</td>
<td>사용자명 인증을 통한 대칭 비인딩의 인증 기능 강화</td>
</tr>
<tr>
<td>17.4.3</td>
<td>상호 인증 보안(Mutual Certificates Security)</td>
</tr>
<tr>
<td>17.4.4</td>
<td>SSL을 통한 SAML 인증</td>
</tr>
<tr>
<td>17.4.5</td>
<td>보안 대화(Secure Conversation)</td>
</tr>
<tr>
<td>17.4.6</td>
<td>웹 서비스 신뢰(WS-Trust)</td>
</tr>
<tr>
<td>17.4.7</td>
<td>실행</td>
</tr>
<tr>
<td>17.5</td>
<td>JAX-RPC(JEUS 5) 웹 서비스 보안 이전 방법</td>
</tr>
<tr>
<td>17.5.1</td>
<td>암호화(Encryption)</td>
</tr>
<tr>
<td>17.5.2</td>
<td>서명(Signature)</td>
</tr>
<tr>
<td>17.5.3</td>
<td>Timestamp</td>
</tr>
</tbody>
</table>

vi JEU5 Web Service 안내서
제18장 UDDI 이용 ................................................................. 197
  18.1. 개요 ................................................................................. 197
  18.1.1. UDDI 이용 ..................................................................... 197
  18.2. JEA서서 UDDI 서버 운영 ......................................... 199
    18.2.1. UDDI DataStore 생성 ................................................. 199
    18.2.2. UDDI 서버 Deploying ............................................... 199
    18.2.3. UDDI 서버 구성 설정 ............................................... 200
    18.2.4. 새로운 사용자 추가 .................................................. 200
    18.2.5. UDDI 서버 실행 ...................................................... 202
  18.3. JEA서서서의 UDDI Explorer 사용 ............................. 203
    18.3.1. UDDI 레지스트리 Querying ........................................ 203
    18.3.2. UDDI 레지스트리에 대한 Publish ........................... 204
    18.3.3. JEA UDDI Explorer 설정 ......................................... 209
  18.4. UDDI 클라이언트 생성 ................................................. 211
    18.4.1. UDDI 클라이언트 작성 ............................................... 211
    18.4.2. UDDI 클라이언트 컴파일 ......................................... 213
    18.4.3. UDDI 클라이언트 실행 ............................................... 213
  18.5. XML 디지털 서명 사용법 ............................................. 213
    18.5.1. 디지털 서명(Digital Signature) ................................. 213
    18.5.2. UDDI 클라이언트 XML 서명 생성 방법 ..................... 214
    18.5.3. UDDI 클라이언트 XML 서명 검증 ............................ 215
  18.6. UDDI subscription 사용법 ............................................. 216
    18.6.1. 기본 개념 .................................................................. 216
    18.6.2. 생성 방법 .............................................................. 217
    18.6.3. UDDI subscription 예제 ................................................ 217
    18.6.4. UDDI subscription 클라이언트 프로그래밍 .................. 221
    18.6.5. 이메일 공지를 받기 위한 UDDI 서버 설정 .................. 222
  18.7. UDDI WSDL Publishing 사용법 .................................... 223
    18.7.1. UDDI WSDL Publishing ........................................... 223
    18.7.2. wsd12uddi 사용 ...................................................... 224

제19장 JEA 웹 서비스 XML ............................................... 227
  19.1. 개요 ................................................................. 227
  19.2. JAXB(XML 바인딩을 위한 Java 어키텍처) ..................... 227
    19.2.1. 바인딩 컴파일러(XJC) 관련 프로그래밍 기법 .............. 228
    19.2.2. 스키마 생성기(Schemagen) 관련 프로그래밍 기법 .......... 231
  19.3. JAXP(XML을 다루기 위한 Java 표준 API) ................... 234
    19.3.1. StAX(Java 스크리밍 XML 파서) ............................ 234

II. JAX-RPC 웹 서비스의 지원 ............................................. 235
<table>
<thead>
<tr>
<th>제20장</th>
<th>JAX-RPC 웹 서비스 구현</th>
<th>237</th>
</tr>
</thead>
<tbody>
<tr>
<td>20.1.</td>
<td>개요</td>
<td>237</td>
</tr>
<tr>
<td>20.2.</td>
<td>Java 클래스 웹 서비스의 구현</td>
<td>237</td>
</tr>
<tr>
<td>20.2.1.</td>
<td>간단한 예제</td>
<td>237</td>
</tr>
<tr>
<td>20.2.2.</td>
<td>Java 클래스의 작성 원칙</td>
<td>238</td>
</tr>
<tr>
<td>20.3.</td>
<td>EJB 웹 서비스의 구현</td>
<td>239</td>
</tr>
<tr>
<td>20.3.1.</td>
<td>간단한 예제</td>
<td>239</td>
</tr>
<tr>
<td>20.3.2.</td>
<td>EJB 웹 서비스 작성 원칙</td>
<td>241</td>
</tr>
<tr>
<td>20.4.</td>
<td>WSDL로부터 웹 서비스 구현</td>
<td>241</td>
</tr>
<tr>
<td>20.5.</td>
<td>SAAJ의 사용</td>
<td>273</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>제21장</th>
<th>JAX-RPC 웹 서비스 생성과 배치</th>
<th>243</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.1.</td>
<td>Java 클래스 웹 서비스 생성과 배치</td>
<td>243</td>
</tr>
<tr>
<td>21.1.1.</td>
<td>서비스 설정 파일 작성</td>
<td>243</td>
</tr>
<tr>
<td>21.1.2.</td>
<td>WSDL 파일과 JAX-RPC 매핑 파일 생성</td>
<td>244</td>
</tr>
<tr>
<td>21.1.3.</td>
<td>웹 서비스 DD 파일 작성</td>
<td>245</td>
</tr>
<tr>
<td>21.1.4.</td>
<td>패키징과 배치</td>
<td>247</td>
</tr>
<tr>
<td>21.2.</td>
<td>EJB 웹 서비스 생성과 배치</td>
<td>250</td>
</tr>
<tr>
<td>21.2.1.</td>
<td>서비스 설정 파일 작성</td>
<td>250</td>
</tr>
<tr>
<td>21.2.2.</td>
<td>WSDL 파일과 JAX-RPC 매핑 파일 생성</td>
<td>251</td>
</tr>
<tr>
<td>21.2.3.</td>
<td>웹 서비스 DD 파일 작성</td>
<td>252</td>
</tr>
<tr>
<td>21.2.4.</td>
<td>패키징과 배치</td>
<td>254</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>제22장</th>
<th>JAX-RPC 웹 서비스 호출</th>
<th>257</th>
</tr>
</thead>
<tbody>
<tr>
<td>22.1.</td>
<td>JAX-RPC 웹 서비스 호출(Java SE 클라이언트)</td>
<td>257</td>
</tr>
<tr>
<td>22.1.1.</td>
<td>Stub 클라이언트</td>
<td>257</td>
</tr>
<tr>
<td>22.1.2.</td>
<td>DII 클라이언트</td>
<td>261</td>
</tr>
<tr>
<td>22.2.</td>
<td>JAX-RPC 웹 서비스의 호출(Java EE 클라이언트)</td>
<td>264</td>
</tr>
<tr>
<td>22.2.1.</td>
<td>Java EE 클라이언트 프로그래밍 모델</td>
<td>264</td>
</tr>
<tr>
<td>22.2.2.</td>
<td>Java EE 클라이언트 프로그래밍 절차</td>
<td>265</td>
</tr>
<tr>
<td>22.2.3.</td>
<td>Java EE 클라이언트의 작성과 예제</td>
<td>266</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>제23장</th>
<th>JAX-RPC 웹 서비스 SOAP 메시지 헨들러 생성</th>
<th>273</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.1.</td>
<td>SAAJ 사용</td>
<td>273</td>
</tr>
<tr>
<td>23.1.1.</td>
<td>SOAP 메시지의 생성</td>
<td>274</td>
</tr>
<tr>
<td>23.1.2.</td>
<td>SAAJ 문서 다루기</td>
<td>274</td>
</tr>
<tr>
<td>23.1.3.</td>
<td>SAAJ를 이용한 SOAP 메시지 전송</td>
<td>275</td>
</tr>
<tr>
<td>23.2.</td>
<td>SOAP 메시지 헨들러의 생성</td>
<td>275</td>
</tr>
<tr>
<td>23.2.1.</td>
<td>메시지 헨들러의 생성</td>
<td>276</td>
</tr>
<tr>
<td>23.2.2.</td>
<td>메시지 헨들러와 헨들러 체인의 설계</td>
<td>277</td>
</tr>
<tr>
<td>23.2.3.</td>
<td>헨들러 인터페이스의 구현</td>
<td>277</td>
</tr>
<tr>
<td>23.2.4.</td>
<td>Java EE 웹 서비스 DD 파일 작성</td>
<td>279</td>
</tr>
<tr>
<td>23.2.5.</td>
<td>클라이언트에서 SOAP 메시지 헨들러의 사용</td>
<td>279</td>
</tr>
<tr>
<td>23.2.6.</td>
<td>파일 송수신하는 웹 서비스와 클라이언트 예제</td>
<td>279</td>
</tr>
</tbody>
</table>
제24장  JAX-RPC 웹 서비스 설정 파일 작성 ......................................................... 289
  24.1.  JAX-RPC 웹 서비스 DD 파일 작성 ................................................................. 289
  24.2.  웹 서비스의 매핑 파일 작성 ......................................................................... 294
        24.2.1.  JAX-RPC 매핑 파일 내용 ....................................................................... 294
        24.2.2.  JAX-RPC 매핑 파일 작성 ....................................................................... 295

제25장  JAX-RPC 웹 서비스 데이터 타입 ................................................................. 301
  25.1.  개요 .................................................................................................................... 301
  25.2.  Java와 XML 타입 매핑 ................................................................................... 302
        25.2.1.  내장 타입 매핑 ......................................................................................... 302
        25.2.2.  배열 ............................................................................................................ 304
        25.2.3.  사용자 정의 타입 : JAX-RPC Value Type .............................................. 304
  25.3.  JAX-RPC Value 태입의 사용 ........................................................................ 304
        25.3.1.  JAX-RPC Value 태입을 사용하는 웹 서비스 생성 .................................. 305
        25.3.2.  JAX-RPC Value 태입을 사용하는 웹 서비스 클라이언트 생성 ............... 306
  25.4.  Holder 클래스 .................................................................................................. 308
        25.4.1.  내장 Holder 클래스 .................................................................................. 308
        25.4.2.  사용자 정의 태입을 위한 Holder 클래스 작성 ...................................... 310
  25.5.  Exception과 SOAP Fault ................................................................................ 313
  25.6.  MIME 태임을 DataHandler 태임으로 매핑 ..................................................... 314
        25.6.1.  Wsdl2java에서 dataHandlerOnly 옵션 사용 ........................................ 314
        25.7.  Doc/Literal에서 데이터 바인딩을 사용하지 않기 ....................................... 315
              25.7.1.  Wsdl2java에서 noDataBinding 옵션 사용 .................................... 315

제26장  JAX-RPC 웹 서비스의 보안 ........................................................................ 319
  26.1.  개요 .................................................................................................................... 319
  26.2.  전송 수준 보안 ................................................................................................... 319
  26.3.  메시지 수준 보안 ............................................................................................... 320
        26.3.1.  웹 서비스 보안 적용 ................................................................................ 320
        26.3.2.  웹 서비스 보안 앱서버 사용 .................................................................... 321
        26.3.3.  JEUS 웹 서비스 보안 설정 .................................................................... 322
        26.3.4.  패스워드 Callback 클래스 생성 ................................................................ 323
        26.3.5.  JEUS 웹 서비스 서버 보안 적용 예제 ....................................................... 325
        26.3.6.  JEUS 웹 서비스 클라이언트 보안 적용 예제 .......................................... 329
        26.3.7.  보안 API를 이용한 JEUS 웹 서비스 클라이언트의 작성 .......................... 331
  26.4.  접근 제어 설정 .................................................................................................. 336
        26.4.1.  Java 클래스 웹 서비스 접근 제어 설정 ....................................................... 336
        26.4.2.  EJB 웹 서비스의 접근 제어 설정 ............................................................... 338
        26.4.3.  접근 제어가 설정된 웹 서비스 호출 .......................................................... 340

용어해설 .................................................................................................................... 341
색인 ............................................................................................................................ 343
<table>
<thead>
<tr>
<th>그림 목차</th>
<th>페이지</th>
</tr>
</thead>
<tbody>
<tr>
<td>[그림 2.1] JEOS 웹 서비스의 구조</td>
<td>7</td>
</tr>
<tr>
<td>[그림 7.1] Relationship between the message contexts</td>
<td>63</td>
</tr>
<tr>
<td>[그림 7.2] Handler Framework</td>
<td>64</td>
</tr>
<tr>
<td>[그림 11.1] Fast Infoset의 직렬화/파싱</td>
<td>108</td>
</tr>
<tr>
<td>[그림 17.1] WS-Trust</td>
<td>198</td>
</tr>
<tr>
<td>[그림 18.1] 정보 계층과 주요 XML 태그 이름</td>
<td>198</td>
</tr>
<tr>
<td>[그림 18.2] JEOS UDDI 레지스트리 초기 화면</td>
<td>202</td>
</tr>
<tr>
<td>[그림 18.3] UDDI Search 입력 폼 화면</td>
<td>203</td>
</tr>
<tr>
<td>[그림 18.4] UDDI 레지스트리에서 찾은 결과 화면</td>
<td>204</td>
</tr>
<tr>
<td>[그림 18.5] 등록한 entry 화면</td>
<td>205</td>
</tr>
<tr>
<td>[그림 18.6] Provider 수정 화면</td>
<td>206</td>
</tr>
<tr>
<td>[그림 18.7] 서비스 추가 화면</td>
<td>206</td>
</tr>
<tr>
<td>[그림 18.8] tModel 저장 결과 화면</td>
<td>207</td>
</tr>
<tr>
<td>[그림 18.9] 서비스 바인딩(bindingTemplate) 추가 화면</td>
<td>208</td>
</tr>
<tr>
<td>[그림 18.10] 서비스 바인딩(bindingTemplate) 저장 결과 화면</td>
<td>209</td>
</tr>
<tr>
<td>[그림 18.11] JEOS UDDI Explorer 사용자 관리 화면</td>
<td>210</td>
</tr>
<tr>
<td>[그림 18.12] JEOS UDDI Explorer 사용자 등록 화면</td>
<td>210</td>
</tr>
<tr>
<td>[그림 18.13] UDDI subscription 흐름도</td>
<td>216</td>
</tr>
<tr>
<td>[그림 18.14] WSDL description과 UDDI Data Structure 매핑</td>
<td>223</td>
</tr>
<tr>
<td>[그림 18.15] UDDI에 Publishing된 wsdI의 모습</td>
<td>225</td>
</tr>
<tr>
<td>[그림 23.1] Structure of SOAP with Attachment</td>
<td>273</td>
</tr>
<tr>
<td>[그림 26.1] JEOS 웹 서비스 보안 아키텍처</td>
<td>321</td>
</tr>
</tbody>
</table>
표 목차

[표 10.1]  JAXB 2.0 specification of xmime:expectedContentType to Java type mapping ................. 96
[표 18.1]  uddi.properties .......................................................................................................... 200
[표 18.2]  PUBLISHER 테이블의 컬럼 ........................................................................................ 202
[표 20.1]  Required Mappings : Java to MIME ............................................................................ 242
[표 25.1]  내장 XML/Java 타입 매핑 ........................................................................................... 302
[표 25.2]  내장 Holder 클래스 .................................................................................................... 308
예 목차

[예 3.1]  << Addnumbersimpl.java >> ................................................................................................. 16
[예 3.2]  << build.xml >> .............................................................................................................. 17
[예 3.3]  << AddNumbersImplService.wsdl >> ............................................................................... 18
[예 3.4]  << AddNumbers.java >> ................................................................................................ 19
[예 3.5]  << AddNumbersResponse.java >> ............................................................................... 20
[예 3.6]  << AddNumbers.impl.java >> ........................................................................................ 21
[예 3.7]  << build.xml >> .............................................................................................................. 22
[예 3.8]  << AddNumbersImplService.wsdl >> ............................................................................... 23
[예 3.9]  << AddNumbers.java >> ................................................................................................ 24
[예 3.10] << AddNumbersResponse.java >> ................................................................................ 25
[예 3.11] << AddNumbers.wsdl >> ................................................................................................ 25
[예 3.12] << build.xml >> .............................................................................................................. 28
[예 3.13] << AddNumbersPortType.java >> ................................................................................ 29
[예 3.14] << AddNumbers.java >> ................................................................................................ 30
[예 3.15] << AddNumbersResponse.java >> ............................................................................... 30
[예 3.16] << AddNumbersImpl.java >> ........................................................................................ 31
[예 4.1]  << web.xml >> .............................................................................................................. 35
[예 4.2]  << AddNumbersImpl.java >> ........................................................................................ 36
[예 4.3]  << AddNumbersImpl.java >> ........................................................................................ 36
[예 4.4]  << AddNumbersImpl.java >> ........................................................................................ 37
[예 4.5]  << AddNumbersImpl.java >> ........................................................................................ 37
[예 4.6]  << AddNumbersImpl.java >> ........................................................................................ 38
[예 4.7]  << AddNumbersImpl.java >> ........................................................................................ 38
[예 4.8]  << AddNumbersImpl.java >> ........................................................................................ 39
[예 4.9]  << AddNumbersImpl.java >> ........................................................................................ 39
[예 4.10] << AddNumbersImpl.java >> ........................................................................................ 40
[예 5.2]  << build.xml >> .............................................................................................................. 43
[예 5.3]  << AddNumbersImplService.java >> ............................................................................... 44
[예 5.4]  << AddNumbersImpl.java >> ........................................................................................ 46
[예 5.5]  << AddNumbers.java >> ................................................................................................ 46
[예 5.6]  << AddNumbersResponse.java >> ............................................................................... 47
[예 5.7]  << AddNumbersClient.java >> ......................................................................................... 48
[예 5.8]  << AddNumbersClient.java >> ......................................................................................... 50
[예 6.1]  << custom-client.xml >> ................................................................................................ 54
[예 6.2]  << custom-client.xml >> ................................................................................................ 54
[예 6.3]  << AddNumbers.wsdl >> ................................................................................................ 55
[예 6.4]  << custom-client.xml >> ................................................................................................ 56
[예 6.5]  << custom-client.xml >> ................................................................................................ 56
[예 6.6]  << custom-client.xml >> ................................................................................................ 56
<table>
<thead>
<tr>
<th>Page</th>
<th>File Path</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.7</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>57</td>
</tr>
<tr>
<td>6.8</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>57</td>
</tr>
<tr>
<td>6.9</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>58</td>
</tr>
<tr>
<td>6.10</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>58</td>
</tr>
<tr>
<td>6.11</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>59</td>
</tr>
<tr>
<td>6.12</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>59</td>
</tr>
<tr>
<td>6.13</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>59</td>
</tr>
<tr>
<td>6.14</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>60</td>
</tr>
<tr>
<td>6.15</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>60</td>
</tr>
<tr>
<td>6.16</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>60</td>
</tr>
<tr>
<td>6.17</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>61</td>
</tr>
<tr>
<td>7.1</td>
<td>&lt;&lt; MyLogicalHandler.java &gt;&gt;</td>
<td>64</td>
</tr>
<tr>
<td>7.2</td>
<td>&lt;&lt; MySOAPHandler.java &gt;&gt;</td>
<td>65</td>
</tr>
<tr>
<td>7.3</td>
<td>&lt;&lt; MyLogicalHandler.java &gt;&gt;</td>
<td>65</td>
</tr>
<tr>
<td>7.4</td>
<td>&lt;&lt; MyServiceImpl.java &gt;&gt;</td>
<td>66</td>
</tr>
<tr>
<td>7.5</td>
<td>&lt;&lt; handlers.xml &gt;&gt;</td>
<td>66</td>
</tr>
<tr>
<td>7.6</td>
<td>&lt;&lt; LoggingHandler.java &gt;&gt;</td>
<td>67</td>
</tr>
<tr>
<td>7.7</td>
<td>&lt;&lt; handlers.xml &gt;&gt;</td>
<td>68</td>
</tr>
<tr>
<td>7.8</td>
<td>&lt;&lt; AddNumbersImpl.java &gt;&gt;</td>
<td>69</td>
</tr>
<tr>
<td>7.9</td>
<td>&lt;&lt; AddNumbersClient.java &gt;&gt;</td>
<td>69</td>
</tr>
<tr>
<td>7.10</td>
<td>&lt;&lt; custom-client.xml &gt;&gt;</td>
<td>69</td>
</tr>
<tr>
<td>8.1</td>
<td>&lt;&lt; AddnumbersImpl.java &gt;&gt;</td>
<td>75</td>
</tr>
<tr>
<td>8.2</td>
<td>&lt;&lt; AddNumbersClient.java &gt;&gt;</td>
<td>78</td>
</tr>
<tr>
<td>8.3</td>
<td>&lt;&lt; AddNumbersImpl.java &gt;&gt;</td>
<td>82</td>
</tr>
<tr>
<td>8.4</td>
<td>&lt;&lt; AddNumbersClient.java &gt;&gt;</td>
<td>83</td>
</tr>
<tr>
<td>9.1</td>
<td>&lt;&lt; custom-schema.xml &gt;&gt;</td>
<td>88</td>
</tr>
<tr>
<td>9.2</td>
<td>&lt;&lt; build.xml &gt;&gt;</td>
<td>88</td>
</tr>
<tr>
<td>9.3</td>
<td>&lt;&lt; AddNumbersClient.java &gt;&gt;</td>
<td>89</td>
</tr>
<tr>
<td>9.4</td>
<td>&lt;&lt; AddNumbersClient.java &gt;&gt;</td>
<td>90</td>
</tr>
<tr>
<td>9.5</td>
<td>&lt;&lt; AddNumbersClient.java &gt;&gt;</td>
<td>91</td>
</tr>
<tr>
<td>9.6</td>
<td>&lt;&lt; AddNumbersImpl.java &gt;&gt;</td>
<td>93</td>
</tr>
<tr>
<td>9.7</td>
<td>&lt;&lt; web.xml &gt;&gt;</td>
<td>93</td>
</tr>
<tr>
<td>10.1</td>
<td>&lt;&lt; hello.wsdl &gt;&gt;</td>
<td>98</td>
</tr>
<tr>
<td>10.2</td>
<td>&lt;&lt; hello.wsdl &gt;&gt;</td>
<td>102</td>
</tr>
<tr>
<td>10.3</td>
<td>&lt;&lt; AttachmentApp.java &gt;&gt;</td>
<td>105</td>
</tr>
<tr>
<td>11.1</td>
<td>&lt;&lt; AddNumbersClient.java &gt;&gt;</td>
<td>110</td>
</tr>
<tr>
<td>12.1</td>
<td>&lt;&lt; domain.xml &gt;&gt;</td>
<td>113</td>
</tr>
<tr>
<td>12.2</td>
<td>&lt;&lt; AddNumbersImpl.java &gt;&gt;</td>
<td>114</td>
</tr>
<tr>
<td>12.3</td>
<td>&lt;&lt; AddNumbers.wsdl &gt;&gt;</td>
<td>115</td>
</tr>
<tr>
<td>12.4</td>
<td>&lt;&lt; AddNumbersClient.java &gt;&gt;</td>
<td>115</td>
</tr>
<tr>
<td>13.1</td>
<td>&lt;&lt; service-config.xml &gt;&gt;</td>
<td>119</td>
</tr>
<tr>
<td>14.1</td>
<td>&lt;&lt; AddNumbersImpl.java &gt;&gt;</td>
<td>125</td>
</tr>
<tr>
<td>14.2</td>
<td>&lt;&lt; AddNumbersImpl.java &gt;&gt;</td>
<td>125</td>
</tr>
</tbody>
</table>
메시지 핸들러를 정의하는 <port-component>의 예

메시지 핸들러를 정의하는 <port-component>의 예
[예 23.6]  << web.xml >> ........................................................................................................ 284
[예 23.7]  << jeus-webservices-dd.xml >> ............................................................................. 284
[예 23.8]  << ClientAttachmentHandler.java >> ................................................................. 285
[예 23.9]  << Client.java >> .................................................................................................. 286
[예 24.1]  << webservices.xml >> ......................................................................................... 289
[예 24.2]  DocLitEchoService : <<webservices.xml >> ......................................................... 291
[예 24.5]  AddressBookService : << web.xml >> .............................................................. 292
[예 24.6]  FileAttachmentService : <<webservices.xml >> .................................................. 293
[예 24.7]  JAX-RPC 매핑 파일의 구조 ...................................................................................... 295
[예 24.8]  << AddressBookService.java >> ............................................................................. 297
[예 25.1]  <<Calculator.java>> ............................................................................................. 305
[예 25.2]  <<CalcData.java>> ............................................................................................... 305
[예 25.3]  <<CalculatorIF.java>> .......................................................................................... 306
[예 25.4]  <<CalcClient.java>> .............................................................................................. 306
[예 25.5]  << Calculator.java >> .......................................................................................... 308
[예 25.6]  << CalcClient.java >>의 run() 메소드 >> ............................................................. 309
[예 25.7]  << CalcData.java >> ............................................................................................... 310
[예 25.8]  << CalcDataHolder.java >> ................................................................................... 311
[예 25.9]  << Calculator.java >> ........................................................................................... 313
[예 26.1]  <jeus-webservices-dd.xml> ................................................................................. 322
[예 26.2]  <jeus-web-dd.xml> ............................................................................................... 323
[예 26.3]  < PWCallback.java >> ...................................................................................... 323
[예 26.4]  << Ping.java >> .................................................................................................... 326
[예 26.5]  << PingImpl.java >> ............................................................................................ 326
[예 26.6]  << PingPWCallback.java >> .................................................................................. 327
[예 26.7]  <jeus-webservices-dd.xml> .................................................................................. 327
[예 26.8]  < pingClient.jsp >> ............................................................................................. 329
[예 26.9]  < jeus-web-dd.xml >> ............................................................................................ 330
[예 26.10]  < pingClient.jsp >> ............................................................................................. 331
[예 26.11]  < accounts.xml >> ............................................................................................. 336
[예 26.12]  < jeus-web-dd.xml >> .......................................................................................... 337
[예 26.13]  < web.xml >> ..................................................................................................... 337
[예 26.14]  < accounts.xml >> ............................................................................................. 337
[예 26.15]  < jeus-ejb-dd.xml >> ............................................................................................ 338
[예 26.16]  < ejb-jar.xml >> .................................................................................................... 339
안내서에 대하여

안내서의 대상

본 안내서는 기본적인 웹 서비스의 개념과 Java EE 플랫폼에서 프로그래밍이 가능한 개발자라면 웹 서비스를 모르는 상태에서도 웹 서비스 생성이 가능하게 하기 위한 안내서이다. JEUS 웹 컨테이너 및 JEUS EJB 컨테이너에서 웹 서비스 애플리케이션을 개발, 배치 그리고 유지 보수하는 개발자들을 대상으로 하며 반드시 읽어볼 것을 권장한다.

안내서의 전체 조건

본 안내서를 원활하게 이해하기 위해서는 다음과 같은 사항을 미리 알고 있어야 한다.

- 서블릿 개발과 패키징에 대한 기본적인 이해
- EJB 개념에 관한 기본적인 이해
- XML, SOAP 프로토콜 그리고 WSDL의 개념에 관한 이해

안내서의 제한 조건

본 안내서에서는 서블릿 또는 EJB 기술처럼 기본적인 Java EE에 대한 자세한 설명과 Java EE EJB 스펙에 대한 내용은 설명하지 않는다. 그와 관련된 내용은 관련 안내서를 참조한다.

참고

소스 코드와 관련된 XML 파일들은 JEUS를 설치한 후에 samples 디렉터리에 위치한다.
본 안내서는 총 26개의 장으로 구성되어 있다.

- **제1장 웹 서비스**
  웹 서비스의 기본 개념에 대해 설명한다.

- **제2장 JEUS 웹 서비스**
  JEUS에서 지원하는 웹 서비스의 개념과 지원 방식 및 설계 지침에 대해 설명한다.

- **제3장 JEUS 웹 서비스 구현**
  웹 서비스를 구현하는 여러 가지 방식에 대해 설명한다.

- **제4장 웹 서비스 생성과 배치**
  Java 클래스와 EJB를 Endpoint로 가지는 웹 서비스의 생성과 배치 방법에 대해 설명한다.

- **제5장 웹 서비스 호출**
  웹 서비스를 호출하기 위한 여러 가지 클라이언트 작성과 호출 방법에 대해 설명한다.

- **제6장 표준 바인딩 선언 및 사용자화**
  WSDL 바인딩의 사용자화 선언에 대해 설명한다.

- **제7장 핸들러 프레임워크**
  웹 서비스의 핸들러 프레임워크를 설명한다.

- **제8장 프로바이더와 디스패치 인터페이스**
  메시지를 XML 메시지 수준으로 다루는 것에 대해 설명한다.

- **제9장 비동기 웹 서비스**
  비동기 웹 서비스에 대해 설명한다.

- **제10장 MIME Attachment 메시지 전송**
  MIME 메시지를 이용하여 웹 서비스의 바이너리 데이터를 효과적으로 다루기 위한 방법을 설명한다.

- **제11장 Fast Infoset 이용한 웹 서비스**
  Fast Infoset을 사용하는 웹 서비스에 대해 설명한다.

- **제12장 JAX-WS JMS 기반 전송**
  JMS 기반 전송을 사용하는 웹 서비스에 대해 설명한다.

- **제13장 웹 서비스 정책**
  웹 서비스에서 정책 기능을 설정하고 운용하는 방법에 대해 설명한다.

- **제14장 웹 서비스 Addressing**
  웹 서비스에서 Addressing 기능을 설정하고 운용하는 방법에 대해 설명한다.
제15장 신뢰성 메시징 기술
신뢰성(reliable) 있는 웹 서비스에 대해 설명한다.

제16장 웹 서비스 트랜잭션
웹 트랜잭션에 대해 설명한다.

제17장 웹 서비스 보안
JAX-WS 웹 서비스의 보안 설정에 대해 설명한다.

제18장 UDDI 이용
UDDI 개요와 운영 및 사용 방법에 대해 설명한다.

제19장 JEUS 웹 서비스 XML
JEUS 웹 서비스가 지원하고 있는 XML을 다루기 위한 여러 가지 방식에 대해 설명한다.

제20장 JAX-RPC 웹 서비스 구현
JAX-RPC 웹 서비스를 구현하는 여러 방식에 대해 설명한다.

제21장 JAX-RPC 웹 서비스 생성과 배치
Java 클래스와 EJB를 Endpoint로 가지는 JAX-RPC 웹 서비스의 생성과 배치 방법에 대해 설명한다.

제22장 JAX-RPC 웹 서비스 호출
JAX-RPC 웹 서비스를 호출하기 위한 여러 가지 클라이언트의 작성과 호출 방법에 대해 설명한다.

제23장 JAX-RPC 웹 서비스 SOAP 메시지 핸들러 생성
JAX-RPC 웹 서비스의 메시지 핸들러와 SAAJ를 이용한 SOAP 메시지 처리에 대해 설명한다.

제24장 JAX-RPC 웹 서비스 설정 파일 작성
JAX-RPC 웹 서비스의 표준 웹 서비스 Deployment Descriptor와 JAX-RPC 매핑 파일의 작성 방법에 대해 설명한다.

제25장 JAX-RPC 웹 서비스 데이터 타일
JAX-RPC 브로거에 근거하여 데이터 타입에 관련된 향상된 웹 서비스 프로그래밍에 대해 설명한다.

제26장 JAX-RPC 웹 서비스의 보안
JAX-RPC 웹 서비스의 보안 설정 방법에 대해 설명한다.
안내서 규약

<table>
<thead>
<tr>
<th>표기</th>
<th>의미</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&lt;&lt;AaBbCc123&gt;&gt;&quot;</td>
<td>프로그램 소스 코드의 파일명</td>
</tr>
<tr>
<td>&quot;&lt;Ctrl&gt;+C&quot;</td>
<td>Ctrl과 C를 동시에 누름</td>
</tr>
<tr>
<td>&quot;[Button]&quot;</td>
<td>GUI의 버튼 또는 메뉴 이름</td>
</tr>
<tr>
<td>&quot;진하게&quot;</td>
<td>강조</td>
</tr>
<tr>
<td>&quot;&quot;(따음표)&quot;</td>
<td>다른 관련 안내서 또는 안내서 내의 다른 장 및 절 연급</td>
</tr>
</tbody>
</table>

'입력항목'
화면 UI에서 입력 항목에 대한 설명

| 하이퍼링크 | 메일계정, 웹 사이트 |
| "=" | 메뉴의 진행 순서 |
| '+' | 하위 디렉터리 또는 파일 있음 |
| '-' | 하위 디렉터리 또는 파일 없음 |

참고
참고 또는 주의사항

주의
주의할 사항

[그림 1.1]
그림 이름

[표 1.1]
표 이름

AaBbCc123: Java 코드, XML 문서

[ command argument ] 옵션 파라미터

< xyz >: '〈'와 '〉' 사이의 내용이 실제 값으로 변경됨

| | 선택 사항, 예) A|B: A나 B 중 하나 |
|...| 파라미터 등이 반복되어서 나옴 |
시스템 사용 환경

본 안내서의 모든 예제와 환경 구성은 UNIX의 스타일에 준하여 작성되어 Microsoft Windows™(이하 Windows)와 같이 다른 환경에서 작업하는 경우 몇 가지 사항을 고려해야 한다.

예를 들어 경로 구분자의 경우 UNIX 스타일인 “/”를 Windows 스타일인 “\”로 바꿔서 사용한다. 또한 환경 변수도 Windows 스타일로 변경해서 사용하면 된다.

문서의 내용은 Java 표준을 고려해서 작성했기 때문에 대부분의 내용은 동일하게 적용된다.

관련 안내서

<table>
<thead>
<tr>
<th>안내서</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>JEUS Server 안내서</td>
<td>JEUS 시스템과 서버의 개요와 시스템 관리를 위한 안내서이다.</td>
</tr>
<tr>
<td>JEUS Web Engine 안내서</td>
<td>JEUS 웹 엔진의 관리에 대해 내용과 Java EE WAR Archive, 서블릿/JSP의 관리 및 deploy하는 방법에 대해 기술한 안내서이다.</td>
</tr>
<tr>
<td>JEUS EJB 안내서</td>
<td>JEUS EJB 엔진과 EJB 모듈의 Deploy에 대해 기술한 안내서이다.</td>
</tr>
</tbody>
</table>
참고 자료

- SOAP 1.1 Specification (http://www.w3.org/TR/soap11)
- SOAP 1.2 Specification (http://www.w3.org/TR/soap12)
- WSDL 1.1 Specification (http://www.w3.org/TR/wsd1)
- WS-I Basic Profile 1.0 (http://www.ws-i.org/Profiles/BasicProfile-1.0.html)
- Servlet 2.4 Specification (http://jcp.org/en/jsr/detail?id=154)

JEUS Web Service Annotations and Feature

웹 서비스 Endpoint에서 사용할 수 있는 JEUS Web Service Annotations는 다음과 같다.

- Service Scope
  - @jeus.webservices.jaxws.api.ApplicationScope
    배치(deployment)된 웹 서비스 Endpoint는 하나의 인스턴스를 갖는다. 이 값은 기본값이다.
  - @jeus.webservices.jaxws.api.SessionScope
    각각의 HTTPSession은 하나의 웹 서비스 Endpoint 인스턴스를 생성한다. 각각 다른 인스턴스를 갖는다. HTTP 전송을 사용할 경우에만 사용할 수 있다.
  - @jeus.webservices.jaxws.api.RequestScope
    요청(request) 메시지가 들어올 때마다 새로운 웹 서비스 Endpoint 인스턴스를 생성한다.

- Transport
  - @jeus.webservices.jaxws.api.JMSWebService
    JMS 기반 전송이 가능한 웹 서비스 Endpoint가 된다.
웹 서비스 클라이언트에서 사용할 수 있는 JEUS Web Service Feature는 다음과 같다.

- jeus.webservices.jaxws.api.LocalTransportFeature
  클라이언트의 웹 서비스 Port가 Local 전송을 사용하여 웹 서비스 Endpoint와 통신한다. 웹 서비스 클라이언트와 Endpoint가 동일한 JVM에 존재할 때만 유효하다.

- 그 밖의 설정 및 API에 관련된 자세한 내용은 다음의 문서를 참고한다.
  
  - 콘솔툴 xjc
  - 콘솔툴 schemagen
  - 콘솔툴 tcpmon
  - ANT Task
    위치: JEUS_HOME/docs/reference-book/webservice_appendix_ant_task.html
  - XML Reference - jeus-webservices-config.xml
    XML Reference - jeus-webservices-dd.xml
    XML Reference - jeus-webservices-dd-client.xml
    위치: JEUS_HOME/docs/reference/schema/index.html
Korea

TmaxSoft Co., Ltd
272-6, Seohyeon-dong, Bundang-gu,
Seongnam-si, Gyeonggi-do, 463-824
South Korea
Tel: +82-31-8018-1000
Fax: +82-31-8018-1115
Email: info@tmax.co.kr
Web (Korean): http://www.tmax.co.kr
기술지원: http://technet.tmaxsoft.com

USA

TmaxSoft, Inc.
560 Sylvan Avenue Englewood Cliffs, NJ 07632
U.S.A
Tel: +1-201-567-8266
Fax: +1-201-567-7339
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com

Japan

TmaxSoft Japan Co., Ltd.
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073
Japan
Tel: +81-3-5765-2550
Fax: +81-3-5765-2567
Email: info@tmaxsoft.co.jp
Web (Japanese): http://www.tmaxsoft.co.jp
China

TmaxSoft China Co., Ltd.
Beijing Silver Tower, RM 1508, 2# North Rd Dong San Huan,
Chaoyang District, Beijing, China, 100027
China
Tel: +86-10-6410-6145~8
Fax: +86-10-6410-6144
Email: info.cn@tmaxsoft.com
Web (Chinese): http://www.tmaxsoft.com.cn
제1장 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 메시지를 통해 네트워크로 접근될 수 있는 오픈레이션들을 기술하는 소프트웨어 인터페이스이다. 또한 웹 서비스는 인터넷에만 연결되어 있다면 서비스에 대한 권한을 가지고 있는 사용자 누구에게서도 비즈니스를 공개하고 사용될 수 있도록 메시징 프로토콜, 프로그래밍 표준, 서비스 발간을 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 찰 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 이루어진 소프트웨어 시스템을 100개의 작은 모듈로 나눈 후 각각의 서버 시스템으로 구성하게 되었다. 이러한 작은 모듈화된 서버 소프트웨어 시스템들은 각각 다른 기술로 구현될 수 있게 되었다. 또한 하나의 컴퓨터에 존재하지 않아도 되며, 제사용할 수 있게 되었다. XML과 HTTP와 같은 표준 웹 프로토콜을 사용하여 인터넷의 어떠한 사용자도 쉽게 접근할 수 있게 되었다.

이러한 서비스 지향 기술은 이미 수년전부터 RMI, COM 그리고 CORBA와 같은 여러 다양한 형태의 기술들로 구현되어 왔으며 최근에 발생한 완전히 새로운 형태의 것이라고 보는 힘들다. 하지만 이렇게 언급한 기술들은 전자나 구현된 기술에 종속적이라는 단점이 있다. 하지만 웹 서비스는 이러한 단점을 극복한다.

다음은 웹 서비스의 주요 특징에 대한 설명이다.

- 웹을 통해 접근할 수 있다.
- 스스로 표방하고 서술한다.

웹 서비스는 스스로의 역할과 기능, 속성에 대해서 수록함에 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 혹은 비동기 통신으로 호출된다. 이러한 작동 방식이나 통신 방식과는 무관하게 웹 서비스와 웹 서비스 클라이언트 간에 교환되는 기본적인 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 수정 없이 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 JEUSS 웹 서비스는 다음과 같은 표준을 따르고 있다.

- EWS(Implementing Enterprise Web Services) 표준
- JAX-WS(Java API for XML-based Web Services) 표준
- JAX-RPC(Java API for XML-based RPC) 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
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 Body를 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 환경이나 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 얻고 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 서비스들 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 전영의 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. JAX-RPC 표준

JAX-RPC(Java API for XML-based RPC) 표준은 Sun Microsystems에서 제공하는 웹 서비스 API를 정의 하는 표준이다. 웹 서비스가 해결해야 하는 당면 과제 중 하나는 상호 호환성을 가지는 서비스의 개발이 다. 이러한 문제를 해결하기 위해 WS-I, OASIS, W3C 그리고 SOAPBuilders 같은 다양한 표준 단체들이 수많은 노력을 경주해왔다.

JAX-RPC는 이러한 상호 호환성의 문제를 해결하고 웹 서비스 구현에 대한 유연한 프로그래밍 API를 제공하기 위한 J2EE 1.4 스펙의 Java 전영의 최초의 노력의 산물이다. 이러한 API를 채택함으로 인해 JAX-RPC는 서비스 사용자와 개발자들로 하여금 상호 호환성이라는 아주 큰 점을 Runtime Framework으로 덜게 해 주었다. 이는 후에 JAX-WS라는 웹 서비스를 위한 Framework로 발전한다.
1.2.6. 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 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 가진 메소드를 호출하고 서버는 응답으로 하나의 값을 반환한다. RPC 방식의 메시지 교환 방식에서는 SOAP은 메시지 Body의 형태를 별도로 정의한다.

  - 문서(Document) 방식

    문서 방식에서는 XML 문서가 교환되고 각각의 element의 의미는 서버와 클라이언트가 해석의 둔으 로 남겨둔다. 즉, SOAP 메시지의 Body의 구조에 대한 제약 사항은 SOAP에서 규정하고 있지 않으며, 응용 프로그램이나 또는 별도로 정해진 XML 스키마에 의해 SOAP Body에 위치한 XML 문서의 구조가 결정된다.
제2장 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

Java EE 6 호환 WAS(Web Application Server)인 JEUS는 Java EE 6에서 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 Java EE 6 스펙을 준수하는 벤더들과의 웹 서비스 상호 호환성이 보장된다.

JEUS 웹 서비스의 구조는 다음과 같다.

[그림 2.1] 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. 또한 하위 호환성을 위해 이러한 정보들을
**webservices.xml**과 같은 웹 서비스 Deployment Descriptor(이하 DD)에 설정하는 JAX-RPC 웹 서비스 또한 지원한다.

- 툴과 유필리티

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.


- 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 서비스의 설계

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

#### 2.2.1. 웹 서비스 Back-end의 선택

웹 서비스 컴포넌트는 웹 컨테이너나 EJB 컨테이너에서 실행되는 Java 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 선택할 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.


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. RPC 방식과 문서 방식의 선택

JEUS 웹 서비스는 RPC 방식과 문서(Document) 방식을 지원한다.

- RPC 방식

WSDL 1.1 스펙에 의하면 RPC 방식은 SOAP 메시지가 웹 서비스 오퍼레이션의 인자와 리턴값을 포함하고, 문서 방식은 XML 문서를 SOAP 메시지가 포함한다. 또한 RPC 방식은 원격 프로시저 호출(RPC) 방식의 메시지 교환 방식을 가리키며, 클라이언트와 서버가 잘 정의된 프로그래밍 모델로 만들어져야 한다. 클라이언트는 인자를 가지는 메소드를 호출하고 서버는 응답으로 하나의 값만을 반환한다.

RPC 방식에서는 인자들의 개수에 제한이 없다.

- 문서(Document) 방식

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

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

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>표준 문서 방식</td>
<td>각각의 웹 서비스 오퍼레이션은 1개의 인자(XML 문서)만 가질 수 있다.</td>
</tr>
<tr>
<td>WRAPPED 방식</td>
<td>여러 인자들을 하나의 복합 데이터 타입(complex data type)으로 포장하여 하나의 인자로 만들어 SOAP 메시지에 포함하기 되므로 웹 서비스 오퍼레이션은 여러 개의 인자들을 가질 수 있다.</td>
</tr>
</tbody>
</table>

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 방식과 JAX-RPC 방식으로 구현될 수 있다.

- 서비스 Endpoint로부터 시작

서비스 구현이 단순하고 쉬우며 직관적으로 구현 가능하다.

- WSDL로부터 시작

구현 언어에 중립적으로 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-WS와 JAX-RPC는 어떻게 서비스 Endpoint로부터 WSDL을 도출해야 하는지에 대한 지침을 제공한다. 이 방식은 Java 개발자에게는 웹 서비스 개발을 시작하는 가장 쉬운 방법이다.

- JAX-WS 방식

  JAX-WS 방식에서 유의할 사항은 서비스 설정에 관한 정보를 기술(description)하지 않아도 된다는 것이다. JAX-WS 스펙을 위해 J2EE 웹 서비스 구현에서는 별도의 서비스 설정 파일을 작성하지 않고도 wsgen 톱깃에 의해 웹 서비스를 구현할 수 있다.

  JAX-WS 방식의 웹 서비스 구현에 대한 자세한 내용은 "제3장 J2EE 웹 서비스 구현"을 참고한다.

- JAX-RPC 방식

  JAX-RPC 방식에서 유의할 사항은 서비스 설정에 관한 정보를 기술(description)해야 한다는 것이다. JAX-RPC 스펙은 Java로부터 WSDL로의 매핑에 대해 자세히 정의하고 있다. 그러나 서비스 Endpoint 인터페이스만으로는 완전한 WSDL을 생성할 수 없다. 서비스 Endpoint 인터페이스는 WSDL의 서비스와 바인딩에 대한 정보를 제공하지 못한다. JAX-RPC 스펙을 위해 J2EE 웹 서비스 구현에서는 별도의 서비스 설정 파일을 작성하여 이러한 정보를 Java-to-WSDL 톱깃에 제공해야 한다.

  JAX-RPC 방식의 웹 서비스 구현에 대한 자세한 내용은 "제20장 JAX-RPC 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 중립적으로 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 태그과 용어에서 보다 중립적이다. 즉, Java 환경에 익숙하지 않은 개발자에게도 접근할 수 있는 일반적 명명법으로 WSDL을 작성할 수 있게 한다.

그러나 이 방식은 개발자로 하여금 WSDL에 대한 보다 깊은 이해를 요구한다. JAX-WS 웹 서비스 구현은 "제3장 J2EWS 웹 서비스 구현"에서 설명하고 있다. 또한, JAX-RPC 웹 서비스 구현은 "제20장 JAX-RPC 웹 서비스 구현"에 설명하고 있다.

2.2.4. SOAP 메시지 핸들러의 생성

SOAP 메시지의 Header나 Body 또는 Attachment를 직접 다루려고 할 때 메시지 핸들러의 생성이 필요하다.

- JAX-WS 방식
JAX-WS Handler 프레임워크를 이용할 수 있다. 자세한 설명은 "제7장 핸들러 프레임워크"를 참고한다.

- JAX-RPC 방식

웹 서비스의 경우 SOAP 메시지의 JAX-RPC 표준 핸들러를 생성하고, SAAJ(SOAP with Attachments API for Java)를 이용하여 직접 메시지를 다룰 수 있다. 보다 자세한 설명은 "제23장 JAX-RPC 웹 서비스 SOAP 메시지 핸들러 생성"을 참고한다.
JEUS 7 웹 서비스는 Java EE 6의 웹 서비스 표준인 JAX-WS 웹 서비스를 지원한다. JAX-WS 웹 서비스의 가장 중요한 특징은 설정 파일 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 알아보며 마지막으로 상호 운용성을 위한 웹 서비스, UDDI, JAXR 그리고 XML을 다루는 다양한 방법들에 대해 설명한다.

앞으로 나오는 여러 장들을 통해 이러한 JAX-WS 웹 서비스를 JEUS 7 웹 서비스가 어떻게 지원하고 있는지 자세하게 설명할 것이다.
제3장 JEUS 웹 서비스 구현

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

3.1. 개요

JEUS 7 웹 서비스는 Java EE 6 웹 서비스 표준인 JAX-WS 방식 웹 서비스를 지원한다.

Java EE 6 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 헨들러를 구현하기 위해 이를 직접 개발자가 사용했으나 JAX-WS 웹 서비스 모델에서는 메시지 헨들러를 위한 기본 Framework를 제공하며 SAAJ는 그 뒤에서 동작하게 된다)와 더불어 웹 서비스의 중심 컴퓨터에 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 항상됨에 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 DD(Deployment Descriptor) 파일들을 Annotation으로 처리, POJO(Plain Old Java Object) 방식의 웹 서비스를 구성할 수 있게 되었다.

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

3.2. Java 클래스 웹 서비스의 구현

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

- javax.jws.WebService Annotation을 수반해야 한다.
- 메소드는 javax.jws.WebMethod Annotation을 수반할 수 있다.
모든 메소드의 파라미터들과 반환 타입은 JAXB의 Java로부터 XML 스키마로의 매핑 정의에 명시된 것 이어야 한다.

메소드의 파라미터 또는 반환 타입은 java.rmi.Remote 인터페이스를 직접적적으로 구현하는 것이어서 는 안된다.

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

[예 3.1] << Addnumbersimpl.java >>

```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 Web Service라 설정하고 Endpoint 구현 로직을 구성하면 기본적인 서버의 웹 서비스 프로그래밍은 모두 완료된 것이다. 이렇게 Java Endpoint 구현 클래스를 구현했으면 다음으로는 JEUS 7 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 7 웹 서비스에서 제공해주는 툴인 wsgen를 사용한다.

다음은 wsgen의 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 and Xsoap1.2, the default is soap1.1. Xsoap1.2 is not standard and 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

참고

JEUS 7 웹 서비스는 wsgen의 Ant Task도 지원하는데 보다 자세한 콘솔 명령에 대한 설명과 Ant Task 항목에 대한 설명은 “JEUS Reference Book”의 “5.5.3. wsgen”과 “JEUS Reference Book”의 “5.6.1. wsgen” 참고한다.

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

[예 3.2] << build.xml >>

```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>
```
wsigen 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 오브젝트로 또는 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.

[예 3.3] << AddNumbersImplService.wsdl >>

```xml
<?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"
```

18  JEUS Web Service 안내서
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 AddNumbers message에 대한 Java Bean 클래스이다.

[예 3.4] << AddNumbers.java >>

```java
@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;
}
```
다음은 서비스로부터 리턴되는 메시지인 AddNumbersResponse message element에 대한 Java Bean 클래스이다.

[예 3.5] << AddNumbersResponse.java >>

```java
@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 서비스의 구현

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

JEUS 7에서 제공하는 EJB 3.0 프로그래밍 모델은 서비스 Endpoint 구현 클래스를 더욱 쉽게 작성할 수 있도록 여러 가지 기능들을 제공하고 있는데, javax.ejb.SessionBean 또는 javax.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.

참고

JEUS 7에서 제공하는 기능에 대한 보다 자세한 내용은 "JEUS EJB 안내서"를 참고한다.
JEUS 7에서 제공하는 EJB 3.0 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.

[예 3.6] Addnumbersimpl.java

```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"라 설정하고 Endpoint 구현 로직을 구성하면 기본적인 서버의 웹 서비스 프로그래밍은 모두 완료된 것이다.

이렇게 EJB Endpoint 구현 클래스를 구현했으면 다음으로는 JEUS 7 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 클래스들을 이루어진다. 이러한 Portable Artifact들을 생성하기 위해 JEUS 7 웹 서비스에서 제공해주는 툴인 wsgen을 사용한다.

다음은 wsgen의 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
  and Xsoap1.2, the default is soap1.1.
  Xsoap1.2 is not standard and can onlybe
  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

참고

JEUS 7 웹 서비스는 wsgen의 Ant Task도 지원하는데 보다 자세한 콘솔 명령에 대한 설명과 Ant Task
항목에 대한 설명은 “JEUS Reference Book”의 “5.5.3. wsgen”과 “JEUS Reference Book”의 “5.6.1.
wsgen” 참고한다.

wsgen 툴을 사용하여 여기서 생성한 Java Endpoint 구현 클래스를 사용해서 다음과 같이 Portable Artifact
d들을 생성한다.

[예 3.7] << build.xml >>

...
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 오브젝트로 또는 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.

[예 3.8] << AddNumbersImplService.wsdl >>

```xml
<?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>
</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 AddNumbersResponse이다. 다음은 클라이언트로부터 호출되는 메시지인 AddNumbers message에 대한 Java Bean 클래스이다.

[예 3.9] << AddNumbers.java >>

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

public int getArg0() {
    return this.arg0;
}
```
다음은 서비스로부터 리턴되는 메시지인 `AddNumbersResponse message element`에 대한 Java Bean 클래스이다.

**예 3.10)** << AddNumbersResponse.java >>

```java
@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 7 웹 서비스에서 제공하는 둔인 `wsimport`를 사용해서 우선 서비스 `Endpoint` 인터페이스를 생성해야 한다.

다음은 본 절에서 사용하는 WSDL 파일의 예이다.

**예 3.11)** << AddNumbers.wsdl >>

```xml
<?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"/>
이와 같이 WSDL 파일을 구현했으면 다음으로는 JEUS 7 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 external jaxws or jaxb binding files (Each <file> must have its own -b)
  -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
  -http proxy:<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
```
specification version. version 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

참고
JEUS 7 웹 서비스는 wsimport의 Ant Task도 지원하는데, 보다 자세한 콘솔 명령에 대한 설명과 Ant Task 항목에 대한 설명은 “JEUS Reference Book”의 “5.5.4. wsimport”와 “JEUS Reference Book”의 “5.6.2. wsimport”을 참고한다.

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

[예 3.12] << build.xml >>

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

fromwssl/server/AddNumbers.class
fromwssl/server/AddNumbersPortType.class
fromwssl/server/AddNumbersResponse.class
fromwssl/server/AddNumbersService.class
AddNumbers와 AddNumbersResponse 파일들은 각각 JAXB가 addNumbers 메소드에 대한 요청과 응답을 Java 오브젝트로 또는 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 파일이다.

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

[예 3.13] << AddNumbersPortType.java >>

```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.

주의

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

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

[예 3.16] << AddNumbersImpl.java >>

```java
package fromwsdl.server;

@javax.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.
제4장 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 7에서 동작시키는 방법에 대해 설명한다.

4.1. 개요

기본적으로 웹 서비스를 Java 클래스로부터 구현하는 웹 서비스는 POJO 방식의 웹 서비스 구현을 지향하고 있으므로 이러한 DD(Deployment Descriptor)의 개수는 JAX-RPC 웹 서비스에 비해 대폭적으로 줄어든다.

JEUS 5 JAX-RPC 방식의 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
```


이와 같이 생성된 웹 서비스 WAR 파일은 JEUS 7에서 제공하는 방식으로 deploy되어 접근할 수 있다. 생성된 WAR 파일을 JEUS 7에 deploy하기 위해 콘솔에서 다음의 명령을 실행한다.

```
$ ant build deploy
```
정상적으로 `deploy`되면 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 "4.5. Endpoint 주소 결정 방식"을 참고한다.

이와 같이 생성된 웹 서비스 JAR 파일은 JEUS 7에서 제공하는 방식으로 `deploy`되어 접근할 수 있다. 생성된 JAR 파일을 JEUS 7에 `deploy`하기 위해 콘솔에서 다음과 같은 명령을 실행한다.

```
$ ant build deploy
```

정상적으로 `deploy`되면 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
```
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 “4.5. Endpoint 주소 결정 방식”을 참고한다.

이와 같이 생성된 웹 서비스 WAR 파일은 JEUS 7에서 제공하는 방식으로 deploy되어 웹 서비스로 공개 될 수 있다. 생성된 WAR 파일은 JEUS 7에 deploy하기 위해 콘솔에서 다음과 같은 명령을 실행한다.

```bash
$ 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 webservices.xml 등의 기존 JEUS 5의 JAX-RPC 웹 서비스에서 필요했던 DD 파일이 없어도(Descriptor-free) JEUS 7에 deploy가 가능하다.

본 절에서는 서블릿 기반의 웹 서비스와 EJB 기반의 웹 서비스에 대해 이러한 JAX-WS가 JEUS 7에 deploy 될 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.

   **[예 4.1] << web.xml >>**

   ```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>
   ```
[예 4.2] << AddNumbersImpl.java >>

```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.

[예 4.3] << AddNumbersImpl.java >>

```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.
4.4] << AddNumbersImpl.java >>

```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”

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

다음은 Endpoint 클래스의 예이다.

[예 4.5] << AddNumbersImpl.java >>

```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.

[예 4.6] << AddNumbersImpl.java >>

```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

- 컨텍스트(context)
  @jeus.webservices.annotation.EndpointDescription Annotation에 contextPath 속성을 사용하는 경우 컨텍스트 기본값을 overwrite한다.

다음은 Endpoint 클래스에 대한 예이다.

[예 4.7] << AddNumbersImpl.java >>

```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.

[예 4.8] << AddNumbersImpl.java >>

```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

- 컨텍스트(context)
  
  @WebService Annotation에 serviceName 속성을 사용하는 경우 컨텍스트 기본값을 overwrite한다.

다음은 Endpoint 클래스의 예이다.

[예 4.9] << AddNumbersImpl.java >>

```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.

[예 4.10] << AddNumbersImpl.java >>

```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

- 컨텍스트(context)

컨텍스트(context) 기본값은 그 웹 서비스의 WSDL 문서의 ServiceName이다. 또한, 이 ServiceName은 Endpoint 클래스 이름에 Service를 붙힌 값이 기본값으로 설정된다.
따라서 위의 Endpoint 클래스의 웹 서비스에 대해 다음과 같은 주소로 접근할 수 있다.

http://server:port/AddNumbersImplService/AddNumbersImpl
제5장 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 Java EE 방식으로 나누어진다. 이 2가지 방식은 모두 공통적으로 Client Artifact들을 생성하는 것을 전제로 하고 있다.

본절에서는 이러한 Client Artifact들의 생성 방법에 대해 설명하고, Java SE 방식 그리고 Java 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 결과 그 서비스가 Publish하는 WSDL 파일을 이용하여 Client Artifact들을 생성한 후 호출하는 Java 클래스의 작성을 통해 이루어진다. 따라서 본 장에서는 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 7 웹 서비스에서 제공하는 툴인 wsimport를 통해 얻어진 서비스 인터페이스 클래스들과 서비스 Endpoint 인터페이스 클래스로 이루어진다.

콘솔에서 wsimport를 이용해서 Client Artifact들을 얻는 방법은 다음과 같다.

```bash
$ wsimport -help
Usage: wsimport [options] <WSDL_URI>

where [options] include:
  -b <path>  specify external jaxws or jaxb binding files (Each <file> must have its own -b)
  -catalog <file>  specify catalog file to resolve external
```

제5장 웹 서비스 호출 41
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

-http_proxy:<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 specification version. version 2.0 will generate compliant code for JAXWS 2.0 spec.

-verbose
output messages about what the compiler is doing

-version
print version information

-wsdl_location <location>
@WebServiceClient.wsdlLocation value

Examples:
wsimport stock.wsdl -b stock.xml -b stock.xjb
wsimport -d generated http://example.org/stock?wsdl
예 5.2] << build.xml >>

...
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 서버스 인터페이스 클래스인 AddNumbersImplService 클래스이다.

[예 5.3] << AddNumbersImplService.java >>

```java
@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 __getWsdlLocation() {
    if (ADDNUMBERSIMPLSERVICE_EXCEPTION!= null) {
        return URL.valueOf("http://server.fromjava/"
                           + "AddNumbersImplPort");
    }
    return null;
}
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.

[J5.4] << AddNumbersImpl.java >>

```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.

[J5.5] << AddNumbers.java >>

```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;
}

@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 "5.2.1. 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 프로그램의 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.

[예 5.7] << AddNumbersClient.java >>

```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 Webservices 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 프로그램의 호출

본 절에서는 지금까지 구현한 클래스들 및 기타 설정 파일들을 사용하여 웹 서비스를 구현하여 JEOS 7에 deploy한 후 클라이언트를 실행하는 방법을 설명한다.

원격의 웹 서비스에 접근하는 클라이언트 프로그램의 호출 방식은 그 서비스가 Java로부터 구현하거나 WSDL로부터 구현하거나 둘 다 있으나.

다음과 같이 Java로부터 구현한다면 fromjava 디렉터리에서 서비스를 생성하고, WSDL 파일로부터 구현 한다면 fromwsdl 디렉터리에서 서비스를 생성하여 JEOS 7에 deploy한다.

```
$ ant build deploy
```
위의 과정이 모두 실행되어 서비스가 정상적으로 **deploy**되면, 클라이언트를 빌드한다. Java SE 클라이언트에서 **wsimport**의 과정을 거치므로 서비스의 **deploy**가 모두 완료되었을 때 클라이언트의 구성이 가능하다.

다음과 같이 클라이언트를 생성하고 서비스를 호출한다.

```java
$ ant run
...
run:
[java] ####################################################################
[java] ##### JAX-WS Webservices 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. Java EE 클라이언트 방식

본 절에서는 간단히 앞 장에서 WSDL 문서를 통해 생성한 웹 서비스를 호출하는 클라이언트를 Java EE 방식으로 작성하고 호출한다. 시작하기 전에 우선 “5.2.1. Client Artifact 생성”에서 설명한 방식으로 클라이언트 Portable Artifact들을 얻은 것을 가정한다.

5.2.3.1. Java EE 클라이언트 프로그램의 작성

Java EE 클라이언트 방식에서 중요한 점은 기존 JAX-RPC 환경의 Java 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 Java EE 클라이언트 구성은 @WebServiceRef Annotation을 설정함으로써 그와 동일한 작업이 가능하다는 것이다. 다음은 @WebServiceRef Annotation을 설정하는 하나의 예이다.

```java
... @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. 그러나 이 멤버 변수는 컨타임 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.

[예 5.8] << AddNumbersClient.java >>

```java
public class AddNumbersClient extends HttpServlet {

@WebServiceRef
static AddNumbersImplService svc;

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

    int number1 = 10;
    int number2 = 20;

    System.out.println("""" Invoking addNumbers(" + number1 + ", " + number2 + ")"""");
    int result = port.addNumbers(number1, number2);
    System.out.println("""" Result: " + result);
}
}
```

5.2.3.2. Java EE 클라이언트 프로그램의 호출

지금까지 구현한 클래스들 및 기타 설정 파일들을 사용하여 웹 서비스를 구현하여 JEU 7에 deploy한 후 Java 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 구현 한다면 fromwsdl 디렉터리에서 서비스를 생성하여 JEU 7에 deploy한다.

```
$ ant build deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, Java EE 클라이언트를 빌드한다. Java EE 클라이언트에서 wsimport의 과정을 거치므로 서비스의 deploy가 모두 완료되었을 때 클라이언트의 구성이 가능하다.
다음과 같이 클라이언트를 생성하고 호출한다.

```bash
$ 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 javax.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 (javax.activation.DataSource)으로 레스트(REST) 웹 서비스를 생성할 때 사용될 수 있다.

이에 대한 자세한 내용은 "제8장 프로바이더와 디스패치 인터페이스"를 참고한다.
제6장 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 클래스로부터 또는 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 wsiomt 툴을 통해 얻을 수 있다.

본 장에서는 WSDL 문서를 가지고 wsiomt 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 얻는 웹 서비스를 구현하는 방법은 계속되는 다음 장에서 설명한다.

JEUS 웹 서비스는 WSDL 문서에서 Java 클래스로의 바인딩 선언 및 사용자화(주로 wsiomt 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 7 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(Exceptio) 클래스 등과 같은 Java 언어로의 매핑을 사용자화할 수 있다.
• 비동기형, 프로바이더, 래퍼(Wraper) 방식, 부가적인 Header들과 같은 기능을 사용자화할 수 있다.

6.2. 표준 바인딩 선언


바인딩을 선언하는 장소에 따라 2가지로 나눌 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 값을 가지고 추가한다.

[예 6.1] << custom-client.xml >>

```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.

[예 6.2] << custom-client.xml >>

```xml
<jaxws:bindings
    wsdlLocation="http://localhost:8088/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.

```
```

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.
위의 예제는 WSDL 문서 내에서 `<wsdl:portType>`에 대한 사용자화를 나타낸다.

`<jaxws:bindings>`라는 확장 element는 이 `<wsdl:portType>`으로부터 생성되는 서비스 Endpoint 인터페이스 클래스의 이름에 대한 사용자화를 나타내는데, 표준 기본값으로 생성되는 클래스 이름(여기에서는 `AddNumbersImpl`)을 `MathUtil`로 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 유호하며 `jaxws:bindings@wsdlLocation`에 명시한 WSDL 문서의 `<wsdl:definition>` 내 전체에 적용된다.

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

전체적인 바인딩에 해당하는 element들은 위와 같다.

사용되는 예는 다음과 같다.
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.

이러한 패키지명의 사용자화는 다음과 같이 나타낼 수 있다.

[예 6.5] << custom-client.xml >>

```xml
<jaxws:bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsd/"
  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.

[예 6.6] << custom-client.xml >>

```
<jaxws:bindings
  xmlns:wsdl="http://schemas.xmlsoap.org/wsd/"
  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"/>
  </bindings>
</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.

[예 6.7] << custom-client.xml >>

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 적용 된다.
- 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 "제9장 비동기 웹 서비스"에서 자세히 설명한다.

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

[예 6.8] << custom-client.xml >>
6.3.5. 프로바이더 인터페이스

본 절에서는 프로바이더(provider) 인터페이스를 바인딩 선언을 통해 설정하는 방법을 알아본다. 자세한 내용은 "제8장 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:server는 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.

[예 6.9] << custom-client.xml >>

```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.

[예 6.10] << custom-client.xml >>

```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 예에 예외(Exception) 클래스명은 기본값인 'AddNumbersExeption'이 아닌 'TmaxException'이 된다.

● 서비스 클래스
다음은 서비스 클래스명에 대한 바인딩 사용자화의 예이다.

[예 6.11] << custom-client.xml >>

```xml
<jaxws:bindings node="wsdl:definitions/
    wsdl:service[@name='AddNumbersService']">
    <jaxws:method name="add" />
</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.

[예 6.12] << custom-client.xml >>

```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.

[예 6.13] << custom-client.xml >>

```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 혹은 wsdl:portType의 메소드 파라미터를 변경할 수 있다.

[예 6.14] << custom-client.xml >>

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.

[예 6.15] << custom-client.xml >>

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

[예 6.16] << custom-client.xml >>
외부 표준 JAXB 바인딩 선언 파일은 wsimport 터널 `-b` 소위치에 의해 전달된다.

6.3.10. 핸들러 체인의 사용자화

jaxws:bindings 바인딩 선언은 핸들러를 추가하거나 사용자화하는 데에도 사용된다. 핸들러에 대한 자세한 사항은 “제7장 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.

[예 6.17] << custom-client.xml >>

```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="http://java.sun.com/xml/ns/jaxws"
    xmlns:javaee="http://java.sun.com/xml/ns/javaee">
    <jaxws:bindings node="wsdl:definitions">
        <javaee:handler-chain>
            <javaee:handler-chain-name>
                LoggingHandlers
            </javaee:handler-chain-name>
            <javaee:handler>
                <javaee:handler-name>Logger</javaee:handler-name>
                <javaee:handler-class>
                    fromwsdlhandler.common.LoggingHandler
                </javaee:handler-class>
            </javaee:handler>
        </javaee:handler-chain>
    </jaxws:bindings>
</jaxws:bindings>
```

이와 같은 외부 바인딩 선언 문서와 WSDL 문서를 wsimport 터널을 통해 Portable Artifact들을 생성하면 핸들러 설정 파일이 또한 생성된다. 생성된 서비스 Endpoint 인터페이스 클래스에는 JAX-WS 런타임이 핸들러들을 찾기 위한 @javax.jws.HandlerChain Annotation이 추가된다.
본 장에서는 웹 서비스의 핸들러 프레임워크에 대한 기본 개념 및 구성과 예를 통해 설계 방법에 대해 설명한다.

7.1. 개요

JAX-WS 웹 서비스는 핸들러를 위한 보다 쉬운 플러그인 형태의 프레임워크를 제공하는데, 이는 JEOS 7 웹 서비스의 컨테이너(Runtime) 시스템 기능을 보다 향상시킬 수 있다.

이러한 핸들러의 종류는 다음과 같은 2가지로 구분할 수 있다.
- 논리적 핸들러(Logical Handler)
  프로토콜에 무관하게 메시지의 페이지드(payload)에 접근할 수 있는 핸들러이다.
- SOAP 핸들러(SOAP Handler)
  header를 포함한 SOAP 메시지 전체에 접근할 수 있는 핸들러이다.

[그림 7.1] 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 페이지드만을 필요로 할 경우에는 논리적 핸들러를 사용한다.

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

핸들러 체인은 다음의 그림과 같이 왜이어(Wire) 상에서 나가는(Outbound) 메시지인 경우 모든 Logical
핸들러가 SOAP 핸들러보다 앞서 처리된다. 반대로 들어오는(Inbound) 메시지인 경우 모든 SOAP 핸들러
가 Logical 핸들러보다 앞서 처리된다.

주의

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

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 클
래스로 작성한다.

다음은 각각의 핸들러 클래스의 예이다.

[예 7.1] << MyLogicalHandler.java >>

```java
public class MyLogicalHandler implements LogicalHandler<LogicalMessageContext> {
    public boolean handleMessage(LogicalMessageContext messageContext) {
        LogicalMessage msg = messageContext.getMessage();
        return true;
    }
}
```
위와 같이 Logical 핸들러, SOAP 핸들러는 공통적으로 Handler 인터페이스를 구현하고 있다. Handler 인터페이스는 handlerMessage() 그리고 handleFault() 메소드를 지원하고 있다.

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

7.4. 핸들러 클래스 설정

본 절에서는 구성한 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.
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.

```xml
<bindings xmlns="http://java.sun.com/xml/ns/jaxws">
  <handler-chains xmlns:jws="http://java.sun.com/xml/ns/javaee">
    <jws:handler-chain>
      <jws:handler>
        <jws:handler-class>fromwsdl.handler.TmaxHandler</jws:handler-class>
      </jws:handler>
    </jws:handler-chain>
  </jws: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.

```xml
<handler-chains xmlns="http://java.sun.com/xml/ns/javaee">
  <handler-chain>
    <service-name-pattern xmlns:tm="http://tmaxsoft.com">
    tm:Tmax*Service
  </service-name-pattern>
</handler-chains>
```
위와 같이 여러 개의 `<handler-chain>`이 하나의 `<handler-chains>`로 구성될 때에는 어떤 핸들러에 서비스 이름이나 포트 이름 혹은 프로토콜과 같은 속성을 부여할 수 있다.

7.4.3. 클라이언트의 구성

위 서비스의 구성 중 WSDL로부터 웹 서비스를 구성하는 방법과 동일하다.

7.5. 핸들러 체인을 사용하는 웹 서비스의 예제

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


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

[예 7.6] << LoggingHandler.java >>

```java
public class LoggingHandler implements SOAPHandler<SOAPMessageContext> {
    public Set<QName> getHeaders() {
        return null;
    }

    public void close(MessageContext messageContext) {
```

제7장 핸들러 프레임워크 67
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;
}
다음은 @HandlerChain에 등록된 파일인 handlers.xml의 예이다.

[예 7.8] << AddNumbersImpl.java >>

```java
@WebService
@HandlerChain(file = "handlers.xml")
public class AddNumbersImpl {
    public int addNumbers(int number1, int number2) {
        return number1 + number2;
    }
}
```

**클라이언트 클래스**

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

[예 7.9] << AddNumbersClient.java >>

```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.

[예 7.10] << custom-client.xml >>

```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="http://java.sun.com/xml/ns/jaaxe">
        <jws:handler-chains>
            <jws:handler-chain>
                <jws:handler>
                    <jws:handler-class>
                    fromjavahandler.common.LoggingHandler
```
7.6. 웹 서비스의 핸들러 프레임워크 실행

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

다음과 같이 핸들러 프레임워크를 설정한 서비스를 생성하여 JEUS에 deploy합니다.

```
$ ant build deploy
```
위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되며, 클라이언트를 발드하고 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 Webservices examples - handler ##
[java] #=====================================================================

[java] Client message:
[java] #=====================================================================
[java] ## JAX-WS Webservices examples - handler ##
[java] #=====================================================================

[java] Server message:
```

70 JEUS Web Service 안내서
30

BUILD SUCCESSFUL
제8장 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 7 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. @javax.xml.ws.WebServiceProvider Annotation을 가지고 있어야 한다.

2. 해당 클래스는 Provider<javax.xml.transform.Source>, Provider<javax.xml.soap.SOAPMessage>, Provider<javax.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<javax.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<javax.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 null값이라면 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 @javax.xml.ws.ServiceMode Annotation이 담당한다.

  @javax.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 Artifact 들 중 어떤 포트들은 프로바이더 인터페이스를 구현하는 서비스 구현 클래스에 의해 실제로 사용되지 않게 된다. 따라서 WSDL로부터 wsimport 툴을 사용하여 Portable Artifact 들을 생성하는 데 성능의 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.

[예 8.1] << AddnumbersImpl.java >>

```
@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 =
            Source sumsourse =
                new StreamSource(new ByteArrayInputStream(body.getBytes()));

            return sumsourse;
        } 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 구현 하는 웹 서비스를 실행하는 법을 설명한다.

다음과 같이 프로바이더 인터페이스를 구현하는 웹 서비스를 생성하여 JEUS에 deploy한다.

```bash
$ ant deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되었으면, 클라이언트를 빌드한다. 클라이언트에서 wsimport의 과정을 거쳐서 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.

```bash
$ ant run
...

run:

 java] ###################################################################################################################################
 java] ### JAX-WS Webservices examples - Provider ###
 java] ###################################################################################################################################
 java] Testing Provider webservice...
 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 2.2 - JEUS 7

--------------
---[HTTP response 200]---
```
8.3. 클라이언트 디스패치 인터페이스

본 절에서는 클라이언트 디스패치 인터페이스에 대해 설명한다.

8.3.1. 디스패치 인터페이스

서비스 Endpoint의 프로바이더 인터페이스와 대칭되는 개념으로 클라이언트에서도 애플리케이션의 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.

   
   서비스 Endpoint의 프로바이더 인터페이스가 Provider(Source)를 구현해서 클라이언트의 메시지로부터 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(javax.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.

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

또한 이렇게 생성된 서비스 객체에 대해 디스패치 객체가 바인드되는 특정의 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.
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 둘에 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 둘을 통해 생성된 Portable Artifact들 중 서비스 객체를 생성하는 예제 코드이다.

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

3. 디스패치 객체를 생성한다.

서비스 객체를 동적으로 또는 WSDL 파일로부터 생성했으면 디스패치 객체는 다음의 서비스 클래스의 2가지 메소드를 통해 생성된다.

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


또한 디스패치 객체가 javax.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.

[예 8.2]  << AddNumbersClient.java >>

```java
public class AddNumbersClient {

    public static void main(String[] args) {
        try {
            Service service = new AddNumbersService();
            String request = "<addNumbers xmlns="http://tmaxsoft.com">
```
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 법은 다음과 같다.

다음과 같이 디스패치 인터페이스를 구현하는 웹 서비스를 생성하여 JEUS에 deploy한다.

```
$ ant deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 빌드한다. 클라이언트에서 wsimport의 과정을 거쳐서 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 Webservices examples - Dispatch ###
  [java] ##############################################
  [java] Testing Dispatch webservice...
  [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 2.2 - JEUS 7

---[HTTP response 200]---

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 서비스라고 한다. 기본적으로 RESTful 웹 서비스의 서비스와 클라이언트는 프로바이더와 디스패치 인터페이스를 사용한다. 프로바이더와 디스패치 인터페이스는 서비스와 클라이언트가 SOAP 메시지 형식으로 메시지를 주고받도록 하기 위해 작성된다. 이는 RESTful 사양에 따라 해당 서비스가 클라이언트에게 제공할 수 있도록 하는 전략을 의미한다.
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)과 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

"8.2. 서비스 Endpoint 프로바이더 인터페이스"에서 프로바이더 인터페이스를 구현하기 위해 서비스 Endpoint가 Provider<Source> 또는 Provider<SOAPMessage>로 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.

```java
@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 인터페이스를 구성하는 방법의 2에서 살펴본 바와 같이 동적인 서비스 객체를 생성하고 HTTPBinding.HTTP_BINDING으로 포트를 생성해서 다음과 같이 프로그래밍한다.

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

이렇게 생성된 서비스 객체를 사용해서 디스패치 객체를 생성하고, 필요하다면 이 서비스 객체에 Request 를 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.

```java
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);
```

주의

`invoke` 메소드에 아무런 파라미터도 넘기지 않는다는 점에 주의한다.

8.4.2. RESTful 웹 서비스 예제

본 절에서는 RESTful 웹 서비스와 클라이언트 에제에 대해서 설명한다.

RESTful 웹 서비스 예제

다음은 RESTful 웹 서비스에서 서비스에 해당하는 Java 클래스의 예이다.

[예 8.3] << AddNumbersImpl.java >>

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

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.

**[예 8.4] << AddNumbersClient.java >>**

```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("#############################################");
        
        try {
            MessageContext mc = wsContext.getMessageContext();
            String query = (String) mc.get(MessageContext.QUERY_STRING);
            String st = new StringTokenizer(query, "=&/");
            int number1 = Integer.parseInt(st.nextToken());
            int number2 = Integer.parseInt(st.nextToken());
            int sum = number1 + number2;
            String body =
                "<ns:addNumbersResponse xmlns:ns="urn:AddNumbers">" + 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 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 JEOS에 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 webservice...


...

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 2.2 - JEUS 7

-------------------

---[HTTP response 200]---
<ns:addNumbersResponse xmlns:ns="urn:AddNumbers"><ns:return>30</ns:return></ns:addNumbersResponse>

-------------------

...

제8장 프로바이더와 디스패치 인터페이스 85
제9장 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 Servelet 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


9.2.1.1. 비동기화 바인딩 선언

`WSDL`에서 명시된 `wsdl:operation element`를 비동기화된 매핑으로 사용하기 위해서는 JEUS 웹 서비스의 `wsimport`와 같은 둘을 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.

[예 9.1] << custom-schema.xml >>

```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.

[예 9.2] << build.xml >>

```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.

```java
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.

```java
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.

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

위와 같이 Polling 메소드로 매핑된 메소드는 javax.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.

**[예 9.3] << AddNumbersClient.java >>**

```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 Webservices examples - polling ### ");
            System.out.println("############################################# ");
        }
    }
}
```
Callback 메소드를 사용하는 클라이언트의 구성 방법

wsimport를 통해 얻은 비동기 SEI의 Callback 메소드는 다음과 같다.

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

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

다음은 `AsyncHandler`를 구현한 핸들러 객체의 예제 코드이다.

[예 9.4] << AddNumbersClient.java >>

```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();
           }
           }
   }
}
```
위와 같이 추가적인 AsyncHandler를 구현한 핸들러 객체는 런타임에 서버로부터 그 웹 서비스 오퍼레이션의 결과를 얻을 수 있을 때 handleResponse라는 메소드를 호출하게 되고, 클라이언트 애플리케이션은 getResponse() 메소드를 통해 그 결과 값을 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.

```java
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.

[예 9.5] << AddNumbersClient.java >>

```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("#\n\n\nresult: %d\n", output.getReturn());
        } catch (Exception e) {
```

제9장 비동기 웹 서비스 91
9.2.1.3. 비동기화 클라이언트 실행

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

다음과 같이 비동기화 오퍼레이션을 설정한 웹 서비스를 생성하여 JEUSe deploy한다.

```
$ ant build deploy
```
위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면 클라이언트를 찾을 수 있다.

헌들리 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 Webservices examples - polling ###
  [java] #############################################
  [java] call webservices in an Asynchronous Polling way...result : 30
  [java] #############################################
  [java] ### JAX-WS Webservices examples - callback ###
  [java] #############################################
  [java] call webservices 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. AsyncHandler는 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 할당한 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 서버로 설정하는 예제이다.

[예 9.6] << AddNumbersImpl.java >>

```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.

[예 9.7] << web.xml >>

```xml
<web-app>
    <servlet>
        <servlet-name>AddNumbers</servlet-name>
        <servlet-class>fromwsdl.server.AddNumbersImpl</servlet-class>
    </servlet>
</web-app>
```
<async-supported>true</async-supported>
</servlet>
<servlet-mapping>
  <servlet-name>AddNumbers</servlet-name>
  <url-pattern>/addnumbers</url-pattern>
</servlet-mapping>
</web-app>
제10장 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 페이지드의 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.


JEUS 7 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 XOP과 함께 XML binary data such as xs:base64Binary 또는 xs:hexBinary와 같은 XML 바이너리 데이터가 네트워크에 어떻게 최적화되어 전송될 수 있는지에 대해 정의하고 있다.

xs:base64Binary와 같은 XML 태그는 SOAP Envelope에 포함되어 전송되기 때문에 이러한 태그의 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 태그의 element 값이 MIME Attachment의 형태로 XOP 패키징될 때 그 MIME Attachment의 Content-Type은 xmime:expectedContentType 속성에 지정해줄 수 있고 이는 JAXB에서 지원하는 태그 매핑의 적용을 받는다.

xs:base64Binary 또는 xs:hexBinary 태그의 스키마 element는 xmime:expectedContentType 속성과 함께 사용했을 때 JAXB에서 지원하는 태그 매핑을 적용받아 이 element 값의 크기가 어떤 상한선을 넘게 되었을 때 다음과 같은 MIME 태그로 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-stream와 같은 값으로 MIME Attachment에 포함된다.

이러한 xmime:expectedContentType의 Java 태그로의 JAXB 태그 매핑은 다음과 같다.
표 10.1 JAXB 2.0 specification of xmime:expectedContentType to Java type mapping

<table>
<thead>
<tr>
<th>MIME 타입</th>
<th>Java 타입</th>
</tr>
</thead>
<tbody>
<tr>
<td>image/gif</td>
<td>java.awt.Image</td>
</tr>
<tr>
<td>image/jpeg</td>
<td>java.awt.Image</td>
</tr>
<tr>
<td>text/plain</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>text/xml or application/xml</td>
<td>javax.xml.transform.Source</td>
</tr>
<tr>
<td><em>/</em></td>
<td>javax.activation.DataHandler</td>
</tr>
</tbody>
</table>

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

10.2.1. 기본 동작

MTOM/XOP 환경을 이용할 수 있도록 WSDL 문서의 wsdl:type에 xs:base64Binary 또는 xs:hexBinary와 같은 타입으로 정의된 element가 xmime:expectedContentType 속성으로 정의된다. 정의된 속성을 wsimport 를 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 @javax.xml.ws.soap.MTOM Annotation을 추가한다.

```java
@javax.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.

```java
@BindingType(value="javax.xml.ws.SOAPBinding.SOAP11HTTP_MTOM_BINDING")
@BindingType(value="javax.xml.ws.SOAPBinding.SOAP12HTTP_MTOM_BINDING")
```

클라이언트에서의 MTOM 동작

클라이언트에서 MTOM을 동작시키기 위해서는 서비스로부터 프록시(proxy) 또는 디스패치(dispatch) 객체를 javax.xml.ws.soap.MTOMFeature라는 파라미터를 통해 얻는 방법이다.
다음은 프록시 또는 디스패치가 MTOM이 설정되어있는지 확인하는 메소드이다.

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

다음은 실제 WSDL의 `wsdl:type`의 스키마 예제이다.

```xml
<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.

### 10.2.2. Attachment 바이너리 데이터 크기 설정

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

XOP 인코딩되어 메시지에 패키지될 때 원래의 `element`에는 `<xop:Include href=...>`와 같은 값이 설정되는 데 이 `element`의 `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.

- 서버는 @MTOM Annotation로 설정한다.

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

- 클라이언트에서는 MTOMFeature 클래스를 통해 설정한다.

```java
Server port = new ServerService().getServerPort(new MTOMFeature(3000));
javax.xml.ws.Service.createDispatch(..., new javax.xml.ws.soap.MTOMFeature())
```
10.2.3. MTOM/XOP 예제

MTOM/XOP가 적용된 WSDL 파일의 예제는 다음과 같다.

[예 10.1] << hello.wsdl >>

```xml
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:types="http://tmaxsoft.com/mtom/data"
```
<wsdl:types>
  <schema xmlns="http://www.w3.org/2001/XMLSchema"
    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">
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 MTOM을 설정한 서비스를 생성하여 JEOS에 deploy한다.

```
$ ant deploy
```

위의 과정이 모두 실행되어 서비스가 정상적으로 deploy되면, 클라이언트를 볼한다. 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:23ba193c-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 2.2 - JEUS 7
--uuid:23ba193c-bd1a-4323-abdd-339bf5c05cd1
Content-Type: application/xop+xml; charset=utf-8;type="text/xml"
Content-Transfer-Encoding: binary

10.3. swaRef

JAX-WS 웹 서비스는 WSDL의 `wsdl:type` 안에 `element`를 `wsi:swaRef`라는 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 `element`는 `javax.activation.DataHandler`의 형태의 Java 클래스로 매핑된다.

10.3.1. swaRef 사용법

`wsi:swaRef` 타입의 XML `element`는 DataHandler Java 클래스로 매핑되며 실제 네트워크에서는 Attachment의 형태로 전송된다.

예를 들어 다음과 같은 XML 스키마가 WSDL에 정의되어 있다고 가정한다.

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

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

```xml
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">
```

제10장 MIME Attachment 메시지 전송 101
10.3.2. swaRef 예제

swaRef가 적용된 WSDL 파일의 예제는 다음과 같다.

[예 10.2] << hello.wsdl >>

```xml
<?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>
```

102  JEUS Web Service 안내서
위와 같이 claimForm과 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 해더러 프레임워크를 실행하는 방법에 대해서 설명한다.

다음과 같이 swaRef를 설정한 서비스를 생성하여 JEOS에 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 2.2 - JEOS 7
--uuid:26973efe-2e29-436a-8fce-3fa58b6fd91f
Content-Type: text/xml


--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"?>
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
<wsdl:types xmlns:types="http://tmaxsoft.com/swaref/data"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/">
<wsdl:binding name="swaref">
...
10.4. 스트리밍 방식으로 첨부 파일을 처리하는 방법

JEUS 7 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 7 서버는 Chunked Streaming 방식을 지원한다.

[예 10.3] << AttachmentApp.java >>

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

Map<String, Object> reqCnt = ((BindingProvider)port).getRequestContext();
new Integer(4*1024));
```

주의

모든 HTTP 서버가 이 방식을 지원하지는 않는다는 것에 유의한다.
제11장 Fast Infoset 이용한 웹 서비스

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

11.1. 개요

- ITU-T(통신장비 및 시스템의 조합 표준을 육성하기 위한 조직체)
- ISO(International Standards Organization)
  http://www.iso.org/iso/home/standards_development/list_of_iso_technical_committees.htm


Fast Infoset의 설계

Fast Infoset 문서는 XML 문서와 마찬가지로 직렬화되거나 파싱(Parsing)될 수 있다.
- 직렬화 과정
  XML 문서로부터 얻은 DOM 문서 혹은 SAX 이벤트에 해당하는 XML Infoset으로부터 Fast Infoset을 얻고 다시 이 Fast Infoset으로부터 Fast Infoset 문서를 얻는 과정이다.
- 파싱 과정
다음은 이러한 직렬화 또는 파싱 과정을 나타낸 것이다.

[그림 11.1] Fast Infoset의 직렬화/파싱

Fast Infoset의 장점

Fast Infoset 문서는 XML 문서에 비해 파싱하는 속도와 직렬화하는 속도가 빠르며 문서의 크기가 더 작아 진다는 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 빠른다.
– 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 헤더인 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 헤더를 통해 이러한 결정이 이루어진다는 점에서 Content Negotiation 기법이라고 한다.


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

- 클라이언트의 속성을 설정한다.

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

- 클라이언트가 동작하는 VM의 시스템 변수를 설정한다.

```java
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.

참고

서버에서는 다른 옵션을 통한 기능은 제공하지 않는다. Fast Infoset 속성은 모두 클라이언트에 의해서 결정된다.
다음은 Fast Infoset 속성을 위한 클라이언트 애플리케이션의 예제이다.

[예 11.1] << AddNumbersClient.java >>

```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 Webservices examples - fastinfoset ###");
        System.out.println("########################################################");
        System.out.println("Testing Fast Infoset webservices...");
        int result = port.addNumbers(number1, number2);
        if (result == 30) {
            System.out.println("Success!");
        }
    }
}
```

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

### 11.4. Fast Infoset 웹 서비스 실행

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

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

```
$ ant build deploy
```

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

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

```
$ ant run
...
```
run:

```java
### JAX-WS Webservices examples - fastinfoset ###

Testing Fast Infoset webservices...

Success!
```

BUILD SUCCESSFUL

```bash
$ 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 2.2 - JEUS 7

<?xml version="1.0" ?>

```xml
<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]---
제12장 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-factor>, <destination>을 설정한다. 자세한 설정 방법은 "JEUS MQ 안내서"를 참고한다.

다음은 JAX-WS JMS 기반 전송 설정에 대한 예제이다.

[예 12.1] << domain.xml >>

```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>
```
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.

[예 12.2] << AddNumbersImpl.java >>

```java
@WebService
@jeus.webservices.jaxws.api.JMSWebService(
    connectionFactory = "QueueConnectionFactory",
    destination = "ExamplesQueue",
    portName = "AddNumbersJMSPort")
public class AddNumbersImpl {
    ...
}
```

JMSWebService의 속성은 다음과 같다.

<table>
<thead>
<tr>
<th>속성</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectionFactory</td>
<td>JMS 서버의 ConnectionFactory 이름으로 domain.xml#&lt;connection-factory&gt;&lt;name&gt; 값과 일치해야 한다.</td>
</tr>
<tr>
<td>destination</td>
<td>JMS 서버의 Destination 이름으로 domain.xml#&lt;destination&gt;&lt;name&gt; 값과 일치해야 한다.</td>
</tr>
<tr>
<td>portName</td>
<td>fromJava 모델에서는 published WSDL에 공개될 port 이름이다. @JMSWebService#portName 속성이 명시되지 않으면, JAX-WS JMS 기반 전송을 위한 portName은 &quot;JMSTransport&quot; + default_PortName으로 설정된다. 예를 들어 portName을 설정하지 않으면 &quot;JMSTransportAddNumbersImplPort&quot;가 된다.</td>
</tr>
</tbody>
</table>
12.2.3. WSDL 설정

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

다음은 WSDL에 JAX-WS JMS 기반 전송을 위한 wsdl:port를 설정하는 예제이다.

[예 12.3] << AddNumbers.wsdl >>

```xml
<definitions name="AddNumbers" targetNamespace="urn:AddNumbers" ... >

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

  <port name="AddNumbersJMSPort" binding="impl:AddNumbersBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL" />
  </port>

</service>
</definitions>
```

wsdl:port에 설정된 JMS 기반 전송을 위한 <port name>은 웹 서비스 Endpoint의 '@JMSWebService#port Name' 속성값과도 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:port를 가지고 있다. 이 WSDL로부터 JAX-WS JMS 기반 전송을 위한 JAX-WS JMS 기반 전송을 위한 WSDL로부터 JAX-WS JMS 기반 전송을 위한 웹 서비스의 wsimport 도구로 생성된 Service(extends javax.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.

[예 12.4] << AddNumbersClient.java >>

```java
//AddNumbersPortType port = new AddNumbersService().getAddNumbersPort();
AddNumbersPortType port = new AddNumbersService().getAddNumbersJMSPort();
...  
```
제13장 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 서비스에서의 웹 서비스 정책 시나리오는 크게 서버 정책과 클라이언트 정책으로 구분할 수 있다. 서버는 WSDL을 통해 웹 서비스 정책을 노출시킬 수 있고, 클라이언트는 웹 서비스의 정책에 맞는 기능을 자동으로 구성한다.

웹 서비스의 각각 정책 설정은 "제14장 웹 서비스 Addressing", "제15장 신뢰성 메시징 기술", "제16장 웹 서비스 트랜잭션", "제17장 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.

참고

웹 서비스 정책 프레임워크(Framework)

다음은 웹 서비스 정책 프레임워크에 대한 설명이다.

- 정책 컨테이너(Policy Container)

웹 서비스 정책 프레임워크에서 가장 핵심이 되는 주요 컨테이너는 "Policy"라는 element로 표현된 정책 컨테이너이다. 이 element는 ID 값을 부여받을 수 있어 다른 곳에서 이를 참조하거나 재사용할 수 있
다. 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 전제 또는 연산자들이 여러 개인 경우 이 중 어느 하나만을 선택하여 정책으로 가지고 있겠다는 것을 나타내기 위한 연산자이다.

다음은 연산자 사용에 대한 예이다.

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

  - All Operator

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

다음은 연산자 사용에 대한 예이다.

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

  - Optional Operator

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

다음은 연산자 사용에 대한 예이다.

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

```xml
<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
```

d음은 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.

[예 13.1] << service-config.xml >>

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
```
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 패키지될 WEB-INF 폴더 아래에 위치시킨다.
5. 패키지된 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.
13.4. 클라이언트의 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 톨을 통해 클라이언트 Java Bean 객체들을 구성한다.
2. 원격의 WSDL 문서를 접근 가능한 저장소에 wsit-client.xml이라는 이름으로 저장한다.

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

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

```
jar_root
 I- classes
```
I- ... (client classes, JAX-WS artifacts, Handler, Validator)
I- META-INF
    I- wsit-client.xml
본 장에서는 트랜스포트에 독립적인 웹 서비스 Addressing과 그에 대한 간단한 예제를 통해 사용법을 설명한다.

### 14.1. 개요


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 나타난다.

```xml
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)를 정의해서 SOAP 메시지 혹은 WSDL 문서로의 바인딩(binding)을 규정해 놓았다.
다음은 메시지 레벨에서 이러한 MAP의 정보가 담긴 웹 서비스 Addressing을 설정한 예시이다.

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

<table>
<thead>
<tr>
<th>element</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="">wsa:MessageID</a></td>
<td>메시지를 유일하게 구분할 수 있도록 하는 절대 URI로 설정된 값이다.</td>
</tr>
<tr>
<td><a href="">wsa:To</a></td>
<td>메시지를 전달받도록 의도된 곳의 주소를 나타내기 위한 절대 URI로 설정된 값이다.</td>
</tr>
<tr>
<td><a href="">wsa:Action</a></td>
<td>메시지가 의미하는 바를 나타내기 위한 절대 URI로 설정된 값이다.</td>
</tr>
</tbody>
</table>

이와 같은 웹 서비스 Addressing 메시지는 모든 정보들이 트랜스포트 또는 애플리케이션과 독립적으로 프로세스될 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.com과 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 `javax.jws.WebService Annotation`과 더불어 `javax.xml.ws.soap.Addressing Annotation`을 추가하면 모든 웹 서비스 Addressing 설정은 완료된다.

예 14.1] << AddnumbersImpl.java >>

```java
@Addressing
@WebService
public class AddNumbersImpl {
    ...
}
```

위와 같이 `javax.xml.ws.soap.Addressing Annotation`을 통해 웹 서비스 Addressing을 설정하면 이를 통해 서비스 Endpoint는 다음과 같은 동작을 수행하게 된다.

- 받아들이는 메시지의 모든 웹 서비스 Addressing Header들을 해석해서 올바른 문법인지 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)를 직접 설정할 수 있다.

예 14.2] << AddnumbersImpl.java >>

```java
@Addressing
@WebService
public class AddNumbersImpl {
    @Action(
        input = "http://tmaxsoft.com/input",
        output = "http://tmaxsoft.com/output",
        fault = { ...
```
이와 같이 설정된 메소드는 WSDL로 변환될 때 다음과 같은 WSDL 문서의 operation element에 바인딩된다.

[예 14.3] << Addnumbers.wsdl >>

```xml
...<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>
...```

126 JEOS Web Service 안내서
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 javax.xml.ws.AddressingFeature(false));
```

위와 같이 클라이언트로부터 프록시(Endpoint Interface)를 얻을 때 javax.xml.ws.AddressingFeature의 값 을 false로 설정하면 클라이언트는 메시지를 전송할 때 서비스에서 WSDL에 규정하고 있는 웹 서비스 Addressing 기능을 동작하지 않은 상태로 메시지를 전송하게 된다.


```
<T> Dispatch<T> createDispatch(javax.xml.namespace.QName, java.lang.Class<T>, Service.Mode, WebServiceFeature...)  
Dispatch<java.lang.Object> createDispatch(javax.xml.namespace.QName, java.xml.bind.JAXBContext, Service.Mode, WebServiceFeature...)  
<T> T getPort(java.lang.Class<T>, WebServiceFeature...)
```
14.4. 예제
다음은 예제에서 다루는 서비스 Endpoint 구현 클래스의 예이다.

[예 14.4] << AddNumbersImpl.java >>

```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 7에 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 Webservices 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 2.2 - JEUS 7
<?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: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:ns2="http://server.wsaddressing/">
  <S:Body>
    <ns2:sum xmlns:ns2="http://server.wsaddressing/">
      <arg0>10</arg0><arg1>10</arg1>
    </ns2:sum>
  </S:Body>
</S:Envelope>
<S:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <wsa:To>http://www.w3.org/2005/08/addressing/anonymous</wsa:To>
    <wsa:MessageID>uuid:815cb296-a2f3-45df-80c5-5a2c1ca836ca</wsa:MessageID>
    <wsa:RelatesTo>uuid:880f3891-d07b-4ad1-bbcb1-8dce8f1aedef</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 2.2 - JEUS 7

<?xml version="1.0" ?>
<S:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <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:ss="http://server.wsaddressing/">
      <arg0>10</arg0><arg1>10</arg1>
    </ns2:addNumbers2>
  </S:Body>
</S:Envelope>
제15장 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 것이 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 끝김(connection drop)이 현상이 발생하는 경우
- 애플리케이션 메시지가 전송 도중 사라지는 경우
- 순서에 맞게 전송되는 애플리케이션 메시지를 요구하는 경우인 데 순서에 맞게 전송되지 않는 경우

신뢰성 메시징 기술을 사용할 것인지를 고려할 때 다음과 같은 장단점을 비교해봐야 한다.

- 신뢰성 메시징은 소스로부터 목적지까지 단 한 번만 전송되는 것을 의미한다. 만약 순서에 맞게 전송되는 응선을 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 응선일 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 도구를 사용하여 웹 서비스를 생성한다.

WSDL 파일에 WS-Reliable Messaging을 웹 서비스 정책을 설정하는 방법은 다음과 같다.

- 기본적으로 PolicyAssertion에 다음을 설정한다.

```xml
```

- 순서에 맞는 전송(inorder)을 보장받기 위해서는 PolicyAssertion에 다음을 설정한다.

```xml
```

다음은 이와 같이 구성한 WSDL 파일의 예이다.

[예 15.1] << AddNumbers.wsdl >>

```xml
<?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"
 <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>

<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">
        <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>
15.2.2. Java 클래스로부터 설정

Java 클래스로부터 WS-Reliable Messaging을 설정하기 위해서는 다음과 같이 wsgen 툴의 -policy 기능을 이용하여 wsit-endpoint.xml을 먼저 얻어야 한다.

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

다음은 service-config.xml의 내용이다.

[예 15.2] << service-config.xml >>

```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 7 웹 서비스와 동일하다.

다음은 service-config.xml 파일의 예이다.

[예 15.3] << service-config.xml >>

```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>

15.5. 예제 실행

위의 service-config.xml 파일을 가지고 wsgen을 통해 웹 서비스를 생성하기 위해서는 다음과 같이 입력한다.

```bash
$ 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 패키지에서 WEB-INF 폴더에 추가된 것이다.

서비스를 deploy하기 위해 다음과 같이 실행한다.

```bash
$ ant deploy
```

웹 서비스가 정상적으로 deploy되면 이를 사용하는 클라이언트를 다음과 같이 생성하고 서비스를 호출한다.

```bash
$ ant run
...
run:
  [java] ##########################################
  [java] ### JAX-WS Webservices examples - wsit ###
  [java] ##########################################
```
[java] Testing wsit webservices...
[java] Success!

... 

BUILD SUCCESSFUL
제16장 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 애플리케이션 에 참여하는 모든 참여인들에게서 동의한 결과를 얻을 수 있도록 하는 메커니즘이다.

전통적으로 트랜잭션은 일명 "ACID"라는 다음 속성을 포함하고 있다.
- 원자성(Atomicity) : 성공적이면 모든 작업이 발생하고 실패하면 어떤 작업도 발생하지 않는다.
- 일관성(Consistency) : 애플리케이션이 완료될 때 유 효한 상태 변화를 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 코딩이 필요하다. 또한 유연한 방식으로 다중 서비스에서 실행된 결과들의 프로세싱 코딩이 필요하며 이를 위해 자유로운 형식의 트랜잭션이 필요하다.

JEUS 7 웹 서비스에서 다음의 트랜잭션 스펙을 지원한다.
- WS-Cooordination
- WS-AtomicTransaction

위 트랜잭션 스펙은 협동하고 있는 작동에 대해 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 문서 에 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.
웹 서비스 트랜잭션 정책을 설정한 WSDL 파일은 다음과 같다.

[예 16.1] << AddNumbers.wsdl >>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="AddNumbersService" targetNamespace="http://server.fromwsdl/
 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">
   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>
</definitions>
```

140 JEUS Web Service 안내서
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.

[예 16.2] << AddnumbersImpl.java >>

```java
@WebService
@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 클라이언트가 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 annotation을 붙이는 것을 제외한 나머지 설정은 기본적인 JEUS 7 웹 서비스와 동일하다.

웹 서비스 클라이언트에서는 JTA(Java Transaction API)를 사용하여 프로그래밍한다.

[예 16.3] << AddNumbersClient.jsp >>

```java
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();
제17장 웹 서비스 보안

본 장에서는 전송 수준과 메시지 수준의 웹 서비스 보안에 대해 자세히 설명하고, JEUS 웹 서비스에서 이 류한 보안을 적용하는 방법에 대해 설명한다.

17.1. 개요

웹 서비스에 보안을 적용하기 위해서는 전통적으로 다음과 같은 2가지 방식이 존재한다.

- 전송 수준 보안(Transport-level Security)
  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 임히기 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 보안은 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 참조한다.

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. `wsimport`를 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 비밀성을 고려하여 안전한 웹 서비스를 구현할 때 사용되는 확장된 SOAPElement들을 정의하고 있다.

다음은 웹 서비스 보안 명세서를 사용한 웹 서비스 보안 시나리오이다.

- 사용자명 인증(Username Authentication)을 통한 대칭 비밀키의 인증 가능 강화
- 상호 인증 보안
• SSL을 통한 SAML 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 누구인지 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 이용하여 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 해독하고 서명을 확인한다.

• Keystore 설정

다음은 Keystore 설정 정보이다.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>클라이언트</td>
<td>서버의 공개 키가 담긴 Keystore이다.</td>
</tr>
<tr>
<td>서버</td>
<td>서버의 개인 키가 담긴 Keystore이다.</td>
</tr>
</tbody>
</table>

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 상대편이 누구인지 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 꺼낸 후 암호화된 메시지를 해석한다. 서버의 공개 키로 서명을 확인한다.

● Keystore 설정

다음은 Keystore 설정 정보이다.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>클라이언트</td>
<td>서버의 공개 키가 담긴 Keystore이다.</td>
</tr>
<tr>
<td>서버</td>
<td>서버의 개인 키가 담긴 Keystore이다.</td>
</tr>
</tbody>
</table>

17.3.2.3. SSL을 통한 SAML 인증

본 절에서는 SAML(Security Assertions Mark-up Language) 토큰을 SOAP 메시지 Header에 포함시켜 보내는 방법에 대해 설명한다.

실제 웹 서비스에서 SAML 토큰을 사용한 서비스를 하려면 이러한 SAML 토큰을 생성하는 SAML 토큰 프레임워크가 필요하다. 여기서는 단순히 헤더를 통해 SAML 토큰을 텍스트로 생성해서 보내는 예를 통해 어떻게 SAML 토큰을 SOAP 메시지 Header에 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 자세히 정의한다.
- 사용자의 신원을 Export시키지 않고 사용자 속성을 결정하는 기능을 비롯하여 여러 가지 개인 정보 보호 메커니즘을 지원한다.
- UNIX, Microsoft Windows, X.509, LDAP, DCE, XCML 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 Header에 포함하여 SSL 설정을 통해 메시지를 보낸다.
2. 서버는 SOAP 메시지 Header에 포함된 SAML 토큰을 해석하고 응답 메시지를 보낸다.

**참고**
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 생성하는 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 방식으로 정의되는 기본 메커니즘을 제공하고 더욱 더 효율적인 키나 새로운 키 재료와 함께 컨텍스트가 수립되도록 한다. 이러한 접근은 전체적인 성능과 다음에 따르는 메시지 교환을 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.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>클라이언트</td>
<td>클라이언트의 개인 키와 서버의 공개 키가 담긴 Keystore이다.</td>
</tr>
<tr>
<td>서버</td>
<td>서버의 개인 키와 클라이언트의 공개 키가 담긴 Keystore이다.</td>
</tr>
</tbody>
</table>

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 토클에 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 토큰 때문에 서로를 인증하지 못하는 상황은 클라이언트와 서버 사이에 각각의 인증 역할을 증가해 주는 또 다른 웹 서비스가 존재하여야 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.
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.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>클라이언트</td>
<td>1. 클라이언트의 개인 키와 서버의 공개 키가 담긴 Keystore이다.</td>
</tr>
<tr>
<td></td>
<td>2. 클라이언트의 개인 키와 STS 서버의 공개 키가 담긴 Keystore이다.</td>
</tr>
<tr>
<td>서버</td>
<td>서버의 개인 키와 클라이언트의 공개 키가 담긴 Keystore이다.</td>
</tr>
<tr>
<td>STS 서버</td>
<td>서버의 개인 키와 클라이언트의 공개 키가 담긴 Keystore이다.</td>
</tr>
</tbody>
</table>
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.

참고
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 클래스의 classes 디렉터리 하위의 META-INF 디렉터리에 Keystore 파일이 위치해야 한다.

[예 17.1] << jeus-webservices-config.xsd >>

```xml
<?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"
```
'include-token' 속성 설정

'include-token' 속성은 메시지에 토큰을 포함할 것인지를 여부를 설정한다.

X509 토큰에서 사용되는 예는 다음과 같다.

[예 17.2] << 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"
xmns: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>
```

참고

그 밖의 다른 element 설정에 대한 자세한 설명은 'jeus-webservices-config.xsd' 스키마 및 "JEUS Reference Book"이나 sample 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
```

'policy'의 인자로 사용되는 service-config.xml을 구성하려면 service-config.xml을 위한 JEUS 보안 정책 스키마인 jeus-webservices-config.xsd와 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.

[예 17.3] << jeus-webservices-config.xsd >>

```xml
<?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: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.

[예 17.4] << 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="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.

[예 17.5] << 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">
```
• '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.

[예 17.6] << jeus-webservices-config.xsd >>

• '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 사용하는 대칭 키(Symmetric Key)는 SOAP 메시지의 Header 부분에 추가적인 element(EncryptedKey) 하위에 암호화된 상태로 메시지와 함께 전송되는데, 이 대칭 키를 암호화하거나 복호화할 때 사용되는 토큰이 갈다는 것을 의미한다. 서명할 때 도 서명을 검증할 때와 같은 토큰을 통하여 이루어진다는 것을 의미한다. 하위의 'protection-token' element는 추가적으로 메시지를 암호화 및 서명에 사용되는 토큰이 동일함을 의미한다.

[예 17.7] << jeus-webservices-config.xsd >>

• '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 Header 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.

[예 17.8] << jeus-webservices-config.xsd >>
다음은 하위의 element에 대한 설명이다.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>initiator-token</td>
<td>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.</td>
</tr>
<tr>
<td>recipient-token</td>
<td>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.</td>
</tr>
</tbody>
</table>

– token

다음은 token element 보안 설정에 대한 예로 하위의 element 중 하나를 선택할 수 있다.

[예 17.9] << jeus-webservices-config.xsd >>

```xml
<?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:schema>
```

156  JEUS Web Service 안내서
다음은 하위의 element에 대한 설명이다.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>supporting-token</td>
<td>이곳에 설정한 토큰 값은 실제 메시지의 Header 부분에 포함되며 다른 메시지 부분을 암호화하거나 서명할 때 사용할 수 있다.</td>
</tr>
<tr>
<td>signed-supporting-token</td>
<td>'supporting-token'과 같다. 추가적으로 이 토큰 또한 서명이 된다.</td>
</tr>
<tr>
<td>endorsing-supporting-token</td>
<td>서명을 다시 서명하는 데 사용되는 토큰을 설정할 때 사용한다. 'Signature' element를 전체를 서명한다.</td>
</tr>
<tr>
<td>signed-endorsing-supporting-token</td>
<td>'endorsing-supporting-token'과 같다. 추가적으로 이 토큰 또한 원래 서명을 통해 서명한다. 상호 서명하는 것을 의미한다.</td>
</tr>
</tbody>
</table>

- protection

다음은 protection element 보안 설정에 대한 예로 하위의 element 중 하나를 선택할 수 있다.

[예 17.10] << jeus-webservices-config.xsd >>

```xml
<?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.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed-part</td>
<td>서명(검증)할 부분을 설정할 때 사용한다.</td>
</tr>
<tr>
<td>encrypted-part</td>
<td>암호화 혹은 복호화할 부분을 설정할 때 사용한다.</td>
</tr>
<tr>
<td>signed-element</td>
<td>암호화 혹은 복호화할 부분을 설정할 때 사용한다. 특정 element를 부분 서명 할 때 사용하면 유용하다. 'signed-element' element 설정하는 경우 'disable-streaming-security' element의 값을 'true'로 함께 설정해야 한다. 자세한 사항은 샘플을 참고한다.</td>
</tr>
<tr>
<td>encrypted-element</td>
<td>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.</td>
</tr>
</tbody>
</table>

- '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.

[예 17.11] << jeus-webservices-config.xsd >>

```xml
<?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.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>supporting-token</td>
<td>이 설정은 'endpoint-policy-subject' element의 'token' [156] element와 같다. 단, 이 설정은 이 메소드에만 설정되는 값이다.</td>
</tr>
<tr>
<td>protection</td>
<td>이 설정은 'endpoint-policy-subject' element의 'protection' [157] element와 같다. 단, 이 설정은 이 메소드에만 설정되는 값이다.</td>
</tr>
</tbody>
</table>

• '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.

[예 17.12] << jeus-webservices-config.xsd >>

```xml
<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.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>supporting-token</td>
<td>이 설정은 'endpoint-policy-subject' element의 'token' [156] element와 같다. 단, 이 설정은 이 메소드의 매개변수 및 반환값에만 설정되는 값이다.</td>
</tr>
<tr>
<td>protection</td>
<td>이 설정은 'endpoint-policy-subject' element의 'protection' [157] element와 같다. 단, 이 설정은 이 메소드의 매개변수 및 반환값에만 설정되는 값이다.</td>
</tr>
</tbody>
</table>
WSDL로부터 웹 서비스 구현

WSDL로부터 구현하기 위해서는 WSDL 문서에 직접 메시지 보안 정책 설정을 하고 wsimport 도구를 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.

참고


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.

[예 17.13] << jeus-webservices-config.xsd >>

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions...

targetNamespace="http://tmax.com/" name="NewWebServiceService">
    wsu:Id="NewWebServicePortBindingPolicy">
<ns1:ExactlyOne>
    <ns1:All>
            <ns1:Policy>
                <ns1:ExactlyOne>
                    <ns1:All>
                        <ns7:AlgorithmSuite>
                            <ns1:Policy>
                                <ns1:ExactlyOne>
                                    <ns1:All>
                                        <ns7:TripleDes />
                                    </ns1:All>
                                </ns1:Policy>
                            </ns7:AlgorithmSuite>
                        </ns1:ExactlyOne>
                    </ns1:All>
                </ns1:Policy>
            </ns7:IncludeTimestamp />
        </ns1:Policy>
    </ns1:All>
</ns1:ExactlyOne>
</ns1:Policy>
</ns7:AlgorithmSuite>
<ns7:IncludeTimestamp />
<ns7:Layout>
    <ns1:Policy>
        <ns1:ExactlyOne>
            <ns1:All>
                <ns7:Strict />
            </ns1:All>
        </ns1:Policy>
    </ns1:Policy>
</ns7:AlgorithmSuite>
```
<ns1:ExactlyOne>
</ns1:Policy>
</ns7:Layout>
<ns7:OnlySignEntireHeadersAndBody />
<ns7:ProtectionToken>
<ns1:Policy>
<ns1:ExactlyOne>
<ns1:All>
<ns7:X509Token xmlns:ns7="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
<ns1:Policy>
<ns1:ExactlyOne>
<ns1:All>
<ns8:WssX509V3Token xmlns:ns8="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"/>
</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:Wss11 xmlns: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 wss:visibility="private"
alias="xws-security-server"
storepass="changeit" type="JKS"
location="keystore.jks" />
<sc:TrustStore wss:visibility="private"
peeralias="xws-security-client"
storepass="changeit" type="JKS"
location="cacerts.jks" />
<sc:ValidatorConfiguration xmlns: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/wsdl"/>
</ns1:ExactlyOne>
</ns1:Policy>
wsu:Id="NewWebServicePortBinding_add_Input_Policy">
<ns10:ExactlyOne>
<ns10:All>
<ns11:Body/>
</ns11:EncryptedParts>
<ns12:Body/>
<ns12:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
<ns12:Header Namespace="http://www.w3.org/2005/08/addressing" Name="To"/>
<ns12:Header Namespace="http://www.w3.org/2005/08/addressing" Name="From"/>
<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>
<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>
제17장 웹 서비스 보안 163

</ns10:Policy>
</ns13:UsernameToken>
</ns10:All>
</ns10:ExactlyOne>
</ns10:Policy>
</ns13:SignedSupportingTokens>
</ns10:All>
</ns10:ExactlyOne>
</ns10:Policy>
<ns14:ExactlyOne>
<ns14:All>
<ns15:Body />
</ns15:EncryptedParts>
<ns16:Body />
<ns16:Header Name="ReplyTo"
Namespace="http://www.w3.org/2005/08/addressing"/>
<ns16:Header Namespace="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>
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.

[예 17.14] << jeus-webservices-config.xsd >>

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
... targetNamespace="http://server.fromjava/">
```
제17장 웹 서비스 보안 165

<wsu:Policy wsu:Id="TmaxBP0">
  <wsu:ExactlyOne>
    <wsu: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" />
    </wsu:All>
  </wsu:ExactlyOne>
</wsu:Policy>

<types>
  ...
</types>

<message name="addNumbers">
  ...
</message>

<portType name="AddNumbersImpl">
  ...
</portType>

(binding name="AddNumbersImplPortBinding" type="tns:AddNumbersImpl">
  <wsu: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>
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.

[예 17.15] <<service-config.xml>>

```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>
        <token>
          <signed-supporting-token>
            <username-token>
              <username-password-validator>
                fromjava.server.UsernamePasswordValidator
              </username-password-validator>
            </username-token>
          </signed-supporting-token>
        </token>
        <protection>
          <signed-part>...</signed-part>
          <encrypted-part>...</encrypted-part>
        </protection>
      </security-policy>
    </endpoint-policy-subject>
  </policy>
</web-services-config>
```

166 JEUS Web Service 안내서
사용자명의 유효자(Validator) 클래스 작성


[예 17.16] << UsernamePasswordValidator.java >>

```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.

[예 17.17] <<wsit-client.xml>>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
    ... targetNamespace="http://server.fromjava/">
```
사용자명의 핸들러 클래스 작성

javax.security.auth.callback.CallbackHandler를 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 reader 객체를 사용하여 사용자로부터 직접 입력받을 수도 있다.

[예 17.18] 「UsernamePasswordCallbackHandler.java」

```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");
            }
        }
    }
}
```
17.4.3. 상호 인증 보안(Mutual Certificates Security)

본 절에서는 서버와 클라이언트에서 상호 인증 보안을 설정하는 방법에 대해 설명한다.

17.4.3.1. 서버 설정

서버의 웹 서비스 설정은 service-config.xml 웹 서비스 설정 파일로 구성할 수 있다.

[예 17.19] << service-config.xml >>

```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>
      </security-policy>
    </endpoint-policy-subject>
  </policy>
  <keystore>...</keystore>
  <truststore>...</truststore>
</web-services-config>
```
17.4.3.2. 클라이언트 설정

클라이언트의 웹 서비스 설정은 추가적으로 wsit-client.xml에 Keystore(Truststore) 설정을 해야 한다.

[예 17.20] << wsit-client.xml >>

```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 웹 서비스 설정 파일로 구성할 수 있다.

[예 17.21] << service-config.xml >>

```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/>
            </signed-supporting-token>
          </supporting-token>
        </security-policy>
      </input-message-policy-subject>
    </operation-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.
SAML 메시지 처리를 위한 Callback 핸들러 클래스 작성

SAML 표준에 따라 SOAP 요청 메시지에 정보를 담기 위해 예제에서는 javax.security.auth.callback.Call backHandler를 구현하는 SamlCallbackHandler를 사용한다. SamlCallbackHandler에서는 Callback의 종 류에 따라 필요한 assertion을 저장하는 역할을 한다.
다음은 이 예제에서 사용되는 SamlCallbackHandler의 예이다.
17.4.5. 보안 대화(Secure Conversation)

본 절에서는 서버와 클라이언트에서 보안 대화를 설정하는 방법에 대해서 설명한다.

17.4.5.1. 서버 설정

웹 서비스의 서버 설정은 service-config.xml 웹 서비스 설정 파일로 구성할 수 있다.
<?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>
                  <include-token>true</include-token>
                </asymmetric-binding-recipient-token>
                <include-token>true</include-token>
              </secure-conversation-token>
            </protection-token>
            <encrypt-signature>true</encrypt-signature>
          </symmetric-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>
      </security-policy>
    </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.

[예 17.25] <<wsit-client.xml>>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/

xmlns:soap12="http://schemas.xmlsoap.org/wsd1/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 wsp:visibility="private"
                location="/keystore.jks" type="JKS"
                alias="xws-security-client" storepass="changeit">
            </sc:KeyStore>
        </wsp:All>
    </wsp:ExactlyOne>
</wsp:Policy>
```
17.4.6. 웹 서비스 신뢰(WS-Trust)

본 절에서는 웹 서비스 신뢰를 위한 서버와 STS, 클라이언트의 설정 방법에 대해서 설명한다.

17.4.6.1. 서버 설정

웹 서비스의 서버 설정은 service-config.xml 웹 서비스 설정 파일로 구성할 수 있다.

[예 17.26] <<service-config.xml>>

```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>
          <symmetric-binding>
            <protection-token>
              <x509-token />
            </protection-token>
            <layout>Strict</layout>
          </symmetric-binding>
        </security-binding>
      </security-policy>
    </endpoint-policy-subject>
  </policy>
</web-services-config>
```
웹 서비스는 추가적으로 다음 주소의 STS로부터 보안 토큰을 받을 것을 명시한다

```xml
<token>
  <endorsing-supporting-token>
    <issued-token>
      <issuer-address>
        http://localhost:8088/trust_sts/SecurityTokenService
      </issuer-address>
    </issued-token>
  </endorsing-supporting-token>
</token>
```

17.4.6.2. STS 설정


[예 17.27] WS-SecurityPolicy 설정 : <<sts.wsdl>>

```xml
<?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>
                  <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>
</wsdl:definitions>
```
<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:STSConfiguration>

http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
  <tc:TokenType></tc:TokenType>
</tc:ServiceProvider>
</tc:ServiceProviders>
</tc:STSConfiguration>
<wsap10:UsingAddressing />
</wsp:ExactlyOne>
</wsp:Policy>

...<wssl:binding name="ISecurityTokenService_Binding"
    type="tns:ISecurityTokenService">
  <wsp:PolicyReference URI="#TmaxSTSServerPolicy" />
  ...</wssl:binding>
...</wssl:definitions>
클라이언트 설정

클라이언트의 웹 서비스 설정은 wsit-client.xml에 서버, STS 각각을 위한 Keystore(Truststore) 설정을 해야 한다.

[예 17.28] <<wsit-client.xml>>

```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 ... />
                <sc:SCClientConfiguration wspp:visibility="private">
                    <scc:LifeTime>36000</scc:LifeTime>
                </sc:SCClientConfiguration>
            </wsp:All>
        </wsp:ExactlyOne>
    </wsp:Policy>
    ...
    <wsp:PolicyReference URI="#TmaxFSClientPolicy" />
    ...
</wsdl:definitions>

<!-- STSService WSDL -->
<wsp:Policy wsu:Id="TmaxSTSClientPolicy" ...>
    <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>
```

180  JEUS Web Service 안내서
17.4.7. 실행

다음은 실행 화면이다.

<<__Exception__>>
com.sun.xml.ws.server.UnsupportedMediaTypeException: 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.HttpServletDelegate.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 javax.servlet.http.HttpServlet.service(HttpServlet.java:725)
   at javax.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)


```java
at jeus.servlet.engine.HttpRequestProcessor.run(HttpRequestProcessor.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 javax.servlet.http.HttpServlet.service(HttpServlet.java:725)
at javax.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.HttpRequestProcessor.run(HttpRequestProcessor.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 설정 또한 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 수준
의 보안을 간단하게 처리할 수 있다. 이는 JAX-RPC(JEUS 5) 웹 서비스의 메시지 수준 보안 방식과 유사하다.

17.5.1. 암호화(Encryption)

웹 서비스 메시지 수준 보안에서 암호화에 대한 설정은 다음과 같이 구분된다.

- JAX-RPC(JEUS 5) 웹 서비스 설정
- JAX-WS(JEUS 7) 웹 서비스 설정

JAX-RPC(JEUS 5) 웹 서비스 설정

JAX-RPC(JEUS 5) 웹 서비스 메시지 수준 보안에서 암호화를 설정하는 방식은 다음과 같이 jeus-webservices-dd.xml(서버)와 jeus-web-dd.xml(클라이언트)를 설정한다.

[예 17.29] << jeus-webservices-dd.xml >>

```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>
        </request-receiver>
        <response-sender>
          <action-list>Encrypt</action-list>
          <password-callback-class>ping.PingPWCallback</password-callback-class>
        </response-sender>
        <decryption>
          <keystore>
            <key-type>jks</key-type>
            <keystore-password>changeit</keystore-password>
            <keystore-filename>server-keystore.jks</keystore-filename>
          </keystore>
        </decryption>
      </security>
    </port>
  </service>
</jeus-webservices-dd>
```
JAX-WS(JEUS 7) 웹 서비스 설정

JAX-WS(JEUS 7) 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.

[예 17.30] << service-config.xml >>

```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>
          </keystore-file>
        </keystore>
      </security-policy>
    </endpoint-policy-subject>
  </policy>
</web-services-config>
```
<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(Signature)

웹 서비스 메시지 수준 보안에서 서명에 대한 설정은 다음과 같이 구분된다.

- JAX-RPC(JEUS 5) 웹 서비스 설정
- JAX-WS(JEUS 7) 웹 서비스 설정

JAX-RPC 웹 서비스 설정

JAX-RPC(JEUS 5) 웹 서비스 메시지 수준 보안에서는 다음과 같이 jeus-webservices-dd.xml(서버) 또는 jeus-web-dd.xml(클라이언트)을 통해 서명을 설정한다.

[예 17.31] << jeus-webservices-dd.xml >>

```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>
          <signature-infos>
            <signature-info>
              <keyIdentifier>DirectReference</keyIdentifier>
            </signature-info>
          </signature-infos>
        </response-sender>
      </security>
    </port>
  </service>
</jeus-webservices-dd>
```
JAX-WS 웹 서비스 설정

JAX-WS(JEUS 7) 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.

[예 17.32] << service-config.xml >>

```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>
      </security-policy>
    </endpoint-policy-subject>
  </policy>
</web-services-config>
```
17.5.3. Timestamp

웹 서비스 메시지 수준 보안에서 Timestamp의 설정은 다음과 같이 구분된다.

- JAX-RPC(JEUS 5) 웹 서비스 설정
- JAX-WS(JEUS 7) 웹 서비스 설정
JAX-RPC(JEUS 5) 웹 서비스 설정

JAX-RPC(JEUS 5) 웹 서비스 메시지 수준 보안에서는 다음과 같이 jeus-webservices-dd.xml(서버) 또는 jeus-web-dd.xml(클라이언트)을 통해 Timestamp를 설정한다.

예 17.33] << jeus-webservices-dd.xml >>

```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 Timestamp</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(JEUS 7) 웹 서비스 설정

JAX-WS(JEUS 7) 웹 서비스의 메시지 수준 보안은 웹 서비스를 생성할 때, 즉 wsgen 혹은 wsimport 통해 웹 서비스를 생성할 때 service-config.xml 파일을 추가적으로-policy 인자와 함께 명령문에 설정한다.

[예 17.34] << service-config.xml >>

```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>
    </policy>
    <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-file>
      </truststore>
    </security-policy>
  </policy>
</web-services-config>
```
17.5.4. 사용자명 토큰(Username Token)

웹 서비스 메시지 수준 보안에서 사용자명 토큰(Username Token)의 설정은 다음과 같이 구분된다.

- JAX-RPC(JEUS 5) 웹 서비스 설정
- JAX-WS(JEUS 7) 웹 서비스 설정

**JAX-RPC(JEUS 5) 웹 서비스 설정**

JAX-RPC(JEUS 5) 웹 서비스 메시지 수준 보안에서는 다음과 같이 jeus-webservices-dd.xml(서버) 혹은 jeus-web-dd.xml(클라이언트)을 통해 Timestamp을 설정한다.
예 17.35] << jeus-webservices-dd.xml >>

```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 UsernameToken
          </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 UsernameToken
          </action-list>
          <user>xws-security-server</user>
          <password-callback-class>
            ping.PingPWCallback
          </password-callback-class>
          <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-info>
            </signature-info>
          </signature-infos>
        </response-sender>
      </security>
    </port>
  </service>
</jeus-webservices-dd>
```
JAX-WS(JEUS 7) 웹 서비스 설정

JAX-WS(JEUS 7) 웹 서비스의 메시지 수준 보안은 웹 서비스를 생성(\texttt{wsgen} 또는 \texttt{wsimport} 통해 웹 서비스를 생성)할 때 \texttt{service-config.xml} 파일을 추가적으로 \texttt{-policy} 인자와 함께 명령문에 설정한다.

[예 17.36] << service-config.xml >>

```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>
        </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>
      </security-policy>
    </endpoint-policy-subject>
  </policy>
  <keystore>
    <keystore-file>
      <alias>xws-security-server</alias>
      <key-type>JKS</key-type>
      <keystore-password>changeit</keystore-password>
    </keystore-file>
  </keystore>
  <wss-version>11</wss-version>
  <protection>
```
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 “26.4. 접근 제어 설정”과 동일하며 클라이언트의 호출 방식만 다르다.

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 Artifact의 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.

```bash
$ 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.

[예 17.37] << authorization.txt >>

```plaintext
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.

- javax.xml.ws.BindingProvider.USERNAME_PROPERTY
다음 예는 실제로 프로그램에서 위 설정을 구현한 예이다.

```java
Echo port = // ... SEI(Service Endpoint Interface)의 획득
((javax.xml.ws.BindingProvider) port).getRequestContext().
    put(javax.xml.ws.BindingProvider.USERNAME_PROPERTY, "jeus");
((javax.xml.ws.BindingProvider) port).getRequestContext().
    put(javax.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 접근하려는 때, 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.

**[예 17.38] << AddNumberClient.java >>**

```java
public class AddNumbersClient {
    ...
    public void execute() {
        Authenticator.setDefault(new MyAuthenticator());
        AddNumbersImpl port = new AddNumbersImplService().getAddNumbersImplPort();
        ((javax.xml.ws.BindingProvider) port).getRequestContext().
            put(javax.xml.ws.BindingProvider.USERNAME_PROPERTY, "jeus");
        ((javax.xml.ws.BindingProvider) port).getRequestContext().
            put(javax.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());
        }
    }
}
```
본 장에서는 UDDI에 대한 기본 개념과 JEUS에서의 UDDI 사용 방법에 대해서 설명한다. 또한, UDDI 레지스트리를 프로그램적으로 접근할 수 있는 UDDI 클라이언트를 설명하며, UDDI 레지스트리에 쉽게 접근하기 위한 웹 기반 사용자 인터페이스에 대해서도 설명한다.

18.1. 개요

UDDI(Universal Description, Discovery and Integration) 스펙은 웹 서비스에 대한 정보의 공개 및 검색에 대한 방법을 정의한다. UDDI는 XML(Extensible Markup Language)과 SOAP(Simple Object Access Protocol)과 같은 기존 표준에 기반하고 있다.

UDDI는 일반적인 XML 형태로 구현된 비즈니스들과 그것들의 서비스 기술(description)에 대한 분산 레지스트리에 기반한 접근법을 취한다. UDDI 프로젝트의 핵심 콤포넌트는 UDDI 비즈니스의 등록과 비즈니스 Entity를 기술한 XML 파일과 그 웹 서비스이다.

UDDI는 프로그래밍 모델과 스키마를 제공하고 이것은 레지스트리를 사용하여 구축을 정의한다. UDDI 스펙의 모든 API는 XML로 정의되어 있고, SOAP envelope으로 래핑(wrapping)되어 있으며, HTTP를 통해 전송된다.

18.1.1. UDDI 이용

UDDI는 웹에서 비즈니스 레지스트리의 공유 오퍼레이션을 포함하고 있다. 대부분의 경우 프로그램과 프로그래머는 서비스의 정보를 위치시킬 때, 특히 프로그래머의 경우에는 알려진 웹 서비스와의 호환성있는 시스템을 준비하거나, 자신의 웹 서비스를 다른 사람이 호출할 수 있게 기술하는데 UDDI 비즈니스 레지스트리를 사용한다. 이것은 UDDI 비즈니스 레지스트리에 지정된 정보를 사용한다.

UDDI 비즈니스 레지스트리는 비즈니스 레벨에서 사용되며 주어진 파트너가 특정 웹 서비스 인터페이스 를 가지고 있는지 확인하거나, 해당 서비스를 가지고 해당 산업에 있는 회사를 찾아서 서비스에 대한 기술 정보를 얻기 위해 어떻게 파트너가 웹 서비스를 Export시켰는지에 대한 정보를 위치시킨다.

UDDI Data 구조

UDDI 레지스트리에서 사용하는 핵심 정보 모델은 XML 스키마로 정의된다.

UDDI XML 스키마는 4개의 핵심 타입의 정보로 구성되어 있으며 기술자들이 파트너의 웹 서비스들을 사용하기 위해 알아야 할 정보를 제공한다. 이 4가지는 서비스의 스펙에 대한 비즈니스 정보(<businessEntity>), 서비스 정보(<businessService>), 바인딩 정보(<bindingTemplate>) 그리고 기술 모델 정보(<tModel>) 이다.
웹 서비스에 대한 정보를 기술하거나 검색할 때 사용되는 정보 계층 구조와 XML 태그를 아래 그림에서 보여주고 있다.

[그림 18.1] 정보 계층과 주요 XML 태그 이름

- 비즈니스 Entity (businessEntity)
  웹 서비스를 제공하는 비즈니스 혹은 기관을 기술한다. 이 Entity는 비즈니스의 기본 정보(이름, 수행하고 있는 비즈니스 설명, 연락처)를 나타낸다. Dun & Bradstreet D-U-N-S® Number 같은 비즈니스용 식별자들도 포함되어 있다.

- 비즈니스 서비스 (businessService)
  비즈니스 Entity에 의해 기술된 기관이 제공하는 관련 웹 서비스의 그룹을 기술한다. 이것은 하나의 비즈니스 Entity의 논리적 자식이다.

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

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

JEUS의 UDDI는 UDDI 레지스트리 서버와 UDDI 클라이언트 라이브러리, 웹 기반 사용자 인터페이스로 구성되어 있다. JEUS의 UDDI는 UDDI 버전 2.0과 버전 3.0의 데이터 구조와 프로그래밍 API를 지원한다.

본 절에서는 JEUS에 UDDI 모듈을 deploy하는 방법과 JEUS UDDI 서버 가동에 대해 설명한다.

- JEUS UDDI DataStore를 생성하는 방법
- JEUS UDDI 서버를 deploy하는 방법
- JEUS UDDI 서버를 구성하고 설정하는 방법

JEUS UDDI 패키지의 핵심 라이브러리는 JEUS_HOME/lib/system 디렉터리에 위치하며 웹 인터페이스를 포함한 웹 컨텍스트에 관련된 모듈은 EAR 포맷으로 제공된다.

18.2.1. UDDI DataStore 생성

JEUS의 UDDI 서버는 데이터 영속성을 위해서 관계형 DB가 필요하다. JEUS UDDI 서버는 ANSI 표준 SQL을 지원하는 어떤 관계형 DB라도 사용할 수 있다.

JEUS UDDI DataStore는 JDBC를 사용하여 설정 과정은 간단하다.

첫 번째로, JEUS UDDI 서버 패키지와 함께 제공되는 DB 스크립트를 사용하여 새로운 UDDI DB를 생성한다 (DB 스크립트는 JEUS_HOME/lib/systemapps/uddi/dbscripts 디렉터리에 위치한다). DataStore의 구성 설정을 하기 위해서는 JEUS에서 'uddiDB'라는 데이터소스를 설정해야 한다.

JEUS_HOME/lib/systemapps/uddi/conf 디렉터리의 예제 파일을 참고한다.

18.2.2. UDDI 서버 Deploying

UDDI DB 구성 설정이 완료되었다면 JEUS UDDI 서버를 deploy해야 한다.

JEUS UDDI 서버 EAR 패키지는 JEUS가 설치될 때 JEUS_HOME/lib/systemapps/uddi 디렉터리에 생성된다 (파일 이름은 jeusuddi_v2c.ear과 jeusuddi_v3c.ear이다).

JEUS UDDI 서버를 deploy하는 것은 다른 EAR 애플리케이션을 deploy하는 과정과 같다. 예제에서는 컨텍스트 이름이 'uddi'로 UDDI가 deploy되는 것을 가정한다.

[예 18.1] <<jeus-web-dd.xml>>

```xml
<jeus-web-dd>
  <res-ref>
    <jndi-info>
      <ref-name>jdbc/uddiDB</ref-name>
      <export-name>UDDIDB</export-name>
    </jndi-info>
  </res-ref>
</jeus-web-dd>
```
JNDI 데이터소스 이름인 'jdbc/uddiDB'는 기본값으로 사용되는 JNDI 데이터소스 이름이다. 서버에 설정된 실제 데이터소스와 연결하는 <export-name>은 'UDDIDB'이다. 다른 JNDI 데이터소스 이름을 사용한다면 이 구성 설정 파일들을 수정한다.

18.2.3. UDDI 서버 구성 설정
JEUS UDDI 서버를 구성하기 위해서는 있는 설정 파일 uddi.properties를 수정해야 한다. 파일은 JEUS_HOME/lib/application 디렉터리에 위치한다. 만약 이 설정 파일 존재하지 않으면 deploy할 때 기본 설정 파일을 자동으로 생성한다.

다음은 uddi.properties 파일의 기본 설정에 대한 설명이다.

<표 18.1] uddi.properties

<table>
<thead>
<tr>
<th>UDDI Property Key</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>uddi.operatorName</td>
<td>UDDI 오퍼레이터 이름이다.</td>
</tr>
<tr>
<td>uddi.operatorURL</td>
<td>UDDI 오퍼레이터 사이트 URL이다.</td>
</tr>
<tr>
<td>uddi.auth</td>
<td>현재 사용 중인 UDDI 인증 모듈이다.</td>
</tr>
<tr>
<td></td>
<td>(기본값 : 'jeus.webservices.uddi.auth.DefaultAuthenticator')</td>
</tr>
<tr>
<td>uddi.dataSource</td>
<td>현재 사용 중인 UDDI DataStore 모듈이다.</td>
</tr>
<tr>
<td></td>
<td>(기본값 : 'java:comp/env/jdbc/uddiDB')</td>
</tr>
</tbody>
</table>

18.2.4. 새로운 사용자 추가
UDDI 레지스트리에 UDDI 데이터를 공개하기 위해서는 UDDI 레지스트리의 인증을 얻어야 한다.
JEUS UDDI 레지스트리의 Publisher의 인증은 2단계 과정을 거친다.

1. 사용자 ID/password를 통해 사용자를 인증
JEUS UDDI 레지스트리는 기본 인증 모듈인 'jeus.webservices.uddi.auth.DefaultAuthenticator'를 가진다. 기본적인 인증 모듈은 인증 시도에 대해 간단하게 승인한다. 이것은 단순히 UDDI 사용자의 userid를 인증할 뿐이다.
JEUS UDDI 레지스트리는 추가적인 인증 모듈인 'jeus.webservices.uddi.auth.XMLDocAuthenticator'를 제공한다. 이것은 XML 문서를 바탕으로 하며, 'JEUS_HOME/lib/application'에 있는 'uddi-user.xml'을 추가하거나 수정해야 한다.

다음은 예제 XML 문서이다.

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

g개발자는 Custom 인증 모듈을 개발할 수 있다. 전형적으로 UDDI 레지스트리는 외부의 인증 메커니즘을 사용할 수 있음을 의미한다. 추가적인 JEUS UDDI 인증 모듈은 JEUS UDDI 사용자가 특정 환경에서 어떻게 인증될지 결정함에 따라서 JEUS UDDI 사용자에 의해 개발될 수 있다.

d다음 코드는 Custom JEUS UDDI 인증 모듈을 개발하는 방법에 대한 예제이다.

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

public class SampleAuthenticator implements BaseAuthenticator {

  public SampleAuthenticator() {
  }

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

2. 인증된 사용자를 UDDI 레지스트리 사용자로 등록

d등록하는 방법에는 UDDI 웹 사용자 인터페이스를 이용하는 방법과 DataStore에서 직접 등록하는 방법이 있다. 본 절에서는 DataStore에서 직접 등록하는 방법만을 설명하고 UDDI 웹 인터페이스를 사용하는 방법은 “18.3. JEUS 서버에서의 UDDI Explorer 사용”을 참고한다.

UDDI 레지스트리의 사용자인 Publisher가 UDDI DataStore에 정의되었는지를 확인한다. Publisher는 UDDI DB의 PUBLISHER 테이블에 Publisher를 식별하는 Row가 있을 때 정의된다. SQL을 사용하여 사용자를 정의할 수 있다.

다음은 'jeus'라는 새로운 publisher를 정의하는 예이다.

```sql
INSERT INTO PUBLISHER (PUBLISHER_ID,PUBLISHER_NAME,ADMIN)
VALUES ('jeus','JEUS','false');
```
[표 18.2] PUBLISHER 테이블의 컬럼

<table>
<thead>
<tr>
<th>컬럼명</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUBLISHER_ID</td>
<td>Publisher의 user ID는 인증할 때 사용된다. 외부의 인증 서비스를 통하여 인증할 때 사용되는 값과 동일해야 하다.</td>
</tr>
<tr>
<td>PUBLISHER_NAME</td>
<td>Publisher의 이름(또는 UDDI 안에 검증된 이름)이다.</td>
</tr>
<tr>
<td>ADMIN</td>
<td>Publisher가 관리적인 권한을 가졌는지 여부를 나타낸다. 이 컬럼의 값은 true 또는 false이다.</td>
</tr>
</tbody>
</table>

18.2.5. UDDI 서버 실행

JEUS 서버를 가동시킨다. UDDI 서버의 URL은 다음과 같은 형태이다.

http://hostname:port/context/

기본적으로 JEUS UDDI 서버의 context path는 '/uddi'이다. 여기서 JEUS HTTP Listener의 기본 포트 8088 을 사용한다고 가정한다면 URL은 다음과 같다.

http://localhost:8088/uddi/

웹 브라우저에서 이 URL을 호출하였다면 다음과 같은 화면으로 이동한다.

[그림 18.2] JEUS UDDI 레지스트리 초기 화면
18.3. JEUS 서버에서의 UDDI Explorer 사용

JEUS UDDI Explorer는 UDDI 레지스트리의 UDDI 데이터의 등록, 수정, 삭제 등의 작업을 하기 위한 웹 애플리케이션이다.

JEUS UDDI Explorer는 다음과 같은 기능들을 제공한다.

- Private UDDI 레지스트리에 대한 Querying
- Private UDDI 레지스트리에 대한 Publish
- JEUS UDDI Explorer 설정

본 절에서는 JEUS UDDI Explorer 웹 사용자 인터페이스의 사용 방법에 대해서 설명한다.

18.3.1. UDDI 레지스트리 Querying


'UDDI Registry'를 사용해서 UDDI 레지스트리 오퍼레이터를 선택할 수 있고, 'Search For a'에서 Entity 타입을 선택할 수 있다. UDDI Registry와 Entity 타입(Business, Service, Technical Model)을 선택한 후 아래의 텍스트 필드에 찾을 이름의 일부 또는 이름 전체를 입력한다. 와일드 카드로 '%' 심볼을 이용할 수 있다. 'exact name' 혹은 'case sensitive'를 사용해서 결과를 필터링할 수 있다.

[그림 18.3] UDDI Search 입력 폼 화면
찾을 이름을 입력 후 [submit] 버튼을 클릭하면 검색을 시작한다. 선택된 UDDI 레지스트리는 사용자가 입력한 이름으로 시작하는 Businesses, Services, Technical Model들의 목록을 조회한다.

JEUS UDDI Explorer는 찾은 Entity에 대한 상세 내역을 다음과 같이 출력한다.

[그림 18.4] UDDI 레지스트리에서 찾은 결과 화면

18.3.2. UDDI 레지스트리에 대한 Publish

UDDI 레지스트리에 Publish하기 위해서는 사용자 ID와 패스워드를 사용하여 UDDI 레지스트리에 로그인 해야 한다. 사용자가 접속한 후 JEUS UDDI Explorer는 사용자가 등록한 비즈니스들의 목록과 Technical Model들을 조회한다. 등록한 비즈니스 또는 Technical Model은 왼쪽 트리의 '+'를 클릭하면 등록된 서비스들 및 기타 정보들을 간략하게 확인할 수 있고, 다시 오른쪽 화면에서는 이에 대한 상세한 정보 열람 및 수정이 가능하다.

비즈니스 추가

다음은 비즈니스를 추가하는 과정에 대한 설명이다.


   'Name' 항목에 'JEUS Web Services'로 입력하고 [Update] 버튼을 클릭한다.

2. Provider 화면의 [Detail], [Contact], [Identifier], [Category], [Discovery URL] 탭에서는 비즈니스 설명이나 사용자의 연락처, Discovery URL, 카테고리 그리고 식별자 같은 부가 정보를 추가할 수 있다.

   'Description' 항목을 'eBiz using Webservices'로 입력한다.
서비스 추가

다음은 서비스를 추가하는 과정에 대한 설명이다.


2. Service 화면은 Provider 화면과 같이 각각의 탭들을 통해 여러 가지 정보들을 수정하거나 조회할 수 있다.

[그림 18.7] 서비스 추가 화면
tModel 추가

다음은 tModel을 추가하는 과정에 대한 설명이다.


2. TModel 화면에서는 부가적인 정보인 테크니컬 모델에 대한 설명, Overview URL, Category 그리고 Identifier를 추가할 수 있다. 'Name' 항목은 'tModel for eBiz'로 입력한다.


[그림 18.8] tModel 저장 결과 화면

서비스 바인딩(bindingTemplate) 추가

다음은 서비스 바인딩(bindingTemplate)을 추가하는 과정에 대한 설명이다.

1. 화면 왼쪽의 트리뷰에서 비즈니스와 관련되는 [+]를 클릭한다. 비즈니스에 연관된 서비스를 클릭하면 오른쪽 화면에 서비스에 대한 자세한 내용이 조회된다.


4. [Instance Info] 탭으로 이동하여 [Add Instance Info] 버튼을 클릭한다. [Search] 버튼을 클릭하면 사용자가 공개된 tModel들을 찾을 수 있는 화면을 나타낸다. 찾을 tModel의 전체 이름이나 처음 몇 자리를 'Search for tModel names containing'에 입력하여 찾을 수 있다. 해당 항목에 'tModel'로 입력한다.

5. 화면 왼쪽의 트리뷰에서 'tModel for eBiz'을 선택한 후 [Add Instance Info] 버튼을 클릭하면 테크니컬 모델을 바인딩한 서비스가 저장된다.
JEUS UDDI Explorer에서 JEUS UDDI 레지스트리를 사용하기 위한 사용자를 등록할 수 있다.

우선, 관리자 권한을 가진 사용자로 로그인을 해야 한다. 로그인한 후에 메뉴에서 User 링크를 선택하면 다음과 같은 화면이 나타난다. 이 화면에서 관리자는 새로운 사용자를 추가, 수정, 삭제할 수 있다.

[Add User] 버튼을 클릭하면 다음과 같은 신규 사용자 등록 화면이 나타난다.
화면에서 'Username'과 'Fullname'은 필수 입력 사항이다. 'Username'은 JEUS UDDI Explorer에 로그인과 UDDI Data를 공개하기 위한 사용자 ID이다. 입력값을 입력한 후에 [Add]를 클릭하면 사용자가 등록된다.

[그림 18.12] JEUS UDDI Explorer 사용자 등록 화면
18.4. UDDI 클라이언트 생성

JEUS UDDI 클라이언트 라이브러리의 UDDI 레지스트리에 대한 Java 인터페이스를 제공한다. 이를 통해 구축한 웹 서비스들을 공개할 수 있고, 필요한 웹 서비스를 검색할 수 있다. JEUS UDDI 클라이언트 라이 브러리는 일종의 Java 클래스 라이브러리이며, UDDI 레지스트리와 상호 작용하는 API를 제공한다.

본 절에서는 UDDI 클라이언트 라이브러리를 사용해서 UDDI 클라이언트를 작성, 컴파일, 실행하는 방법을 설명한다. 제시된 예제는 UDDI v2.0 기반의 UDDI 클라이언트이며, UDDI v3.0 기반의 UDDI 클라이언트도 유사한 방법으로 사용할 수 있다.

18.4.1. UDDI 클라이언트 작성

다음은 UDDI 클라이언트를 작성하는 과정에 대한 설명이다.

- UDDI 클라이언트 클래스

UDDI 클라이언트 애플리케이션의 핵심 클래스이다. 이것은 UDDI 레지스트리에 연결하고 질의를 수행 하며 결과를 처리하는 모든 메소드를 가지고 있다. 다음의 예제 코드는 어떻게 객체를 생성하고 UDDI 레지스트리를 참조하는지 보여준다. 여기에 제시된 예제 애플리케이션에서 'UDDIClient'는 UDDI 레지스트리와 상호 작용을 하기 위해 사용하는 것이다.

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

- UDDI 레지스트리에서 비즈니스 질의

```java
Vector inNames = new Vector();
inNames.add(new Name("test_Biz"));
BusinessList list = client.find_business(null, inNames, null, null, null, null, 0);
```

find_business 메소드는 다수의 매개변수를 갖는다. 2번째 매개변수는 검색어이고, 7번째 매개변수는 조건에 부합하는 레퍼의 최대 개수이다(0은 조건에 부합하는 모든 값을 리턴함을 의미한다).

보다 상세한 검색을 위해서 식별자(identifiers), 분류자(categories), URL, 기술 모델(tModel) 등을 검색 조건에 추가할 수 있다. 위 방법을 통해 조건에 부합하는 비즈니스 리스트를 조회할 수 있다.

다음은 리스트에 포함된 비즈니스들의 이름을 출력하는 예제 코드이다.

```java
BusinessInfos infos = list.getBusinessInfos();
Vector businesses = infos.getBusinessInfoVector();
if (businesses != null) {
    for (int i = 0; i < businesses.size(); i++) {
        BusinessInfo info = (BusinessInfo)businesses.elementAt(i);
        Vector outNames = info.getNameVector();
        for (int j = 0; j < outNames.size(); j++) {
            System.out.println(outNames.elementAt(j));
        }
    }
}
```
UDDI 레지스트리에서 비즈니스 공개

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

```java
AuthToken authToken = client.get_authToken("userID", "password");
```

다음 단계는 새로운 비즈니스 Entity를 생성하고 이것을 인증을 얻은 UDDI 레지스트리에 공개하는 것이다. 예제에서는 단순하게 비즈니스 이름만을 정의하였다.

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

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

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

UDDI 레지스트리에서 비즈니스 삭제

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

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

UDDI 클라이언트 코드를 컴파일하려면 클라이언트 소스 코드가 위치한 디렉터리로 이동한다. 예제에서는 JEUS_HOME/samples/webservice/uddi/client 디렉터리에서 다음의 명령을 실행한다.

JEUS_HOME/samples/webservice/uddi/client$ ant build

18.4.3. UDDI 클라이언트 실행

UDDI 클라이언트 코드를 실행하려면 클라이언트 소스 코드가 위치한 디렉터리로 이동한다. 예제에서는 JEUS_HOME/samples/webservice/uddi/client 디렉터리에서 다음의 명령어를 실행한다.

JEUS_HOME/samples/webservice/uddi/client$ ant

실행 결과는 다음과 유사할 것이다.

##### Running FindBusinessSample #####
Found Business name: test_Biz
##### Done #####

18.5. XML 디지털 서명 사용법

본 절에서는 XML 디지털 서명의 개념과 생성 및 검증 방법에 대해 설명한다.

18.5.1. 디지털 서명(Digital Signature)

UDDI v3.0 사양(specification)에서의 중요한 변화 중 하나는 디지털 서명의 지원이다. UDDI Entity가 디지털 서명되면 데이터 무결성(integrity)과 확실성(authenticity)이 UDDI에 의해 전달된다. UDDI 레지스트리에 대한 요청자(inquirer)는 서명된 데이터만을 요청하도록 절의(Query)를 조정할 수 있고, 요청자가 UDDI 레지스트리로부터 가져온 데이터를 검증할 때 데이터가 정확하게 공개자(Publisher)가 의도한 것인지지를 확인할 수 있게 된다.

UDDI 레지스트리에 대한 공개자는 UDDI Entity를 소유하였다고 주장하는 누군가에 의해 잘못 전달되지 않을 것을 확인할 수 있게 되고, 공개자가 서명된 데이터를 가지게 되면, 공개자는 그 데이터의 무결성에 대해 확인할 수 있게 된다.

디지털 서명은 UDDI의 데이터 품질을 향상시키고, 높은 신뢰성을 필요로 하는 웹 서비스 환경에서 요구하는 데이터 보호와 부인 방지를 가능하게 한다.

UDDI v3.0 사양에서는 XML 서명 구문과 처리 방법(http://www.w3.org/TR/xmldsig-core/)을 사용하여 다음의 상위 수준 UDDI Entity에 대한 XML 디지털 서명을 지원한다.

- 비즈니스 Entity(businessEntity)
- 비즈니스 서비스(businessService)
- 서비스 바인딩(bindingTemplate)
기술 모델(tModel)

publisherAssertion

UDDI v3.0 스펙은 XML 서명과 검증을 클라이언트의 역할로 정의하고 있다. 그에 따라 JEUS 환경에서도 클라이언트의 XML 서명과 검증을 지원하고 있으며, 본 안내서에서 기술하고 있는 UDDI 클라이언트는 JEUS UDDI 클라이언트를 사용하고 있음을 가정한다.

참고

XML 디지털 서명의 지원에 대한 보다 많은 정보는 UDDI v3.0 스펙의 "Appendix I, Support For Digital Signatures"를 참고한다.

18.5.2. UDDI 클라이언트 XML 서명 생성 방법

XML 서명은 공개(publication) API 호출에 대한 결과로써 UDDI 레지스트리에 저장되는 상위 수준 element에 대하여 계산된다.

각 API의 호출별 수행되는 element는 다음과 같다.

<table>
<thead>
<tr>
<th>API</th>
<th>element</th>
</tr>
</thead>
<tbody>
<tr>
<td>save_business</td>
<td>businessEntity</td>
</tr>
<tr>
<td>save_service</td>
<td>businessService</td>
</tr>
<tr>
<td>save_binding</td>
<td>bindingTemplate</td>
</tr>
<tr>
<td>save_tModel</td>
<td>tModel</td>
</tr>
<tr>
<td>set_publisherAssertions</td>
<td>publisherAssertion</td>
</tr>
<tr>
<td>add_publisherAssertions</td>
<td></td>
</tr>
</tbody>
</table>

JEUS UDDI 클라이언트는 'java.security.*'의 java.security.PrivateKey 객체와 java.security.cert.X509Certificate 객체를 사용하여 상위 수준 UDDI Entity를 서명한다.

다음은 PrivateKey 객체와 X509Certificate 객체를 얻는 간단한 예제이다.

```java
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.cert.X509Certificate;
...

String keystoreType = "JKS";
String privateKeyAlias = "...";
String privateKeyPasswd = "...";
String certificateAlias = "...";
...
KeyStore keyStore = KeyStore.getInstance(keystoreType);
```
keyStore.load(...);

PrivateKey  privateKey = (PrivateKey)keyStore.getKey(
    privateKeyAlias, privateKeyPasswd.toCharArray());
X509Certificate  certificate =
    (X509Certificate)keyStore.getCertificate(certificateAlias);

얻어진 PrivateKey 객체와 X509Certificate 객체를 사용하여 XML 서명을 지원하는 JEUS UDDI 클라이언트 프로그래밍 방법은 기본적으로 기존의 JEUS UDDI 클라이언트 프로그래밍 방법과 동일하다. 자세한 프로그래밍 방법은 “18.4. UDDI 클라이언트 생성”에서 참고한다.

공개되는 UDDI Entity는 jeus.uddi.v3.client.UDDIClient 클래스에서 제공하는 다음의 메소드를 사용하여 서명된다.

public BindingDetail save_binding(
    String authInfo, bindingTemplate bindingTemplate,
    PrivateKey privateKey, X509Certificate certificate)
public BusinessDetail save_business(
    String authInfo, BusinessEntity businessEntity,
    PrivateKey privateKey, X509Certificate certificate)
public ServiceDetail save_service(
    String authInfo, BusinessService businessService,
    PrivateKey privateKey, X509Certificate certificate)
public TModelDetail save_tModel(
    String authInfo, TModel tModel,
    PrivateKey privateKey, X509Certificate certificate)
public PublisherAssertions set_publisherAssertions(
    String authInfo, PublisherAssertion publisherAssertion,
    PrivateKey privateKey, X509Certificate certificate)

서명할 UDDI Entity는 Entity key를 포함해야 한다. Entity key가 없는 UDDI Entity를 공개할 경우 UDDI 레지스트리는 그 UDDI Entity에 대하여 새로운 Entity key를 할당하게 되고, 이것은 그 UDDI Entity에 대한 서명을 유효하지 않게 만든다. 그러므로 공개자는 Entity key가 포함된 UDDI Entity를 공개할 역할이 있다.

예를 들어 businessService는 serviceKey와 businessKey를 포함해야 한다. businessService가 bindingTemplate을 포함하고 있으며, bindingTemplate은 bindingKey와 serviceKey를 포함하고 있어야 한다.

18.5.3. UDDI 클라이언트 XML 서명 검증

XML 서명 검증은 요청(inquiry) API 호출에 대한 결과로써 응답되는 상위 수준 element에 대해 수행된다. 각 API의 호출별 수행되는 element는 다음과 같다.

<table>
<thead>
<tr>
<th>API</th>
<th>element</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_businessDetail</td>
<td>businessEntity</td>
</tr>
<tr>
<td>get_serviceDetail</td>
<td>businessService</td>
</tr>
</tbody>
</table>
전형적으로 UDDI 레지스트리는 클라이언트의 서명 검증을 하지 않는다. JEUS UDDI 클라이언트는 서명 검증을 위한 별도의 방법을 제공하지 않는다. 해당 요청 API 호출에 대한 응답에 서명 element가 포함된 경우 내부적으로 서명 검증을 자동 수행한다. 서명이 유효하지 않을 때 java.security.SignatureException을 던진다.

### 18.6. UDDI subscription 사용법

본 절에서는 UDDI subscription의 개념과 생성 방법, 클라이언트 프로그래밍 및 이메일을 받기 위한 서버 설정 방법에 대해 설명한다.

#### 18.6.1. 기본 개념

UDDI v3.0 스펙(specification)의 새로운 subscription API는 동기적(synchronous), 비동기적(asynchronous) 인 방법으로 공지(notification) 기능을 지원한다.

subscription API는 UDDI 레지스트리를 추적할 수 있는 방법을 제공한다. 이를 사용하여 가입자(subscriber)는 특별한 질의(Query) 또는 가입자가 관심을 갖는 Entity를 기반으로 subscription을 수립할 수 있다. 이 subscription을 기반으로 하여 UDDI 정보에 대한 동기적 요청 또는 레지스트리에 의해 가입자에게 이루어지는 비동기적 공지를 통해 UDDI 레지스트리의 내용 변경을 추적할 수 있다.

[그림 18.13] UDDI subscription 흐름도

UDDI subscription은 다음의 경우에 유용하다.

- UDDI 레지스트리에 등록된 신규 비즈니스 또는 서비스에 대한 공지
기존 비즈니스 또는 서비스에 대한 변화를 감지

18.6.2. 생성 방법

subscription의 수립과 그 결과의 감시는 일반적으로 여러 단계가 필요한 과정이다.

1. UDDI 레지스트리가 제공하는 비동기식 공지를 받기 위해서는 서비스를 생성해야 한다(가입자가 동기식 요청만을 사용한다면 이 단계를 생략할 수 있다). 가입자는 가입자가 구현한 HTTP/SOAP 웹 서비스나 이메일을 통해서 subscription 결과를 받도록 선택할 수 있다.

2. subscription을 사용하기 위한 필터 조건을 선택한다. 이것은 응답된 결과에 대한 분석이 용이하도록 보다 제한적으로 조건을 선택해야 한다. 일반적으로 가입자는 그가 사용한 subscription 필터[subscription Filter] 조건이 가능한 제한적임을 보증해야 한다.

3. save_subscription API를 사용하여 subscription 요청을 저장한다.

4. 필요에 따라 HTTP/SOAP 또는 이메일에 의한 공지를 처리한다.

18.6.3. UDDI subscription 예제

본 절에서는 비동기식 공지와 동기식 요청 예제에 대해서 설명한다.

비동기식 공지

다음은 간단한 비동기식 공지 형태의 subscription 작동 예제이다.

1. 비즈니스의 변화를 추적할 subscription을 생성한다.

   이를 위하여 가입자는 bindingTemplate을 가진 서비스를, 공지를 전달할 UDDI 레지스트리에 등록해야 하고, 이 bindingTemplate에는 'notify_subscriptionListener' 서비스 또는 이메일을 통해 공지를 받을 수 있도록 Endpoint를 표시한다.

```xml
<save_binding xmlns="urn:uddi-org:api_v3">
  <authInfo>myAuthCode</authInfo>
  <bindingTemplate bindingKey="" serviceKey="uddi:myservicekey">
    <description>notify_subscriptionListener binding for my subscription.</description>
    <accessPoint URLType="https">
      https://www.myCompany.com/services/notify_subscriptionListener
    </accessPoint>
    <tModelInstanceDetails>
      <tModelInstanceInfo tModelKey="uddi:uddi.org:v3_subscriptionlistener"/>
    </tModelInstanceDetails>
  </bindingTemplate>
</save_binding>
```
2. 주기적(예제에서는 매 5일마다)으로 공지하도록 subscription을 생성하고 저장한다. 예에서 'brief' 속성
   은 공지되는 결과에 단지 Entity key만을 사용하도록 제한하기 위해 사용하였다.

   (출처: http://uddi.org/pubs/uddi-v3.0.2-20041019.htm)
다음은 `notify_subscriptionListener` API를 구현한 클라이언트의 호출로 공지되는 결과이다.

```xml
<notify_subscriptionListener>
  <subscriptionResultsList>
    <coveragePeriod>
      <startPoint>20020101T00:00:00</startPoint>
      <endPoint>20020131T00:00:00</endPoint>
    </coveragePeriod>
    <subscription brief="true">
      <subscriptionFilter>
        <find_service xmlns="urn:uddi-org:api_v3">
          <findQualifiers>
            <findQualifier>
              uddi:uddi.org:findqualifier:sql99:like
            </findQualifier>
          </findQualifiers>
        </find_service>
        <categoryBag>
          <keyedReference
            tModeKey="uddi:uddi.org:ubr:taxonomy:naics"
            keyValue="42112_"/>
        </categoryBag>
        <bindingKey>
          bindingKeyOfTheClientsNotifySubscriptionListenerService
        </bindingKey>
        <notificationInterval>P5D</notificationInterval>
        <maxEntities>1000</maxEntities>
        <expiresAfter>20030101T00:00:00</expiresAfter>
      </subscriptionFilter>
      <keyBag>
        <deleted>false</deleted>
        <serviceKey>matchingKey1</serviceKey>
        <serviceKey>matchingKey2</serviceKey>
        <serviceKey>matchingKey3</serviceKey>
        <serviceKey>matchingKey4</serviceKey>
      </keyBag>
      <keyBag>
        <deleted>true</deleted>
        <serviceKey>matchingKey5</serviceKey>
        <serviceKey>matchingKey6</serviceKey>
      </keyBag>
    </subscription>
  </subscriptionResultsList>
</notify_subscriptionListener>
```

(출처: http://uddi.org/pubs/uddi-v3.0.2-20041019.htm)
‘brief’ 속성을 ‘true’로 설정한 subscription은 결과를 단지 Entity key만으로 공지한다. Entity key는 <keyBag>에 포함되어 공지되며, 이미 삭제된 Entity의 key들을 포함한 <keyBag>은 <deleted>가 ‘true’로 설정되어 공지된다.

동기식 요청

API를 사용하여 동기식으로 subscription 결과를 가져올 수 있다. 동기식 요청을 위한 subscription의 저장은 subscription Listener를 위한 <bindingKey>를 필요로 하지 않는다는 것을 제외하면, 비동기식 공지 예제에 살펴본 subscription의 저장 방법과 동일하다.

다음은 전형적인 get_subscriptionResults API를 사용하는 예제이다.

```xml
<get_subscriptionResults>
  <authInfo>myAuthCode</authInfo>
  <subscriptionKey>mySubscriptionKey</subscriptionKey>
  <coveragePeriod>
    <startPoint>20020101T00:00:00</startPoint>
  </coveragePeriod>
</get_subscriptionResults>
```

(출처: http://uddi.org/pubs/uddi-v3.0.2-20041019.htm)

위의 요청은 동기적으로 <subscriptionResultList> 형태로 응답된다.

```xml
<subscriptionResultList>
  <chunkToken>nodeGeneratedToken</chunkToken>
  <coveragePeriod>
    <startPoint>20020101T00:00:00</startPoint>
  </coveragePeriod>
  <subscription brief="true">
    <subscriptionFilter>
      <find_service xmlns="urn:uddi-org:api_v3">
        <findQualifiers>
          <findQualifier>
            uddi:uddi.org:findqualifier:sql99:like
          </findQualifier>
        </findQualifiers>
        <categoryBag>
          <keyedReference
            tModeKey="uddi:uddi.org:ubr:taxonomy:naics"
            keyName="Motor Vehicle Parts"
            keyValue="42112_"/>
        </categoryBag>
      </find_service>
    </subscriptionFilter>
    <bindingKey>
      bindingKeyOfTheClientsNotifySubscriptionListenerService
    </bindingKey>
  </subscription>
</subscriptionResultList>
```
18.6.4. UDDI subscription 클라이언트 프로그래밍

JEUS UDDI 클라이언트는 subscription API를 지원하기 위해 jeus.uddi.v3.client.UDDIClient 클래스에 다음과 같은 메소드를 제공한다.

```java
public subscriptions save_subscription(String authInfo, Subscription subscription)
public void delete_subscription(String authInfo, String subscriptionKey)
public Subscriptions get_subscriptions(String authInfo)
public SubscriptionResultsList get_subscriptionResults(String authInfo, String subscriptionKey,
CoveragePeriod coveragePeriod, String chunkToken)
```

subscription 클라이언트 프로그래밍은 기본적으로 기존의 JEUS UDDI 클라이언트 프로그래밍 방법과 동일하다. 이때 subscription 필터 조건을 가능한 제한적으로 설정하는 것이 주의해야 한다.

다음은 간단한 JEUS UDDI subscription 클라이언트 프로그래밍의 예이다.

```java
UDDIClient client;
...
AuthToken authToken = client.get_authToken(user, password);

// Make a subscription
Subscription subscription = new Subscription();

// Make a subscriptionFilter
FindBusiness findBusiness = new FindBusiness();
findBusiness.addName(new Name("biz"));
subscription.setSubscriptionFilter(new SubscriptionFilter(findBusiness));
```
등록된 subscription으로부터 얻은 subscription 키는 이 subscription을 delete_subscription API를 사용하여 삭제하거나 get_subscriptionResults API를 사용하여 동기식 요청으로 결과를 얻어올 때 사용된다. get_subscriptions API는 가입자가 등록한 모든 subscription을 조회할 때 사용된다.

가입자가 subscription을 등록할 때 등록된 subscription의 결과가 단지 Entity key만을 사용하도록 제한되길 원한다면 다음과 같이 subscription에 'brief' 속성을 'true'로 설정한다.

```java
// Make a Subscription
Subscription subscription = new Subscription();
Subscription.setBrief(true);
```

가입자가 비동기식 공지를 받기를 원한다면, subscription Listener를 위한 <bindingKey>를 subscription에 설정해야 하고 공지 간격을 설정해야 한다. 공지 간격은 XML 스키마에 정의된 xsd:duration 타입이며, 'PnYnMnDTnHnMnS'을 갖는다.

```java
// Make a Subscription
Subscription subscription = new Subscription();
subscription.setBindingKey(
    bindingKeyOfTheClientsNotifySubscriptionListenerService);
subscription.setNotificationInterval("P5D");
```

### 18.6.5. 이메일 공지를 받기 위한 UDDI 서버 설정

JEUS UDDI 서버가 가입자에게 이메일을 통해 subscription 결과를 공지하기 위해서는 메일 서버에 대한 설정이 필요하다. 이 설정은 JEUS UDDI 서버의 설정 파일인 'uddi.properties'에 설정된다.

```properties
uddi.subscription.mail.smtp.host=...
uddi.subscription.mail.smtp.port=...
uddi.subscription.mail.smtp.from=...
```
18.7. UDDI WSDL Publishing 사용법

본 절에서는 UDDI WSDL Publishing의 사용 방법에 대해 설명한다.

18.7.1. UDDI WSDL Publishing

e-business 표준을 위한 기구인 OASIS에서는 UDDI v2.0, v3.0 스펙(specification) 외에 몇 가지 이에 관련된 Technical Note들에 대해 작업하고 있는데 그 중 하나가 바로 UDDI WSDL Publishing(Using WSDL in a UDDI Registry, Version 2.0.2)이다.

WSDL은 abstract한 인터페이스와 임의의 네트워크 서비스의 프로토콜 바인딩들을 제공함으로써 UDDI 표준을 보완할 수 있다. OASIS의 "Using WSDL in a UDDI Registry"는 바로 이러한 WSDL의 description들과 UDDI의 Data Structure와의 매핑에 관한 권고하는 접근 방식을 밝히고 있다.

자세한 사항은 OASIS TC Using WSDL in a UDDI Registry, Version 2.0.2를 참고한다.

[그림 18.14] WSDL description과 UDDI Data Structure 매핑
18.7.2. wsdl2uddi 사용

본 절에서는 JEOS에서 제공하는 wsdl2uddi 툴에 대해 설명하고, wsdl2uddi를 사용해서 wsdl을 UDDI에 Publishing하는 방법을 설명한다.

wsdl2uddi

JEUS UDDI WSDL Publishing은 이러한 매핑을 자동화해주는 툴인 wsdl2uddi를 제공한다. 기본적인 동작 방법은 콘솔에서 다음과 같이 입력하여 명령어를 수행한다.

```
$ wsdl2uddi
```

사용 방법은 다음과 같다.

```
Usage: wsdl2uddi UDDIVersion WSDLURI [wsdl-options] -uddiInquiry
UDDIInquiryURI -uddiPublish UDDIPublish -uddiUsername UDDIUsername
-uddiPassword UDDIPassword [options]
```

* wsdl-options
  -wsdlUsername username to access the WSDL-URI
  -wsdlPassword password to access the WSDL-URI

* options
  -level LEVEL specify log level.
    LEVEL : SEVERE, WARNING, INFO, FINE, FINER, FINEST
  -verbose same as -level FINE

각 파라미터의 설명은 다음과 같다.

<table>
<thead>
<tr>
<th>파라미터</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDDIVersion (필수값)</td>
<td>앞접은 값은 v2 또는 v3 이다.</td>
</tr>
<tr>
<td>WSDLURI (필수값)</td>
<td>실제 WSDL이 존재하는 곳의 URI 값이다.</td>
</tr>
<tr>
<td>wsdl-options</td>
<td>보안이 설정된 WSDL에 접근할 경우 username과 password를 설정할 때 사용한다.</td>
</tr>
<tr>
<td>UDDIInquiryURI (필수값)</td>
<td>실제 UDDI의 Inquiry URI 값을 지정한다.</td>
</tr>
<tr>
<td>UDDIPublishURI (필수값)</td>
<td>실제 UDDI의 publish URI 값을 지정한다.</td>
</tr>
<tr>
<td>UDDIUsername (필수값)</td>
<td>UDDI에 접근할 때 필요한 username을 설정한다.</td>
</tr>
<tr>
<td>UDDIPassword (필수값)</td>
<td>UDDI에 접근할 때 필요한 password를 설정한다.</td>
</tr>
<tr>
<td>options</td>
<td>로그 레벨을 지정할 경우 사용한다.</td>
</tr>
</tbody>
</table>
임의의 WSDL을 UDDI에 Publishing

wsdl2uddi 툴을 이용하여 하나의 WSDL을 UDDI에 Publishing한다.

```
$ wsdl2uddi v2 BasicAuthServiceTestService.wsdl -uddiInquiry
http://localhost:8088/uddi/inquiry -uddiPublish
http://localhost:8088/uddi/publish -uddiUsername jeus -uddiPassword jeus
```

위와 같이 입력하면 wsdl2uddi 툴은 자동으로 wsdl을 해석해서 알맞은 바인딩 규칙에 따라 UDDI에 Publishing한다.

Publishing된 wsdl을 UDDI에서 확인하면 다음과 같다.

[그림 18.15] UDDI에 Publishing된 wsdl의 모습
본 장에서는 JEUS 7 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)
- SJSXP(Sun Java Streaming XML Parser)

본 장을 진행하기에 앞서 XML 문서 혹은 문서 내의 내용물과 Java 오브젝트의 변환에서 일반적으로 사용되는 용어에 대해 설명한다.

<table>
<thead>
<tr>
<th>구분</th>
<th>의미</th>
</tr>
</thead>
<tbody>
<tr>
<td>언마셜링(Unmarshalling)</td>
<td>XML 문서 혹은 XML 콘텐츠를 Java 클래스로 오브젝트화(Java Content Tree)하는 과정이다.</td>
</tr>
<tr>
<td>마셜링(Marshalling)</td>
<td>Java 오브젝트(Java Content Tree)를 XML 문서 혹은 XML 콘텐츠로 변환하는 과정이다.</td>
</tr>
</tbody>
</table>

19.2. JAXB(XML 바인딩을 위한 Java 아키텍처)

어떤 스키마를 준수하는 XML 문서에 관련된 애플리케이션은 그 문서를 생성하거나 수정하거나 혹은 읽어들일 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 7 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 doesn't 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 @javax.annotation.Generated
- **episode <FILE>** : generate the episode file for separate compilation

**참고**

JEUS 7 웹 서비스는 XJC의 Ant Task도 지원하는데 보다 자세한 설명은 “JEUS Reference Book”의 “4.11. xjc” 및 “JEUS Reference Book”的 “5.5.5. xjc”를 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 마샬링(이 예제에서는 출력)한다.
다음은 이 예제의 build.xml의 일부분이다.

[예 19.1] << build.xml >>

```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" />
      <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.

[예 19.2] << Main.java >>

```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 문서인지 판단하기 위한 Schema 객체를 생성하여 Unmarshaller와 Marshaller에 등록한다.

**예 19.3** << Main.java (계속)>>

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");

polInput.xml이라는 XML 문서를 언마샬링한다. 그리고 언마샬링된 Java 객체를 수정한다.

**예 19.4** << Main.java (계속)>>

marshaller.marshal(ts, System.out);

화면에 출력(마샬링)한다.

19.2.2. 스키마 생성기(Schemagen) 관련 프로그래밍 기법

JEUS 7 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 7 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

참고
JEUS 7 웹 서비스는 Schemagen의 Ant Task도 지원하는데 보다 자세한 설명은 "JEUS Reference Book"의 "4.12. schemagen" 및 "JEUS Reference Book"의 "5.5.6. schemagen"을 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.

[예 19.5] << build.xml >>

```xml
...
<project basedir="." default="run">
...

  <taskdef name="schemagen" class="com.sun.tools.jxc.SchemaGenTask">
    <classpath refid="classpath" />
  </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>
  </target>

```
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.

[예 19.6] << Main.java >>

```java
BusinessCard card =
    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 Marshaller로 하나 등록하여 bcard.xml 이라는 파일에 마셜링한다.
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)


- SAX
- DOM
- TrAX
- DOM
- StAX

19.3.1. StAX(Java 스트리밍 XML 파서)

Part II. JAX-RPC 웹 서비스의 지원

JEUS 7 웹 서비스는 JAX-WS 방식의 웹 서비스를 공식 지원하면서 동시에 JEUS 5 웹 서비스에서 지원하던 JAX-RPC 웹 서비스 또한 하위 호환성을 위해 공식 지원한다. 지금까지 JAX-WS 웹 서비스를 구성해서 서비스하고 클라이언트 프로그램을 작성해서 이를 호출하는 등 여러 가지 JAX-WS의 기능들에 대해 알아보았다. 본 Part부터는 하위 호환성을 위한 JAX-RPC 웹 서비스에 대한 JEUS 7 웹 서비스의 지원에 대해 설명한다.

간단히 JAX-WS 방식의 웹 서비스와 JAX-RPC 방식의 웹 서비스의 차이점에 대해 설명하면 다음과 같다. 우선 JAX-WS 방식의 웹 서비스는 Java EE 6 웹 서비스의 표준이며 JAX-RPC 방식의 웹 서비스는 J2EE 1.4 웹 서비스의 표준이다. JEUS 7 웹 서비스는 Java EE 6 웹 서비스의 표준인 JAX-WS 방식의 웹 서비스를 공식 지원하며, 동시에 JEUS 5에서 지원하던 J2EE 1.4 웹 서비스 표준의 JAX-RPC 방식의 웹 서비스 또한 하위 호환성을 위해 지원 한다.

JAX-RPC 방식의 웹 서비스를 구현하기 위해서는 JAX-WS 방식의 웹 서비스에 비해 몇 가지 추가로 구현해야 하는 것들이 있다. Deployment Descriptor 및 JAX-RPC 매핑 파일 등이 그것인데 앞으로 나오는 여러 장들을 통해 이러한 JAX-RPC 웹 서비스를 JEUS 7 웹 서비스가 어떻게 지원하고 있는지 자세하게 설명한다.
본 장에서는 JAX-RPC 웹 서비스를 구현하는 여러 가지 방식에 대한 설명한다.

20.1. 개요

JEUS JAX-RPC 웹 서비스 구현 작업은 웹 서비스를 구현하는 Back-end 구성 요소인 Java 코드를 작성 또는 컴파일하는 것을 의미한다. 기본적으로 JEUS JAX-RPC 웹 서비스는 Java 클래스, EJB와 같은 2가지 타입의 웹 서비스 Back-end 구성 요소를 지원한다.

JEUS JAX-RPC 웹 서비스를 구현하는 주요 절차는 다음과 같다.
1. 웹 서비스를 구성하는 Back-end 구성 요소인 Java 코드를 작성한다.
2. SOAP 메시지의 내용을 직접 다루기를 원하거나, SOAP 메시지의 Attachment에 직접 접근하기를 원한다면 직접 SOAP 메시지 핸들러나 핸들러 체인(Chain)을 생성한다.
3. Java 코드를 컴파일한다.

20.2. Java 클래스 웹 서비스의 구현

Java 클래스 웹 서비스는 JAX-RPC 웹 서비스의 구현에서 웹 서비스를 생성하는 가장 간단한 방법이다. 간단한 예를 통해 구현하는 방법에 대해 설명한다.

20.2.1. 간단한 예제

Java 클래스 웹 서비스를 생성하기 위해서는 SEI와 구현 클래스를 정의해야 한다.

SEI는 Java 클래스 웹 서비스 Endpoint가 Java 메소드의 형식으로 지원할 웹 서비스 오퍼레이션들을 정의한다. 구현 클래스는 이러한 SEI들을 구현한다.

다음의 Endpoint 인터페이스는 echoString과 echoString_double이라는 웹 서비스 오퍼레이션들을 정의하고 있다.

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

public interface Echo extends java.rmi.Remote {
    public String echoString(String arg1)
        throws java.rmi.RemoteException;
    public String echoString_double(String arg1, String arg2)
        throws java.rmi.RemoteException;
}
```

다음이 구현 클래스에 대한 예이다.

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

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

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

구현 클래스는 Endpoint 인터페이스를 구현하고 있다. Java EE 서버 내에서 인스턴스화되어 실행되며 런타임에는 웹 서비스로 동작하게 될 것이다.

20.2.2. Java 클래스의 작성이 원칙

Java 클래스 Back-end로서 웹 서비스를 구현할 때 다음의 규칙들을 준수해야 한다.

- **SEI를 정의한다.**
- **public 클래스를 정의한다.**
- **파라미터 없는 디폴트 생성자를 정의한다.**
- **웹 서비스에서 public, non-static으로 Export되는 Java 클래스의 메소드들을 정의한다.**
- **스레드 안전화(Thread-safety) Java 코드를 작성한다.**
- **상호 운용성을 위해서 오버 로딩된 메소드들을 사용하지 말아야 한다.**

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

- **웹 서비스 메소드를 포함하고 있는 Java 클래스를 정의한다.**
- **서비스로 명시적으로 공개할 메소드를 위한 인터페이스를 정의한다.**
20.3. EJB 웹 서비스의 구현

EJB는 Java EE 웹 서비스를 개발하는 데 있어 이전에 언급되었던 Java 클래스 웹 서비스 프로그래밍 모델보다 좀 더 복잡하지만 더 풍부한 기능을 가질 수 있게 하는 프로그래밍 모델이다. 이러한 복잡성을 트랜잭션을 자동으로 관리하게 되면서 부수적으로 발생하는 것이라고 볼 수 있다.

EJB 웹 서비스를 구현하기 위해서는 EJB에 대한 어느 정도의 이해를 필요로 한다. 만약 웹 서비스를 구현 하는 데 있어 EJB를 사용하지 않아도 된다면, 본 장의 내용은 축지하지 않아도 된다.

20.3.1. 간단한 예제

Stateless Session EJB 또한 웹 서비스로 Export될 수 있다. 이 경우에도 SEI가 필요하다. 다음의 Endpoint 인터페이스(HelloIF.java)는 sayHello(String) 웹 서비스 오퍼레이션을 정의하고 있다.

[예 20.1] << HelloIF.java >>

```java
package helloejb;

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

public interface HelloIF extends Remote {
    public String sayHello(String s) throws RemoteException;
}
```


다음은 리모트 인터페이스의 예이다.

[예 20.2] << Hello.java >>

```java
package helloejb;

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

public interface Hello extends EJBObject {
    String sayHello(String s) throws RemoteException;
}
```
다음은 홈 인터페이스의 예이다.

**[예 20.3] << HelloHome.java >>**

```java
package helloejb;

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

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

다음은 Session Bean을 구현한 EJB Endpoint Bean 클래스이다.

**[예 20.4] << HelloEJB >>**

```java
package helloejb;

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

public class HelloEJB implements SessionBean {

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

웹 서비스 오퍼레이션이 one-way로 설정되어 있으면 EJB 메소드의 Java 코드는 void 타입을 리턴해야 한다. 이 조건을 제외하면 Stateless Session EJB 웹 서비스를 위한 코딩 작업은 일반 EJB 코딩 작업과 다른 점이 없다.

이와 같은 조건을 만족하도록 다음과 같은 절차에 따라 웹 서비스를 구현한다.
1. 웹 서비스 메소드를 포함하고 있는 EJB를 구현한다.
2. 서비스로 명시적으로 공개할 메소드를 위한 인터페이스를 정의한다.

20.4. WSDL로부터 웹 서비스 구현

본 안내서의 대부분은 웹 서비스를 구성하는 Back-end 구성 요소인 Java 코드를 작성하는 것으로부터 시작하는 것을 가정하고 있다.

개발 환경에 따라서 WSDL로부터 서비스 인터페이스를 생성할 수 있다. 이 경우 사용자는 wsdl2java Ant Task를 수행하여 서비스 인터페이스를 얻을 수 있다. wsdl2java Ant Task는 사용자가 작성했거나 가지고 있는 WSDL 파일을 입력으로 한다.

다음은 wsdl2java Ant를 수행하기 위한 빌드 파일의 예이다.

[예 20.5] << build.xml >>

```xml
<target name="do-package-war">
  ...
  <antcall target="wsdl2java">
    <param name="wsdl2java.option" value="-import:server -d ${build.war.dir}/WEB-INF/classes
      -package sample.datahandleronly.service
      -outputmapping ${build.war.dir}/WEB-INF/SubmitBookService-mapping.xml
      -compile ${src.web}/WEB-INF/wsdl/SubmitBookService.wsdl" />
  </antcall>
  ...
</target>
```

다음과 같은 명령을 수행하여 웹 서비스 서비스 인터페이스 소스 코드를 생성할 수 있다.

prompt>ant do-package-war

사용자는 생성된 서비스 인터페이스에 대한 서비스 구현체를 “20.2. Java 클래스 웹 서비스의 구현”과 “20.3. EJB 웹 서비스의 구현”의 구현 방식에 따라서 생성한다.
참고

wsdl2java Task에 대한 더 자세한 설명은 “JEUS Reference Book”의 “4.9. wsdl2java” 또는 “JEUS Reference Book”의 “5.5.2. wsdl2java”를 참고한다.

20.5. SAAJ의 사용

통상적인 SOAP 메시지는 SOAP Body 안에 포함되지만 특정한 Java 타입을 웹 서비스 오퍼레이션을 구성하는 메소드의 파라미터나 런타임 타입으로 사용할 경우에는 SOAP Attachment 형태로 전송된다.

JEUS 웹 서비스는 SAAJ(SOAP with Attachments API for Java)의 사용을 위해 Java에서 MIME 타입으로의 형태 전환을 다음과 같이 정의한다.

[표 20.1] Required Mappings : Java to MIME

<table>
<thead>
<tr>
<th>Java 타입</th>
<th>MIME 타입</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.awt.Image</td>
<td>image/gif or image/jpeg</td>
</tr>
<tr>
<td>java.lang.String</td>
<td>text/plain</td>
</tr>
<tr>
<td>javax.mail.internet.MimeMultipart</td>
<td>multipart/*</td>
</tr>
<tr>
<td>javax.xml.transform.Source</td>
<td>text/xml or application/xml</td>
</tr>
</tbody>
</table>

위에 열거된 각각의 MIME 타입에 대해 JAX-RPC Endpoint Stub이 특정한 Java 타입을 적절히 인코딩된 데이터의 스트림으로 변환 또는 역변환 작업을 한다. 보다 자세한 설명은 “제23장 JAX-RPC 웹 서비스 SOAP 메시지 핸들러 생성”을 참고한다.
제21장 JAX-RPC 웹 서비스 생성과 배치

본 장에서는 Java 클래스와 EJB를 Endpoint로 갖는 JAX-RPC 웹 서비스의 생성과 배치 방법에 대해 설명한다.

21.1. Java 클래스 웹 서비스 생성과 배치

JEUS JAX-RPC 웹 서비스 생성은 개발 편의를 제공하기 위해 Command Line 툴과 Apache Ant 툴을 사용한다. JAX-RPC 웹 서비스 생성과 배치 작업은 다음의 작업들이 순차적으로 진행된다.

1. 서비스 설정 파일의 작성(service-config.xml)
2. Java EE 웹 서비스를 위한 WSDL과 JAX-RPC 매핑 파일의 생성
3. 웹 서비스 DD 작성(webservices.xml, jeus-webservices-dd.xml)
4. 생성된 웹 서비스 모듈의 패키징과 배치

참고

자세한 사항은 “JEUS Reference Book”의 “4.8. java2wsdl”과 “JEUS Reference Book”의 “4.9. wsdl2java” 및 “JEUS Reference Book”의 “5.5.1. java2wsdl”과 “JEUS Reference Book”의 “5.5.2. wsdl2java”를 참고한다.

21.1.1. 서비스 설정 파일 작성

JEUS JAX-RPC 웹 서비스는 service-config.xml과 같이 웹 서비스 생성을 위한 설정을 XML 파일에 저장한다.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service>
    <service-name>DocLitEchoService</service-name>
    <target-namespace>urn:DocLitService</target-namespace>
    <output-wsdl-file>DocLitEchoService.wsdl</output-wsdl-file>
    <output-jaxrpc-mapping-file>DocLitEchoService-mapping.xml</output-jaxrpc-mapping-file>
    <style>wrapped</style>
  </interface>
  <endpoint-interface-class>jeustest.webservices.java2wsdl.doclit.Echo</endpoint-interface-class>
</service>
</web-services-config>
```
위 예에서는 `<service-name>`에 설정된 'DocLitEchoService'라는 이름의 웹 서비스를 생성하게 되며, 생성되는 WSDL과 매핑 파일은 각각 DocLitEchoService.wsdl과 DocLitEchoService-mapping.xml이 된다.

서비스 스타일은 문서 방식 중 WRAPPED 방식이며, 웹 서비스 Endpoint로는 jeustest.webservices.java2wsdl.doclit.Echo라는 Java 클래스 파일로 설정되었다.

참고

service-config.xml 파일에 대한 자세한 설명은 "JEUS XML Reference"의 "25. jeus-webservices-config.xml 설정"을 참고한다.

### 21.1.2. WSDL 파일과 JAX-RPC 매핑 파일 생성

JEUS JAX-RPC 웹 서비스 생성을 위해 Command Line 툴 방식과 Ant Task 방식을 제공한다. 2가지 방법 중 어느 것을 선택해도 무방하며 사용자 편의에 따라 결정한다.

#### Command Line 툴 이용

JAX-RPC 웹 서비스는 웹 서비스의 생성을 위한 Command Line 툴을 제공한다.

**Usage:** java2wsdl [options] configuration_file

where [options] include:

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

"21.1.1. 서비스 설정 파일 작성"에서 생성한 service-config.xml 파일로 웹 서비스를 생성하려면 Command Line에 명령어를 입력한다.

다음은 JEUS_HOME/sample/classes아래 컴파일된 Java 클래스 Back-end 파일이 있는 경우 명령어 실행의 예이다.

JEUS_HOME/sample$ java2wsdl -cp ./classes service-config.xml

위와 같이 명령을 수행하면 JAX-RPC 매핑 파일인 DocLitEchoService-mapping.xml과 WSDL 파일인 DocLitEchoService.wsdl이 생성된다.

#### Ant 툴 이용

JEUS JAX-RPC 웹 서비스는 웹 서비스의 생성을 위한 Ant Task인 java2wsdl을 제공한다. java2wsdl은 서비스 설정 파일의 위치를 입력받아서 WSDL 파일과 JAX-RPC 매핑 파일을 생성한다.
다음과 같이 수행하여 Java 클래스 파일들을 컴파일한다.

`ant compile`

WSDL 파일과 JAX-RPC 매핑 파일 또한 위의 명령어를 수행하면 `/build` 디렉터리에 자동 생성된다.

compiled Ant Task는 내부적으로 `-pre-compile > do-compile > -post-compile` 과정을 거치며 WSDL 파일과 JAX-RPC 매핑 파일을 생성하는 `-post-compile`은 다음과 같이 구성되어 있다.

```xml
<target name="-post-compile">
  <mkdir dir="${build.war.dir}/WEB-INF/wsdl"/>
  <antcall target="java2wsdl">
    <param name="java2wsdl.option" value="-classpath ${build.classes.dir} -d ${build.war.dir}/WEB-INF ${src.conf}/service-config.xml"/>
  </antcall>
</target>
```

참고

java2wsdl Ant Task의 자세한 내용은 “JEUS Reference Book”의 “5.5.1. java2wsdl”를 참고한다.

21.1.3. 웹 서비스 DD 파일 작성

웹 서비스 DD 파일은 웹 서비스의 배치에 관한 기술한 XML 파일로 웹 서비스의 배치에 관한 정보와 웹 서비스 Back-end를 발견하는 방법에 대한 정보를 웹 서비스 엔진에게 제공한다.

웹 서비스 DD 파일에는 Java EE 웹 서비스 스펙에 규정된 DD 파일인 `webservices.xml`과 JEUS 웹 서비스를 위한 DD 파일인 `jeus-webservices-dd.xml`이 있다.

Java EE 웹 서비스 DD 파일 작성

Java EE 웹 서비스 DD 파일명은 반드시 `webservices.xml`로 해야 한다.

[예 21.1] << webservices.xml >>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<webservices version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <webservice-description>
    <webservice-description-name>DocLitEchoService</webservice-description-name>
    <wsdl-file>WEB-INF/wsdl/DocLitEchoService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>WEB-INF/DocLitEchoService-mapping.xml</jaxrpc-mapping-file>
    <port-component>
      <port-component-name>EchoPort</port-component-name>
```

제21장 JAX-RPC 웹 서비스 생성과 배치 245
DocLitEchoService라는 서비스 이름을 가지고, WSDL 파일의 위치는 WAR 파일 내에서 WEB-INF/.wsdl 디렉터리 내에 있으며, 그 이름은 DocLitEchoService.wsdl이다. JAX-RPC 매핑 파일의 위치는 WEB-INF 디렉터리이며, DocLitEchoService-mapping.xml이라는 이름을 가지고 있다. 이 서비스에 접근하려면 WSDL에 표기된 포트 중 EchoPort라는 이름을 가지고 있는 포트를 사용하고, 이 포트에 대한 SEI와 서블릿 라이브러리가 정의된다. 보다 자세한 설명은 "제24장 JAX-RPC 웹 서비스 설정 파일 작성"을 참고한다.

JEUS 웹 서비스 DD 파일 작성

JEUS 웹 서비스 DD 파일명은 jeus-webservices-dd.xml로 해야 한다.

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

참고

DD 파일에 대한 자세한 설명은 "JEUS XML Reference"의 "23. jeus-webservices-dd.xml"을 참고한다.
21.1.4. 패키징과 배치

Java 클래스 웹 서비스는 Java 클래스와 웹 서비스 DD 파일을 웹 모듈처럼 패키징한다. 웹 서비스는 웹 애플리케이션 WAR 파일과 EJB JAR 파일을 포함하는 표준 엔터프라이즈 애플리케이션(Enterprise Application) EAR 파일로 묶이게 된다.

참고

JEUS 웹 컨테이너의 컨텍스트와 웹 애플리케이션에 대한 보다 자세한 설명은 "JEUS Web Engine 안내서"의 "제3장 웹 컨텍스트"를 참고한다.

21.1.4.1. 서블릿 DD 파일 작성

서블릿 DD 파일(web.xml)의 일반적인 역할은 서블릿과 JSP 컴포넌트의 런타임 속성을 기술하는 것이다. Java 클래스 웹 서비스는 런타임에 서블릿에 Embed된 형태로 연동되므로 web.xml 파일은 Java 클래스 웹 서비스를 배치할 때 필요하다.

다음은 web.xml 파일의 작성 예이다.

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

<servlet-class>는 실제 서비스를 구현한 로직이 들어있는 Java 클래스의 이름을 텍스트 노드의 값으로 설정하고, <servlet-mapping>의 하위에 <url-pattern>은 실제로 서비스에 접근하기 위한 URL을 값을 설정한다.

21.1.4.2. JEUS 웹 모듈 DD 파일 작성

JEUS 웹 모듈 DD 파일은 배치하려는 웹 모듈의 컨텍스트를 정의한 파일로 이름은 jeus-web-dd.xml이다.

다음은 DocLitEchoService를 컨텍스트로 가지는 서비스의 JEUS 웹 모듈 DD 파일이다.
21.1.4.3. WAR 패키징

WAR 파일은 확장명이 '.war'로 끝나는 JAR 압축 파일이며 압축과 압축 해제 알고리즘인 zlib 알고리즘 표준을 따라 압축되는 파일이다. WAR 파일은 서블릿, JSP와 같은 웹 컴포넌트들을 묶는 용도로만 사용되며 다음과 같이 특정한 디렉터리 구조를 가지고 있어야 한다.

```
| WEB-INF
|   |-- web.xml (서블릿 DD)
|   |-- webservices.xml (표준 웹 서비스 DD)
|   |-- jeus-webservices-dd.xml (JEUS 웹 서비스 DD)
|   |-- jeus-web-dd.xml (JEUS 웹 모듈 DD)
|   |-- Jax-rpc mapping 파일
|       |-- wsdl
|           |-- wsdl 파일
|       |-- classes
|           |-- Java 클래스 컴포넌트, 핸들러 구현 등
```

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

JAX-RPC 매핑 파일과 WSDL 파일이 webservices.xml 파일에 <wsdl-file>과 <jax-rpc-mapping-file>로 기술한 위치에 존재한다면 반드시 위의 구조를 따르지 않아도 무방하다.
21.1.4.4. EAR 패키징과 배치

본 절에서는 EAR 패키징과 배치에 대해서 설명한다.

**EAR 패키징**

Java EE 응용 프로그램은 웹 컴포넌트나 EJB, Java EE 커넥터를 사용할 수 있는 하나의 독립적인 비즈니스 솔루션이며, EAR(Enterprise ARchive) 파일로 패키징될 수 있다. EAR 파일은 응용 프로그램의 XML DD를 가지고 있으며, Java EE 컴포넌트와 커넥터 등을 EJB JAR나 WAR 또는 RAR 파일 형태로 패키징 하여 포함하고 있다.

Java 클래스 웹 서비스는 서블릿 프로그래밍 모델 위에서 생성되었으므로 WAR 파일 형태로 패키징되고, EJB Endpoint는 SOAP 메시지를 다루는 곳이 EJB이므로 EJB JAR 파일 형태로 패키징된다.

```
EAR
|-- WAR 파일
|-- JAR 파일
|   |-- EJB 웹 서비스 구성 요소
|-- META-INF
|   |-- 표준 application.xml 파일
```

META-INF 디렉터리에 있는 표준 application.xml 파일에 EAR 패키징한 Java EE 컴포넌트들을 기술한다. 다음은 웹 모듈을 Java EE 구성요소로 가지고 있는 EAR 응용 프로그램의 application.xml 파일의 작성 예이다.

[예 21.5] << application.xml >>

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application version="6"
  xmlns="http://java.sun.com/xml/ns/javae">
  <module>
    <web>
      <web-uri>DocLitEchoService.war</web-uri>
      <context-root>DocLitEchoService</context-root>
    </web>
  </module>
</application>
```

<web>에는 웹 모듈 패키징의 이름과 모듈의 컨텍스트의 정보를 설정한다.
웹 서비스의 배치(EAR 응용 프로그램의 배치)
웹 서비스의 배치 작업은 일반적인 Java EE 응용 프로그램의 배치 작업과 동일하다. EAR 배치 작업을 수행함으로써 웹 서비스는 인터넷에서 접근 가능한 서비스로 공개된다.
배치된 서비스의 실제 접근 가능한 URL 주소는 다음과 같다.

http://host:port/DocLitEchoService/DocLitEchoService

21.2. EJB 웹 서비스 생성과 배치

EJB를 이용한 웹 서비스의 구현 작업을 완료한다면 실제로 EJB의 비즈니스 로직을 웹 서비스로 전환하고 배치하는 작업이 필요하다. EJB 웹 서비스 생성과 배치 작업의 전체적인 흐름은 Java 클래스 웹 서비스 생성, 배치 작업과 같다. JEUS JAX-RPC 웹 서비스 생성은 개발 편의를 제공하기 위해 Command Line 툴과 Apache Ant 툴을 사용한다.
전체적인 작업 흐름은 다음과 같다.
1. 서비스 설정 파일의 작성(service-config.xml)한다.
2. Java EE 웹 서비스를 위한 WSDL과 JAX-RPC 매핑 파일의 생성한다.
3. 웹 서비스 DD 작성(webservices.xml, jeus-webservices-dd.xml, ejb-jar.xml)한다.
4. 생성된 웹 서비스 모듈의 패키징과 배치한다.

참고
자세한 사항은 “JEUS Reference Book”의 “4.8. java2wsdl”과 “JEUS Reference Book”의 “4.9. wsdl2java” 및 “JEUS Reference Book”의 “5.5.1. java2wsdl”과 “JEUS Reference Book”의 “5.5.2. wsdl2java”를 참고한다.

21.2.1. 서비스 설정 파일 작성

다음은 서비스 설정 파일 service-config.xml의 작성 예이다.

[예 21.6] << service-config.xml >>

```xml
<?xml version="1.0"?>
<web-services-config xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service>
    <service-name>AddressBookService</service-name>
    <target-namespace>urn:AddressBookService</target-namespace>
    <style>wrapped</style>
    <use>literal</use>
    <interface>
      <endpoint-interface-class>
        address.AddressBookIF
      </endpoint-interface-class>
    </interface>
  </service>
</web-services-config>
```
위 예에서는 `<service-name>`에 설정된 AddressBookService라는 이름의 웹 서비스를 생성하게 되며, 생성되는 WSDL과 매핑 파일은 각각 `AddressBookService.wsdl`과 `AddressBookService-mapping.xml`이 된다. 서비스 스타일은 문서 방식 중 WRAPPED 방식이며, 웹 서비스 Endpoint로는 `address.AddressBookIF`라는 Java 클래스 파일로 설정되었다.

참고

`service-config.xml` 파일에 대한 보다 자세한 설명은 "JEUS XML Reference"의 "25. jeus-webservices-config.xml 설정"을 참고한다.

### 21.2.2. WSDL 파일과 JAX-RPC 매핑 파일 생성

JEUS JAX-RPC 웹 서비스는 웹 서비스의 생성을 위해 Command Line 툴 방식과 Ant Task 방식을 제공한다. 웹 서비스 생성을 위해 2가지 방법 중 어느 것을 선택해도 무방하며, 사용자 편의에 따라 결정한다.

#### Command Line 툴 사용

JEUS JAX-RPC 웹 서비스는 웹 서비스의 생성을 위한 Command Line 툴을 제공한다.

```
Usage: java2wsdl [options] configuration_file
```

where [options] include:

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

이전 장에서 생성한 `service-config.xml` 파일로 웹 서비스를 생성하려면 Command Line에서 다음과 같이 입력하여 명령을 수행한다.

다음 예는 컴파일된 Java 클래스 Back-end 파일들이 JEUS_HOME/sample/classes에 존재하는 경우이다.

```
JEUS_HOME/sample$ java2wsdl -cp ./classes ejb-service-config.xml
```

위와 같이 명령을 수행하면, JAX-RPC 매핑 파일인 AddressBookService-mapping.xml과 WSDL 파일인 AddressBookService.wsdl이 생성된다.
Ant를 사용

JEUS JAX-RPC 웹 서비스는 웹 서비스의 생성을 위한 Ant Task인 java2wsdl를 제공한다. java2wsdl은 입력으로 서비스 설정 파일의 위치를 받아서 WSDL 파일과 JAX-RPC 매핑 파일을 생성한다.

Java 클래스 파일들을 다음과 같은 명령을 수행하여 컴파일한다.

```
ant compile
```

WSDL 파일과 JAX-RPC 매핑 파일 또한 위의 명령어를 수행하면 ./build 디렉터리에 자동 생성된다. 컴파일 Ant Task는 내부적으로 -pre-compile > do-compile > -post-compile 과정을 거쳐며 WSDL 파일과 JAX-RPC 매핑 파일을 생성하는 -post-compile은 다음과 같이 구성되어 있다.

```
<target name="-post-compile">
  <mkdir dir="${build.classes.dir}/META-INF/wsdl"/>
  <antcall target="java2wsdl">
    <param name="java2wsdl.option" value="-classpath ${build.classes.dir}-d ${build.classes.dir}/META-INF ${src.conf}/ejb-service-config.xml"/>
  </antcall>
</target>
```

서비스 설정 파일 경로를 <java2wsdl>의 'configfilepath' 속성에 입력하고, 컴파일된 Java 클래스 파일들의 경로를 <classpath>의 'refid' 속성에 입력한다. 위와 같은 명령을 수행하고 나면 WSDL 파일과 JAX-RPC 매핑 파일이 생성된다.

참고

java2wsdl Task에 관한 보다 자세한 정보는 “JEUS Reference Book”의 “5.5.1. java2wsdl”을 참고한다.

21.2.3. 웹 서비스 DD 파일 작성

웹 서비스 DD 파일은 웹 서비스의 배치에 관해 기술한 XML 파일로 웹 서비스의 배치에 관한 정보와 웹 서비스 Back-end를 발견하는 방법에 대한 정보를 웹 서비스 엔진에게 제공한다.

웹 서비스 DD 파일은 Java EE 웹 서비스 스펙에 규정된 DD 파일인 webservices.xml과 JEUS 웹 서비스를 위한 DD 파일인 jeus-webservices-dd.xml이 있다.

Java EE 웹 서비스 DD 파일 작성

Java EE 웹 서비스 DD 파일명은 반드시 webservices.xml로 해야 한다.

[예 21.7] << webservices.xml >>

```
<?xml version="1.0"?>
<webservices version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee">
```
AddressBookService라는 서비스 이름을 가지고, WSDL 파일의 위치는 JAR 파일 내에서 META-INF/wsl디렉터리 내에 있으며, 그 이름은 AddressBookService.wsdl이다. JAX-RPC 매핑 파일의 위치는 META-INF 디렉터리이며, AddressBookService-mapping.xml이라는 이름을 가지고 있다.

이 서비스에 접근하려면 WSDL에 표기된 포트 중 AddressBookIFPort라는 이름을 가지고 있는 포트를 사용하고, 이 포트에 대한 SEI와 EJB가 정의된다.

**JEUS 웹 서비스 DD 파일 작성**

JEUS 웹 서비스 DD 파일명은 jeus-webservices-dd.xml로 해야 한다.

[예 21.8] << jeus-webservices-dd.xml >>

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <ejb-context-path>webservice</ejb-context-path>
  <service>
    <webservice-description-name>
      AddressBookService
    </webservice-description-name>
  </service>
</jeus-webservices-dd>
```
JEUS 웹 서비스 DD 파일에는 EJB 웹 서비스로 접근하기 위한 URL 접근 경로를 기술한다. URL 접근 경로는 컨텍스트 경로와 URL 패턴을 포함한다. 컨텍스트 경로는 <ejb-context-path>로 지정한다. URL 패턴은 <port> 아래의 <ejb-endpoint-url>로 지정한다.

위의 경우 웹 서비스에 접근하기 위한 실제 URL 주소는 다음과 같다.

http://host:port/webservice/AddressBookService

21.2.4. 패키징과 배치

EJB 웹 서비스는 EJB JAR 패키지 안에 웹 서비스 DD 파일과 WSDL을 같이 포함한다. 웹 서비스는 웹 애플리케이션 WAR 파일과 EJB JAR 파일을 포함하는 표준 엔터프라이즈 애플리케이션(Enterprise Application) EAR 파일로 묶이게 된다.

21.2.4.1. EJB DD 파일 작성

다음은 ejb-jar.xml 파일의 한 예이다.

[예 21.9] << ejb-jar.xml >>

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

<ejb-class>는 실제 서비스를 구현한 로직이 들어있는 Bean 클래스의 이름을 텍스트 노드의 값으로 갖는다. <service-endpoint>는 EJB 웹 서비스의 SEI 클래스의 이름을 텍스트 노드의 값으로 갖는다.

다음은 JEUS에 필요한 또 다른 EJB DD이다. 파일 이름은 jeus-ejb-dd.xml이다.

[예 21.10] << jeus-ejb-dd.xml >>

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

### 21.2.4.2. JAR 패키징

지금까지 작성한 EJB 구현 클래스들과 SEI 그리고 웹 서비스 DD, EJB DD를 하나의 JAR 파일로 패키징 한다.

JAR 패키징한 구조는 다음과 같다.

```
JAR
  |-- META-INF
  |   |-- ejb-jar.xml (표준 EJB DD)
  |   |-- webservices.xml (표준 웹 서비스 DD)
  |   |-- jeus-webservices-dd.xml (JEUS 웹 서비스 DD)
  |   |-- jeus-ejb-dd.xml (JEUS EJB DD)
  |   `-- Jax-rpc mapping 파일
  |         |-- wsdl
  |           |   |-- wsdl 파일
  |           `-- EJB 클래스 컴포넌트, J2EE Portable Artifact
```

JAX-RPC 매핑 파일과 WSDL 파일이 webservices.xml 파일의 <wsdl-file>과 <jax-rpc-mapping-file>로 기술한 위치에 존재한다면 반드시 위의 구조를 따르지 않아도 무방하다.
21.2.4.3. EAR 패키징과 배치

본 절에서는 EAR 패키징과 배치에 대해서 설명한다.

EAR 패키징

Java EE 응용 프로그램은 웹 컴포넌트나 EJB, Java EE 커넥터를 사용할 수 있는 하나의 독립적인 비즈니스 솔루션이며, EAR(Enterprise ARchive) 파일로 패키징될 수 있다. EAR 파일은 응용 프로그램의 XML DD를 가지고 있으며, Java EE 컴포넌트와 커넥터 등을 EJB JAR나 WAR, 혹은 RAR 파일 형태로 패키징 하여 포함하고 있다.

JSE(JAX-RPC Service Endpoint)는 서블릿 프로그래밍 모델 위에서 생성되었으므로 WAR 파일 형태로 패키징되고, EJB Endpoint는 SOAP 메시지를 다루는 곳이 EJB이므로 EJB JAR 파일 형태로 패키징된다.

META-INF 디렉터리 안의 표준 application.xml 파일에 EAR 패키징한 Java EE 콤포넌트들을 기술한다.

다음은 웹 모듈을 Java EE 구성요소로 가지고 있는 EAR 응용 프로그램의 application.xml 파일의 한 예이다.

[예 21.11] << application.xml >>

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<application version="6"
  xmlns:ns1="http://java.sun.com/xml/ns/javaee">
  <module>
    <ejb>AddressBook.jar</ejb>
  </module>
</application>
```

<ejb>에서 EJB 웹 서비스의 JAR 패키징의 이름에 대한 정보를 가지고 있다.

웹 서비스의 배치(EAR 응용 프로그램의 배치)

웹 서비스의 배치 작업은 일반적인 Java EE 응용 프로그램의 배치 작업과 동일하다. EAR 배치 작업을 수행함으로써 웹 서비스는 인터넷에서 접근 가능한 서비스로 공개된다.
제22장 JAX-RPC 웹 서비스 호출

본 장에서는 JAX-RPC 웹 서비스를 호출하기 위한 여러 가지 클라이언트를 작성하고 호출하는 방법에 대해 설명한다.

22.1. JAX-RPC 웹 서비스 호출(Java SE 클라이언트)

웹 서비스 호출은 웹 서비스를 사용하기 위한 클라이언트 응용 프로그램이 수행하는 동작을 의미한다. 클라이언트 응용 프로그램은 Java 또는 .NET과 같은 여러 다양한 기술을 이용하여 JEUS에 배치된 웹 서비스를 호출할 수 있다.

웹 서비스 클라이언트는 웹 서비스에 요청하는 프로그램이다. JEUS JAX-RPC 웹 서비스를 사용하여 2가지 타입의 웹 서비스 클라이언트를 생성할 수 있다.

- Stub 클라이언트
  이 타입의 웹 서비스 클라이언트는 웹 서비스의 WSDL로부터 생성된 stub을 이용한다.

- DII(Dynamic Invocation Interface) 클라이언트
  이 타입의 웹 서비스 클라이언트는 JAX-RPC 클라이언트 API를 이용한다.

22.1.1. Stub 클라이언트

Stub 클라이언트는 특정 웹 서비스의 WSDL로부터 생성된 로컬 Stub의 메소드를 호출한다. Stub 객체는 원격의 웹 서비스와 상호작용을 담당한다.

본 절에서는 "제21장 JAX-RPC 웹 서비스 생성과 배치"에서 이미 작성했던 DocLitEchoService 웹 서비스의 메소드를 호출하는 웹 서비스 클라이언트 프로그램을 작성하는 방법을 설명한다.

22.1.1.1. WSDL로부터 웹 서비스 Stub 생성

웹 서비스 Stub 소스 코드를 생성하기 위해서 wsdl2java Ant Task나 wsdl2java 명령어를 사용한다.

Ant Task를 사용하는 경우 build.xml의 코드는 다음과 같다.

[예 22.1] << build.xml >>

```xml
<target name="-pre-compile">
  <antcall target="wsdl2java">
    <param name="wsdl2java.option" value="-gen:client -d ${build.classes.dir}"
```
다음과 같은 명령을 사용하여 웹 서비스 Stub 소스 코드를 생성할 수 있다.

```
wsclient$ ant -pre-compile
```

위의 과정이 성공했다면 웹 서비스 Stub 소스 코드가 생성된다. Stub 소스 코드는 `wsdl2java` Task의 destDir 속성으로 지정한 디렉터리 아래에 컴파일된다.

다음은 생성된 클래스에 대한 설명이다.

<table>
<thead>
<tr>
<th>클래스</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>Echo.java</td>
<td>portType 인터페이스 클래스이다.</td>
</tr>
<tr>
<td>Echo_Stub.java</td>
<td>portType 인터페이스 클래스의 Stub 클래스이다.</td>
</tr>
<tr>
<td>DocLitEchoService.java</td>
<td>서비스 인터페이스 클래스이다.</td>
</tr>
<tr>
<td>DocLitEchoService_Impl.java</td>
<td>서비스 구현 클래스이다.</td>
</tr>
<tr>
<td>*.java</td>
<td>기타 생성 클래스이다.</td>
</tr>
</tbody>
</table>

생성된 Java 파일 또는 Java 파일 안의 메소드의 이름은 웹 서비스의 WSDL로부터 매핑된다. WSDL에 대응하는 Java 이름 매핑은 다음과 같다.

<table>
<thead>
<tr>
<th>WSDL element</th>
<th>매핑</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;service&gt;</td>
<td>서비스 인터페이스와 구현 Java 클래스에 매핑된다.</td>
</tr>
<tr>
<td></td>
<td>&lt;service&gt;의 name 속성값이 인터페이스의 이름이다. 구현 파일은&lt;service&gt;의 이름 뒤에 &quot;Impl&quot;이 붙는다.</td>
</tr>
</tbody>
</table>
### 매핑

<table>
<thead>
<tr>
<th>WSDL element</th>
<th>매핑</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;port&gt;</td>
<td>&lt;service&gt;의 서비스 인터페이스와 구현 클래스 내의 메소드에 매핑된다. 메소드의 이름은 &quot;get + &lt;port&gt;&quot;의 name 속성값으로 이루어진다.</td>
</tr>
<tr>
<td>&lt;portType&gt;</td>
<td>웹 서비스 operation들을 위한 Java 인터페이스와 구현 Java 클래스에 매핑된다. Java 파일 이름은 &lt;portType&gt;의 name 속성을 사용해서 생성된다. 구현 클래스의 Java 파일명은 &lt;portType&gt;의 name 속성값에 '_Stub'가 붙은 형태이다.</td>
</tr>
</tbody>
</table>

### 22.1.1.2. 웹 서비스 클라이언트 작성

본 절에서는 웹 서비스 클라이언트의 작성을 위해서 위에서 생성한 4개의 클래스의 사용방법에 대해서 설명한다.

1. 웹 서비스 객체 생성

    원격 웹 서비스를 위해 서비스 구현 객체를 생성한다. 예제에서 서비스 구현 클래스는 DocLitEchoService_Impl이다.

    다음은 웹 서비스 객체 생성을 위한 Java 코드이다.

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

2. Stub 객체로부터 Port 객체 얻기

    일반 서비스 구현 객체가 생성되었다면 다음 단계로 서비스 구현 객체로부터 Port 객체를 얻는다.

    서비스 인터페이스 클래스는 Port 객체를 얻기 위한 메소드들을 제공한다. 메소드 이름은 "get + WSDL의 <port>의 name 속성값"으로 구성되며, WSDL의 <portType>의 name 속성을 이름으로 가지는 타입이 리턴된다. WSDL Port 이름은 웹 서비스의 WSDL 문서의 <port> 안에 기술되어 있다. <port>는 <service>의 하위 element이다.

    이 예제에서의 WSDL 문서는 다음과 같다.

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

    다음은 서비스 인터페이스 클래스인 DocLitEchoService.java의 일부 내용이다.

    ```java
    <package echo;
    
    public interface DocLitEchoService extends javax.xml.rpc.Service {
    ```
서비스 객체로부터 Port 객체를 얻기 위한 다음의 소스 코드를 구현한다.

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

3. Port 객체상의 Operation 실행

원격 웹 서비스에서 제공하는 Operation들을 위한 Port 객체를 생성하였다면 Port 객체의 메소드들을 호출함으로써 웹 서비스 Operation을 실행할 수 있다. 예제에서 웹 서비스의 Port는 'echoString'이라는 Operation을 제공한다.

Operation을 실행하기 위한 코드는 다음과 같다.

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

또는 다음의 코드로 구현한다.

```java
Echo port = new DocLitEchoService_Impl().getEchoPort();
String s = port.echoString("JEUS");
```

다음은 웹 서비스 클라이언트를 구현한 예제로 wsclient/src/j2se 디렉터리에 위치한다고 가정한다.

[예 22.2] <<ProxyClient.java>>

```java
package j2se;

import echo.DocLitEchoService_Impl;
import echo.Echo;

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

    public void run() {
        try {
            Echo port = new DocLitEchoService_Impl().getEchoPort();
            String s = port.echoString("JEUS");
            System.out.println("response = " + s);
        } catch (Exception e) {
```
22.1.1.3. 웹 서비스 클라이언트 컴파일

ProxyClient 코드의 컴파일을 위해서 wsclient 디렉터리에서 다음의 명령을 콘솔에서 실행한다.

```
JEUS_HOME/samples/webservice/jaxrpc/from_java/doclit/doclit-client$ ant build
```
다음은 컴파일이 성공한 경우의 컴파일된 클래스이다.

```
JEUS_HOME/samples/webservice/jaxrpc/from_java/doclit/doclit-client/build/classes/j2se/ProxyClient.class
```

22.1.1.4. 웹 서비스 클라이언트 실행

웹 서비스 클라이언트가 위치하는 디렉터리로 이동하여 다음의 명령을 실행한다.

```
JEUS_HOME/samples/webservice/jaxrpc/from_java/doclit/doclit-client$ ant run
```
다음은 실행 결과이다.

```
Response = JEUS
```

22.1.2. DII 클라이언트

DII를 사용하면 클라이언트는 실행되기 전까지 Remote Operation의 signature나 웹 서비스의 이름을 몰라도, Remote Operation을 호출할 수 있다.

본 절에서는 DII 클라이언트를 생성하는 방법에 대해서 설명한다.

22.1.2.1. DII 클라이언트 작성

DII 클라이언트를 작성할 때 JAX-RPC 1.1 API를 사용한다. JEUS JAX-RPC 웹 서비스는 JAX-RPC 1.1 API를 완벽히 지원한다. 본 절에서는 JAX-RPC API의 간단한 사용법만을 설명한다.

참고

DII 호출 방법은 RPC 방식의 웹 서비스를 호출하는 데에만 사용될 수 있다. 예제에서는 RPC 방식의 RpcEncEchoService 서비스의 echoString Operation을 호출하는 DII 클라이언트를 작성한다.

DII 클라이언트의 작성 방법은 다음과 같다.

1. 다음의 문장을 DII 클라이언트 코드에 추가한다.

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

2. ServiceFactory와 Service 객체들을 생성한다.

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

3. Call 객체를 생성하고 설정한다.

```java
String operationName = "echoString";
QName QName_XSD_STRING = new QName("http://www.w3.org/2001/XMLSchema", "string");
Call call = (Call) service.createCall();
call.setTargetEndpointAddress(endpoint);
call.setOperationName(new QName(targetNamespace, operationName));
call.addParameter("String_1", QName_XSD_STRING, ParameterMode.IN);
call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY, "http://schemas.xmlsoap.org/soap/encoding/");
call.setReturnType(QNAME_XSD_STRING);
```

4. 최종적으로 Call 객체상에서 웹 서비스 operation을 실행한다.

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

다음은 DII 클라이언트를 구현한 예제로 wsclient/src 디렉터리에 위치한다고 가정한다.

[예 22.3] << DIIClient.java >>

```java
package j2se;
import javax.xml.namespace.QName;
import javax.xml.rpc.Call;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.Service;
import javax.xml.rpc.ServiceFactory;
```
public class DIIClient {
    private static final String NS_XSD =
        "http://www.w3.org/2001/XMLSchema";
    private static final QName QNAME_XSD_STRING = new QName(NS_XSD, "string");

    public void run() {
        try {
            ServiceFactory factory = ServiceFactory.newInstance();
            String endpoint = "http://localhost:8088/" +
                "RpcEncEchoService/RpcEncEchoService";
            String targetNamespace = "urn:RpcEncService";
            Service service = factory.createService(
                new QName(targetNamespace, "RpcEncService"));
            Call call = service.createCall();

            call.setTargetEndpointAddress(endpoint);
            call.setOperationName(
                new QName(targetNamespace, "echoString"));
            call.addParameter("String_1", QNAME_XSD_STRING, ParameterMode.IN);
            call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
            call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY,
                "http://schemas.xmlsoap.org/soap/encoding/");
            call.setReturnType(QNAME_XSD_STRING);

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

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

DII 클라이언트 코드의 컴파일을 위해서 wsclient 디렉터리로 이동하여 다음의 명령을 콘솔에서 실행한다.

```
$ JEU$HOME/samples/webservice/jaxrpc/from_java/rpcenc/rpcenc-client$ ant build
```

컴파일이 성공한 경우 컴파일된 클래스는 다음과 같다.

```
$ JEU$HOME/samples/webservice/jaxrpc/from_java/rpcenc/rpcenc-client/build/classes/j2se/DIIClIENT.class
```

### 22.1.2.3. DII 클라이언트 실행

DII 클라이언트가 위치하는 디렉터리로 이동하여 다음의 명령을 실행한다.

```
$ JEU$HOME/samples/webservice/jaxrpc/from_java/rpcenc/rpcenc-client$ ant run
```

다음은 실행결과이다.

```
Response = JEU
```

---

### 22.2. JAX-RPC 웹 서비스의 호출(Java EE 클라이언트)

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

#### 22.2.1. Java EE 클라이언트 프로그래밍 모델

웹 서비스는 클라이언트의 입장에서 본다면 클라이언트를 대신하여 비즈니스 로직을 수행하는 메소드의 집합이라 할 수 있다. 클라이언트는 메소드가 로컬에서 수행되는지 원격에서 수행하는지를 구별할 수 없다. 클라이언트는 JAX-RPC 스펙에 정의된 SEI를 사용하여 웹 서비스에 접근한다.

Java EE 클라이언트는 JAX-RPC에 정의된 서비스 인터페이스를 구현한 서비스 객체에 접근하기 위해서 JNDI를 사용한다. 서비스 객체는 클라이언트가 SEI를 구현한 Stub이나 프록시를 얻어오기 위해 사용하는 Factory이다. 서비스 인터페이스는 JAX-RPC에 정의된 javax.xml.rpc.Service 인터페이스이나, 이를 상속하여 생성된 서비스 인터페이스이다.

클라이언트 개발자는 SEI와 서비스 인터페이스를 얻어오는 것으로부터 시작하며 이들은 JAX-RPC에 정의된 WSDL → Java 매핑 범주에 의해 생성된다. 클라이언트 개발자는 Stub을 개발 시점에 생성하지 않을 것을 권장하며, 개발 시점에는 Stub이 아닌 인터페이스를 사용할 수 있다. Stub은 클라이언트 모듈을 배치하는 시점에 클라이언트가 구동되는 환경에 맞게 자동으로 생성될 것이다. 웹 서비스의 JNDI-lookup은 논리적인 이름에 의해 이루어지며, 이 이름은 클라이언트 DD에 정의된다.
22.2.2. Java EE 클라이언트 프로그래밍 절차

Java EE 클라이언트 프로그래밍의 기본적인 절차는 다음과 같다.

1. JNDI Lookup을 통한 서비스(인터페이스)의 획득한다.
   클라이언트 개발자는 서비스의 레퍼런스로 사용되는 논리적인 서비스의 JNDI 이름을 클라이언트의 DD에 정의해야 한다. 필수 요구 사항은 아니지만 모든 서비스의 논리적인 레퍼런스 이름을 JNDI Name Space의 service라는 서브 컨텍스트 아래에 구성하는 것을 권장한다.
   컨테이너는 서비스 인터페이스의 구현을 클라이언트 환경 컨텍스트(java:comp/env) 하위에 서비스 레퍼런스의 논리적인 이름으로 바인딩시켜야 한다.
   ```java
   InitialContext jndiContext = new InitialContext();
   Service service = (Service) jndiContext.lookup("java:comp/env/service/DocLitEchoService");
   ```
   위의 예제에서는 서비스 인터페이스 javax.xml.rpc.Service가 web.xml의 <service-ref> 하위 element인 <service-interface>에 설정되어 있고, JNDI 이름은 <service-ref-name>에 설정되어 서로 바인딩된다.

   참고
   JAX-RPC에서는 ServiceFactory라는 클래스에서 서비스를 생성할 수 있다. 하지만, Java EE 응용 프로그램에서는 ServiceFactory를 사용하는 것을 권장하지 않는다. Java EE 클라이언트는 항상 JNDI 서비스 Lookup을 통해 서비스를 가져와야 한다.

2. 서비스 인터페이스의 API를 이용한 Stub 생성 또는 Call 객체를 생성한다.
   서비스 인터페이스는 클라이언트가 서비스 포트에 접근하기 위해서 Stub과 동적 프록시 혹은 DII Call 객체를 생성하려고 할 때 사용된다. 클라이언트 개발자는 응용 프로그램이 사용하는 서비스 인터페이스의 타입을 명시적으로 클라이언트 DD(web.xml)에 선언해야 한다.
   다음은 서비스 인터페이스의 타입과 JNDI 이름이 설정된 web.xml의 예이다.

   [예 22.4] << web.xml >>

   ```xml
   <web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
      <servlet>
         <servlet-name>jsp_helloClient</servlet-name>
         <jsp-file>/helloClient.jsp</jsp-file>
         <load-on-startup>0</load-on-startup>
      </servlet>
      <service-ref>
         <service-ref-name>
            service/DocLitEchoService
         </service-ref-name>
      </service-ref>
   </web-app>
   ```
3. Stub이나 Call 객체를 이용한 웹 서비스를 호출한다.

- **Stub** 객체를 이용한 호출
  
  클라이언트는 서비스 Lookup을 통하여 가져온 서비스 인터페이스의 다음과 같은 함수들을 이용해서 Stub과 동적인 프록시를 생성할 수 있다.

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

- **Call** 객체를 이용한 호출
  
  클라이언트는 JNDI Lookup을 통하여 얻어온 서비스 인터페이스의 DII 메소드를 사용하여 Call 객체를 가져올 수 있다.

  다음은 동적 포트에 접근하기 위한 API이다.

  ```java
  Call createCall() throws ServiceException;
  Call createCall(QName portName) throws ServiceException;
  Call createCall(QName portName, String operationName)
  throws ServiceException;
  Call createCall(QName portName, QName operationName)
  throws ServiceException;
  ```

### 22.2.3. Java EE 클라이언트의 작성과 예제

Java EE 클라이언트는 WSDL을 사용하거나 사용하지 않고도 작성할 수 있다. 클라이언트는 JSP, 서블릿, EJB 등 여러 형태일 수 있다. 단, 앞에서 제시한 프로그래밍 모델에 부합하기만 하면 된다.

본 절에서는 간단한 JSP 형태의 Java EE 클라이언트 작성 예제를 통해 작성법을 설명한다.
22.2.3.1. WSDL을 사용한 서비스 호출

WSDL을 사용한 서비스를 호출방법은 다음과의 절차를 따른다.

1. JEUS는 wsdl2java Ant Task와 wsdl2java Command Line 툴을 사용해서 Java EE 클라이언트 Portable Artifact와 JAX-RPC 매핑 파일을 생성한다. wsdl2java Task에 대한 보다 자세한 설명은 "JEUS Reference Book"의 "5.5.2. wsdl2java"를 참고한다.

Ant Task를 사용하는 build.xml의 예는 다음과 같다.

[예 22.5] << build.xml >>

```xml
<target name="do-package-war">
  ...

  <antcall target="wsdl2java">
    <param name="wsdl2java.option" value="-import:client
    -d ${build.war.dir}/WEB-INF/classes -package echo
    -outputmapping ${build.war.dir}/WEB-INF/DocLitEchoService-mapping.xml
    -compile ${src.web}/WEB-INF/wsdl/DocLitEchoService.wsdl"/>
  </antcall>
  ...

</target>
```

ant build 명령을 수행하고 나면, Java EE 클라이언트 Portable Artifact와 JAX-RPC 매핑 파일이 생성된다.

2. 클라이언트 프로그램을 작성한다.

WSDL을 가지고 클라이언트를 개발하는 경우에는 다음과 같은 java.xml.rpc.Service 인터페이스 메소드를 이용하여 Stub이나 Call 객체들을 가져올 수 있다.

```java
java.rmi.Remote getPort(QName portName, Class serviceEndpointInterface)
  throws ServiceException;
java.rmi.Remote getPort(java.lang.Class serviceEndpointInterface)
  throws ServiceException;
Call createCall() throws ServiceException;
Call createCall(QName portName) throws ServiceException;
Call createCall(QName portName, String operationName)
  throws ServiceException;
Call createCall(QName portName, QName operationName)
  throws ServiceException;
Call[] getcalls(QName portName) throws ServiceException;
```
다음은 위의 메소드 중 동적인 프록시(Dynamic Proxy)를 생성하여 웹 서비스를 호출하도록 JSP로 작성된 Java EE 클라이언트의 예이다.

[예 22.6] << helloClient.jsp >>

```jsp
<%@ page language="java" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="java.util.*" %>
<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.xml.rpc.Service" %>
<%@ page import="echo.*" %>
<%@ page errorPage="/error.html" %>

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

<%! String msgToSend = "msg_sent_by_jspClient";
    String ret=null;
    String exceptionString=""; %>
<%
    try {
        InitialContext jndiContext = new InitialContext();
        Service service = (Service)jndiContext.lookup("java:comp/env/service/DocLitEchoService");
        java.rmi.Remote port = service.getPort(Echo.class);
        Echo echoPort = (Echo)port;
        ret = echoPort.echoString(msgToSend);
    } catch (Exception e) {
        exceptionString = e.toString();
        e.printStackTrace();
    }
%>
<%= "Result is "+ret+"....." %>
```

3. 앞서 제시한 예제에 대한 표준 DD(web.xml) 파일을 작성한다.

[예 22.7] << web.xml >>

```xml
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <servlet>
        <servlet-name>jsp_helloClient</servlet-name>
        <jsp-file>/helloClient.jsp</jsp-file>
        <load-on-startup>0</load-on-startup>
    </servlet>
</web-app>
```
web.xml의 <service-ref-name>에 JSP 클라이언트에서 JNDI Lookup으로 찾아오는 서비스 이름을 설정한다.


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

[예 22.8] << jeus-web-dd.xml >>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jeus-web-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <service-ref>
    <service-ref-name>
      service/DocLitEchoService
    </service-ref-name>
    <service-interface>
      javax.xml.rpc.Service
    </service-interface>
    <wsdl-file>
      WEB-INF/wSDL/DocLitEchoService.wsdl
    </wsdl-file>
    <jaxrpc-mapping-file>
      WEB-INF/DocLitEchoService-mapping.xml
    </jaxrpc-mapping-file>
    <port-component-ref>
      <service-endpoint-interface>
        echo.Echo
      </service-endpoint-interface>
    </port-component-ref>
  </service-ref>
</jeus-web-dd>
```
22.2.3.2. WSDL을 사용하지 않고 서비스 호출

WS-I라는 표준 기구는 웹 서비스의 상호 호환성을 위해 Basic Profile을 정의하고 이를 준수할 것을 요구하고 있다. 현재 나와있는 Basic Profile 1.0은 WSDL이 항상 접근 가능하게 공개되기를 요구한다.

Basic Profile을 준수하지 않는 서비스에 접근하려면 WSDL이 항상 공개되어 있다고 볼 수 없다. 이 경우 WSDL과 무관하게 웹 서비스를 호출할 수 있는데, DII이 바로 그 방법이다. 하지만 이는 결국 WSDL에 포함된 오퍼레이션 스타일이나 오퍼레이션 이름과 같은 상세를 Call 객체에 제공해야만 가능하다.

WSDL이 클라이언트 DD에 정의되어 있지 않다면 클라이언트 개발자는 다음과 같은 서비스 인터페이스의 메소드를 통해서 Call 객체를 생성해야 한다.

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

다음은 Call 객체를 이용하여 웹 서비스를 호출하는 JSP Java EE 클라이언트의 예이다.

[예 22.9] << helloClient.jsp >>

```jsp
<%@ page language="java" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="java.util.*" %>
<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.xml.rpc.Service" %>
<%@ page import="javax.xml.rpc.Call" %>
<%@ page import="javax.xml.namespace.QName" %>
<%@ page import="javax.xml.rpc.ParameterMode" %>

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

<%! String msgToSend = "msg_sent_by_jspClient"; String ret=null; String exceptionString=""; %>
```
위 예제에서 제시한대로 Call 객체를 생성한 후 오퍼레이션 이름과 오퍼레이션이 갖는 파라미터들을 추가하게 되며, 이때 추가적으로 오퍼레이션의 스타일과 인코딩 방식을 설정할 수도 있다.

설정 방법은 다음과 같다.

```java
call.setProperty(Call.OPERATION_STYLE_PROPERTY, "rpc");
call.setProperty(Call.ENCODINGSTYLE_URI_PROPERTY, "http://schemas.xmlsoap.org/soap/encoding/");
```

이전에 설명한 예제의 표준 DD(web.xml)는 다음과 같이 구성되어 있다.

[예 22.10] << web.xml >>

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


22.2.3.3. 웹 서비스 클라이언트의 패키징

Java EE 웹 서비스 클라이언트는 하나의 Java EE 컴포넌트가기 때문에 컴포넌트 고유의 패키징 방식을 따른다.

다음은 JSP 클라이언트의 패키징 방식에 대한 설명이다.

```
WAR
  |-- JSP Client
  |   |-- WEB-INF
  |       |-- web.xml (서블릿 DD)
  |       |-- jeus-web-dd.xml (JEUS Specific DD)
  |       |-- Jax-rpc mapping 파일
  |       |-- wsdl
  |       |   |-- wsdl 파일
  |       |-- classes
  |           |-- Java 클래스 컴포넌트 (SEI, Service Interface..)
```

위와 같이 패키징 한 Java EE 클라이언트를 배치하고 JSP를 호출하면 웹 서비스를 호출할 수 있게 된다. JAX-RPC 매핑 파일과 WSDL 파일이 webservices.xml 파일에 <wsdl-file>과 <jax-rpc-mapping-file>로 기술한 위치에 존재한다면 반드시 위의 구조를 따르지 않아도 무방하다.
본 장에서는 SAAJ 프로그래밍 모델과 JAX-RPC와 연동하여 사용하는 메시지 핸들러에 대해서 설명한다.

23.1. SAAJ 사용

SAAJ(SOAP with Attachments API for Java)를 사용하면 SOAP 메시지를 직접 생성하고 읽고 다루는 작업을 할 수 있다. 여기에서는 SAAJ의 프로그래밍 모델에 대해서 간략히 언급하고 API 등에 관련된 자세한 내용은 여기에서 다루지 않는다. 보다 자세한 내용을 필요로 할 경우에는 SAAJ 스펙을 참고한다.

SAAJ는 Attachment가 없는 간단한 XML과도 같은 단순한 SOAP 메시지에서부터 MIME Attachment를 가지는 더 복잡한 SOAP 메시지를 다루는 작업까지 가능하다. SAAJ는 Abstract Factory 패턴을 기본으로 하며 MessageFactory에서 메시지(SOAPMessage)를 생성한다. SOAPMessage는 SOAP 문서를 나타내는 SOAPPart와 MIME Attachment를 나타내는 0개 이상의 AttachmentPart 객체를 포함한다.

[그림 23.1] Structure of SOAP with Attachment

23.1.1. SOAP 메시지의 생성

비어있는 SOAP 메시지를 생성하기 위해서는 MessageFactory 객체로부터 SOAPMessage 객체를 새로 생성해야 한다.

다음은 SOAPMessage 객체를 생성하는 소스의 예이다.

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

23.1.2. SAAJ 문서 다루기

SAAJ는 SOAP 문서를 생성하고 다루기 위해 사용할 수 있는 많은 인터페이스를 제공한다.

SOAP 문서는 element와 속성으로 구성된 XML 인스턴스이다. 편의를 위해 SOAP 문서의 주요한 부분들은 SAAJ와 대응하는 타입을 갖는다. envelope는 SOAPEnvelope, Header는 SOAPHeader, Body는 SOAPBody와 대응한다. SOAPElement 타입은 SOAP namespace에 속하지 않는 응용 프로그램에 종속적인(application-specific) element와 대응한다.

- SOAPPart SOAPEnvelope 타입

SwA(SOAP with Attachment) 메시지를 가지고 작업을 하면 SOAPPart와 SOAPEnvelope 타입을 자주 사용하게 된다. SOAPPart는 SwA 메시지의 MIME 부분의 가장 상위이며, SOAPMessage.getSOAPPart() 메소드를 호출하여 접근 가능하다.

SOAPEnvelope에 대한 접근은 SOAPPart의 getEnvelope() 메소드를 호출하여 가능하다. SOAPEnvelope는 XML SOAP 문서의 최상위이며, SOAPHeader와 SOAPBody에 대한 접근과 생성을 위한 메소드를 포함한다.

- SOAPElement 타입

SOAPElement 타입은 SOAP 1.1 또는 1.2 XML의 Namespace로 표현되지 않는 응용 프로그램에 종속적인(application-specific) element를 직접적으로 표현하고자 할 때 사용한다. 이 타입은 XML element를 나타낼 수 있다.

XML element가 다른 XML element를 포함할 수 있듯이 SOAPElement는 다른 SOAPElement 객체를 포함할 수 있다. 이 타입은 다른 SOAP 타입(SOAPEnvelope, SOAPBody, SOAPBodyElement, SOAPHeader, SOAPHeaderElement, fault element)들의 상위 타입(super-type)이다.

- SOAPHeader 타입

SOAP 메시지의 Header element는 0 또는 그 이상의 Header 블록을 가질 수 있다.

SOAPHeader 타입은 Header element를 나타내며, SOAPHeaderElement 타입은 SOAP 메시지에 있어서 각각의 Header 블록을 나타낸다. SOAPHeader 타입은 SOAPHeaderElement 객체를 추가, 삭제, 검사하는 메소드를 제공한다.

- SOAPHeaderElement 타입

SOAPHeaderElement는 특정한 Header 블록의 속성과 자식 element를 변경하거나 검사할 수 있도록 Header 블록을 추상화한 모델이다.
SOAPHeaderElement 태입은 actor와 mustUnderstand같은 속성뿐 아니라 하나 또는 그 이상의 SOAPElement 객체를 포함할 수도 있다.

- SOAPBody 태입과 SOAPBodyElement 태입

SOAPBody 태입은 SOAP 메시지의 Body element를 나타낸다. SOAPBodyElement는 SOAPElement 를 확장하며, 상속하는 메소드 이외의 추가적인 메소드는 없다.

다음은 SOAPBody 태입과 SOAPBodyElement 태입의 활용 예이다.

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

23.1.3. SAAJ를 이용한 SOAP 메시지 전송

SAAJ를 이용한 SOAP 메시지 생성 작업을 마쳤다면 전송 작업을 수행할 수 있다.

SAAJ는 간단하게 구성된 독자적인 메시지 전송 시스템을 가지고 있으며, 기본 API의 한 부분으로 구성된다. SAAJ를 사용함으로써 요청 또는 응답 스타일의 SOAP 메시지를 웹 서비스와 교환할 수 있다.

다음은 SOAPConnection을 생성해서 메시지를 전송하도록 구현한 예이다.

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

String address = "...";

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

23.2. SOAP 메시지 핸들러의 생성

메시지 핸들러는 JAX-RPC 클라이언트와 웹 서비스 Endpoint에 의해 송수신되는 SOAP 메시지를 직접 다룰수 있게 하며, 정적으로 생성된 Stub, 동적으로 생성된 프록시, DII, Java 클래스 Endpoint, EJB Endpoint와 함께 사용될 수 있다.

메시지 핸들러의 가장 중요한 목적은 JAX-RPC 클라이언트와 웹 서비스 Endpoint가 송수신하는 SOAP 메시지의 Header 블록을 다루고, 읽고, 다루는 메커니즘을 제공하는 것이다.
메시지 핸들러는 Java EE 컨테이너에 의해서 다루어진다. JAX-RPC 클라이언트 API를 사용하여 SOAP 메시지를 전송하려 할 때는 JAX-RPC 콘테이너이 웹 서비스로 연결된 네트워크에 내보내기 전에 메시지 핸들러 체인을 통해 SOAP 메시지를 가공하게 된다. 마찬가지로 SOAP 응답 메시지가 JAX-RPC 클라이언트에 의해 수신될 때에는 클라이언트 응용 프로그램으로 결과가 리턴되기 전에 이전에 거쳐 나갔던 동일한 메시지 핸들러 연결고리를 거쳐서 가공된다.

이러한 핸들러는 여러 가지로 사용할 수 있다. 그 중에서 보안에 관련된 경우를 살펴보면 클라이언트에서 암호화된 SOAP 메시지를 송신하는 경우 암호화 작업을 위한 핸들러를 사용해서 메시지를 가공하여 송신할 수 있고, 웹 서비스는 그 요청 메시지를 암호 해독을 위한 핸들러를 이용하여 해독한 다음 그 데이터를 웹 서비스를 구현한 Back-end로 보내게 된다. 이에 대한 SOAP 메시지 응답은 이 과정을 역으로 수행하게 된다.

또 다른 예는 SOAP 메시지의 Header 부분에 저장된 정보에 접근하는 경우이다. SOAP Header에 웹 서비스의 특정한 정보를 저장할 수 있고, 핸들러로 하여금 그 정보를 다룰 수 있게 할 수 있다.

### 23.2.1. 메시지 핸들러의 생성

SOAP 메시지 핸들리는 웹 서비스의 요청과 응답에서 모두 SOAP 메시지를 가로채서 가공할 수 있다. 또한 웹 서비스 Endpoint와 웹 서비스를 호출하는 클라이언트에서 모든 핸들러를 생성할 수 있다.

다음은 javax.xml.rpc.handler API의 주요 클래스와 인터페이스에 대한 설명이다.

<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.rpc.handler. Handler</td>
<td>SOAP 요청 메시지와 응답 메시지, fault 메시지를 다루는 메소드를 포함하는 주요 인터페이스이다.</td>
</tr>
<tr>
<td>javax.xml.rpc.handler. HandlerRegistry</td>
<td>프로그래밍 레벨에서 핸들러의 설정을 조정할 수 있도록 지원하는 인터페이스이다.</td>
</tr>
<tr>
<td>javax.xml.rpc.handler. HandlerInfo</td>
<td>webservises.xml에 정의되는 핸들러의 초기 파라미터와 같은 핸들러의 정보들을 포함하는 클래스이다.</td>
</tr>
<tr>
<td>javax.xml.rpc.handler. MessageContext</td>
<td>핸들러에 의해서 처리되는 메시지의 내용을 추상화한 인터페이스로, 핸들러 체인 내에서 핸들러는 메시지 처리에 관계되는 상태 정보를 공유할 수 있다.</td>
</tr>
<tr>
<td>javax.xml.rpc.handler.soap.SOAPMessageContext</td>
<td>SOAP 요청과 응답 메시지에 접근할 수 있는 메소드를 제공하는 MessageContext의 하위 인터페이스이다.</td>
</tr>
<tr>
<td>javax.xml.rpc.handler. GenericHandler</td>
<td>핸들러 인터페이스를 구현한 추상클래스로서 SOAP 메시지 핸들러 개발자는 이 클래스를 상속받아 구현하면 된다.</td>
</tr>
<tr>
<td>javax.xml.soap.SOAPMessage</td>
<td>SOAP 메시지의 요청과 응답을 포함하는 클래스이다.</td>
</tr>
</tbody>
</table>

다음은 메시지 핸들러와 핸들러 체인을 추가함으로써 웹 서비스를 갱신하는 절차이다.
1. 메시지 핸들러와 핸들러 체인의 설계
2. javax.xml.rpc.handler.Handler 인터페이스를 구현하는 Java 클래스의 생성 및 핸들러 체인의 구현
3. Java 코드의 컴파일
4. 웹 서비스 DD(webservices.xml)의 작성
5. 웹 서비스 패키징과 배치

웹 서비스 클라이언트에서도 SOAP 메시지 핸들러를 사용할 수 있으며 이는 본 장의 뒷부분에 언급된다.

23.2.2. 메시지 핸들러와 핸들러 체인의 설계

SOAP 메시지 핸들러를 설계하려면 다음을 명확하게 해야 한다.
- 추가할 메시지 핸들러의 개수
- 추가할 핸들러의 실행 순서
- 메시지 핸들러로만 웹 서비스를 구성할 것인가?(Back-end 구성요소들을 호출하지 않을 것인가?)

핸들러들은 webservices.xml 안에서 설정해야 하며, 실행 순서가 정해진 핸들러의 묶음을 핸들러 체인 (chain)이라고 한다. 핸들러 체인으로 구성된 각각의 핸들러는 요청 SOAP 메시지를 다루기 위해 메소드 와 응답 SOAP 메시지를 다루기 위한 또 다른 메소드가 있다.

웹 서비스를 호출하면 JEOS 웹 서비스는 핸들러를 다음과 같은 메커니즘으로 실행한다.
1. webservices.xml에 정해진 순서대로 핸들러 체인 안의 각각의 핸들러의 handleRequest() 메소드가 실행된다.
2. 핸들러 체인의 마지막 핸들러의 handleRequest()가 실행되면 JEOS 웹 서비스는 웹 서비스 Back-end 를 호출한다(Back-end가 존재하는 경우).
4. webservices.xml에 정의된 첫 번째 핸들러의 handleResponse()가 실행되면 웹 서비스를 호출한 클라이언트 쪽으로 SOAP 메시지 응답을 보낸다.

23.2.3. 핸들러 인터페이스의 구현


메시지 핸들러 클래스는 핸들러 인터페이스의 다음과 같은 함수들을 구현해야 한다.
<table>
<thead>
<tr>
<th>구분</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>init()</td>
<td>HandlerInfo 객체는 webservices.xml에 정의된 초기화 정보 같은 SOAP 메시지 핸들러에 대한 정보를 포함하고 있다. HandlerInfo.getHandlerConfig() 메소드를 실행하시면 name-value 형태의 맵(Map)을 리턴한다. 핸들러의 초기화 작업을 진행하려면 init() 메소드를 구현해야 한다.</td>
</tr>
<tr>
<td>destroy()</td>
<td>Handler.destroy() 메소드는 핸들러 객체의 인스턴스를 제거하는 경우 불리며, 핸들러의 생성 주 기 동안 획득된 자원을 다시 넣어야 하려면 이 함수를 구현해야 한다.</td>
</tr>
<tr>
<td>getHeaders()</td>
<td>핸들러 인스턴스에 의해 처리되는 header 블록을 가져오는 경우 사용한다. getHeaders() 메소드를 실행시킨 후 MessageContext 객체는 SOAP 메시지의 내용을 가지고 있으며, MessageContext의 서브 인터페이스인 SOAPMessageContext를 사용하면 SOAP 요청 메시지의 내용을 얻거나 변경할 수 있다.</td>
</tr>
</tbody>
</table>
23.2.4. Java EE 웹 서비스 DD 파일 작성

Java EE 웹 서비스 DD 파일은 webservices.xml로, webservices.xml 파일에는 SOAP 메시지 핸들러에 대한 정보, 그리고 핸들러의 처리 순서를 정의할 수 있다.
다음은 메시지 핸들러 정보를 설정하는 예이다.

[예 23.1] 메시지 핸들러를 정의하는 <port-component>의 예

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

23.2.5. 클라이언트에서 SOAP 메시지 핸들러의 사용

이제까진 JEOS 서버에서 동작하여 웹 서비스로 실행되는 핸들러를 생성하는 것에 대해 설명했지만, 클라이언트 응용 프로그램에서도 핸들러를 생성하는 것이 가능하다.
클라이언트의 핸들러를 생성하는 방법은 서버의 핸들러를 생성하는 방법과 같다. javax.xml.rpc.handler.Handler 인터페이스를 구현하는 Java 클래스를 작성한다.

23.2.6. 파일 송수신하는 웹 서비스와 클라이언트 예제

본 절에서는 클라이언트에서 File_Send.txt라는 파일을 서버로 전송하고 서버에서는 이 파일을 지정된 폴더에 저장하고 파일을 받았다는 응답을 전달하는 웹 서비스 파일 송수신 예제이다.
클라이언트는 다음의 과정으로 생성한다.
1. 파일 수신하는 메시지 핸들러 구현
2. 파일 수신 후 파일을 받았다는 메시지를 전달하는 서비스 Back-end 구현
3. 웹 서비스 배치서울자의 작성이 및 웹 서비스 생성과 배치
4. 웹 서비스 클라이언트 핸들러 작성

5. 웹 서비스 클라이언트 작성

6. 실행

23.2.6.1. 파일 수신 메시지 핸들러의 구현

다음은 웹 서비스의 메시지 핸들러 구현으로서 `handleRequest()`가 Back-end로 SOAP 요청을 하기 전에 먼저 실행된다. `handleRequest()` 함수 안에서는 SOAP 메시지에 `Attachment`로 붙어있는 파일을 SAAJ API를 이용하여 다루게 된다.

[예 23.2] << ServerAttachmentHandler.java >>

```java
package filetransfer;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Iterator;
import java.util.Map;
import javax.xml.namespace.QName;
import javax.xml.rpc.JAXRPCException;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.MessageContext;
import javax.xml.rpc.handler.soap.SOAPMessageContext;
import javax.xml.soap.AttachmentPart;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPException;
import javax.xml.soap.SOAPMessage;
import javax.xml.rpc.handler.GenericHandler;

public final class ServerAttachmentHandler
    extends GenericHandler
{
    private File dir;

    private final String DIR_PROP = "directory";

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

        String dirName = (String) m.get(DIR_PROP);
```
if (dirName == null) {
    throw new JAXRPCException("Property named: " + DIR_PROP + " was not found");
}  
dir = new File(dirName);

if (! dir.exists()) {
    if (! dir.mkdirs()) {
        throw new JAXRPCException("Unable to create directory: " + dirName);
    }
}
if (! dir.canWrite()) {
    throw new JAXRPCException("Don't have write permission for " + dirName);
}

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

private void copyFile(InputStream is, OutputStream os) throws IOException {
    byte[] b = new byte[8192];
    int nr;
    while ((nr = is.read(b)) != -1) {
        os.write(b, 0, nr);
    }
}

public boolean handleRequest(MessageContext mc) {
    SOAPMessageContext ctx = (SOAPMessageContext) mc;
    SOAPMessage request = ctx.getMessage();
    if (request.countAttachments() == 0) {
        throw new JAXRPCException("** Expected attachments");
    }
    try {
        Iterator it = request.getAttachments();
    }
while(it.hasNext()) {
    AttachmentPart part = (AttachmentPart) it.next();
    String fileName = getFileName(request);
    System.out.println("Received file named: " + fileName);

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

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

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

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

23.2.6.2. 웹 서비스 Back-end 구현

다음은 웹 서비스 Back-end 구현으로서 메시지 헤더에서 파일을 받는 작업이 수행되고 난 다음 실행이 되며, 클라이언트로 파일을 받았다는 응답을 전달하는 클래스이다.
다음은 SEI를 구현한 예이다.

```java
package filetransfer;

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

다음은 웹 서비스 DD 파일(webservices.xml)이다.

```xml
<?xml version="1.0"?>
<webservices version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee">
    <webservice-description>
        <webservice-description-name>FileAttachmentService</webservice-description-name>
        <wsdl-file>
            WEB-INF/wsdl/FileAttachmentService.wsdl
        </wsdl-file>
        <jaxrpc-mapping-file>
            WEB-INF/FileAttachmentService-mapping.xml
        </jaxrpc-mapping-file>
    </port-component>
</webservice-description>
</webservices>
```
위와 같은 작업이 완료되면 web.xml과 jeus-webservices-dd.xml을 다음과 같이 작성한다.

[예 23.6] << web.xml >>

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
    <servlet>
        <servlet-name>FileAttachmentServlet</servlet-name>
        <servlet-class>filetransfer.FileTransfer</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>FileAttachmentServlet</servlet-name>
        <url-pattern>/FileAttachmentService</url-pattern>
    </servlet-mapping>
</web-app>
```

[예 23.7] << jeus-webservices-dd.xml >>

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <service>
        <webservice-description-name>FileAttachmentService</webservice-description-name>
    <port>
위와 같이 DD 파일의 작성이 완료되면 EAR로 패키징하여 JEUS에 배치 작업을 한다.
이 서비스에 접근할 수 있는 주소는 다음과 같다.

http://localhost:8088/FileAttachmentService/FileAttachmentService

### 23.2.6.4. 웹 서비스 클라이언트 핸들러 작성

클라이언트 핸들러에서는 handleRequest()에서 메시지 컨텍스트를 직접 다루며, SAAJ를 통하여 SOAP 메시지에 Attachment 형태로 파일을 첨부하게 된다.

![예 23.8] << ClientAttachmentHandler.java >>

```java
package filetransfer;

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

public final class ClientAttachmentHandler
    extends GenericHandler {

    private String getFileName(SOAPMessage request)
        throws SOAPException {
        SOAPBody body = request.getSOAPPart().getEnvelope().getBody();
        SOAPElement opElem = (SOAPElement)body.getChildElements().next();
        SOAPElement paramElem = (SOAPElement)opElem.getChildElements().next();
        return paramElem.getValue();
    }
```
public boolean handleRequest(MessageContext mc) {

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

    try {
        String fileName = getFileName(request);
        AttachmentPart part = request.createAttachmentPart();
        part.setContentType("application/x-zip-compressed");
        FileDataSource fds = new FileDataSource(fileName);
        part.setDataHandler(new DataHandler(fds));
        request.addAttachmentPart(part);
    } catch(SOAPException e) {
        e.printStackTrace();
        throw new JAXRPCException(e);
    }
    return true;
}

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

23.2.6.5. 웹 서비스 클라이언트 작성

웹 서비스 클라이언트의 작성에서 주목할 것은 웹 서비스 클라이언트 핸들러를 핸들러 레지스트리에 등록하는 것이다. 파일 송신에 관한 구현은 클라이언트 핸들러에서 이루어진다.

[예 23.9] << Client.java >>

package filetransfer;

import java.io.File;
import java.util.ArrayList;
import java.util.List;
import javax.xml.namespace.QName;
import javax.xml.rpc.Service;
import javax.xml.rpc.Stub;
import javax.xml.rpc.handler.HandlerInfo;
import javax.xml.rpc.handler.HandlerRegistry;
import FileAttachmentService_pkg.*;
public final class Client {

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

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

        QName portName = new QName("urn:FileAttachmentService","FileTransferIFPort");

        registry.setHandlerChain(portName, list);

        FileTransferIF port =
            ((FileAttachmentService_Impl)svc).getFileTransferIFPort();

        String result = port.receiveFile("File_send.txt");
        System.out.println("** File transfer result: "+result);
    }
}

23.2.6.6. 실행

위와 같이 구현한 웹 서비스를 클라이언트에서 실행하기 위해서는 클라이언트의 Stub 생성이 필요하다.

JEUS_HOME/samples/webservice/jaxrpc_saaaj_msg_handler/fileAttachment/fileAttachment-c
lient$ ant build

Stub이 생성되었다면 다음과 같이 실행한다.

JEUS_HOME/samples/webservice/jaxrpc_saaaj_msg_handler/fileAttachment/fileAttachment-c
lient$ ant run

결과와 같은 결과가 출력된다.

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

위와 같은 결과가 출력되었다면 클라이언트에서 File_send.txt라는 파일이 서버로 전달되었고, webser
vices.xml에 설정한 대로 'temp' 폴더에 File_send.txt 파일이 저장되어 있음을 확인할 수 있다.

제23장 JAX-RPC 웹 서비스 SOAP 메시지 핸들러 생성 287
제24장 JAX-RPC 웹 서비스 설정 파일 작성

본 장에서는 JAX-RPC 웹 서비스의 표준 웹 서비스 DD 파일 작성 방법과 JAX-RPC 매핑 파일 작성 방법에 대해 설명한다.

24.1. JAX-RPC 웹 서비스 DD 파일 작성

Java EE JAX-RPC 웹 서비스는 webservices.xml이라는 이름을 가진 웹 서비스 서술자를 Java 클래스 Endpoint나 EJB Endpoint를 포함하는 압축 파일(WAR나 JAR)에 포함할 것을 요구한다.

webservices.xml을 EJB Endpoint의 경우에는 EJB JAR 파일의 META-INF 디렉터리에 저장하고, 서블릿 Endpoint의 경우에는 WAR 파일의 WEB-INF 디렉터리에 위치시킨다.

d음은 FileAttachmentService라는 웹 서비스의 webservices.xml 파일이다.

[예 24.1] << webservices.xml >>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<webservices version="1.1" xmlns="http://java.sun.com/xml/ns/j2ee">
  <webservice-description>
    <webservice-description-name>FileAttachmentService</webservice-description-name>
    <wsdl-file>WEB-INF/wsdl/FileAttachmentService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>WEB-INF/FileAttachmentService-mapping.xml</jaxrpc-mapping-file>
  </webservice-description>
  <port-component>
    <port-component-name>FileAttPort</port-component-name>
    <wsdl-port xmlns:ns2="urn:FileAttachmentService">
      <ns2:FileTransferIFPort/>
    </wsdl-port>
    <service-endpoint-interface>
      <filetransfer.FileTransferIF/>
    </service-endpoint-interface>
    <service-impl-bean>
      <servlet-link>FileAttachmentServlet</servlet-link>
    </service-impl-bean>
    <handler>
      <handler-name>ServerAttachmentHandler</handler-name>
    </handler>
  </port-component>
</webservices>
```
webservices.xml 파일의 root element는 <webservices>이고, <webservice-description>을 하나 또는 그 이상을 필수 요소로 가진다.

<webservice-description>은 동일한 WSDL을 사용하는 Java 클래스나 EJB Endpoint의 집합을 서술한다. 즉, 패키징 내의 다른 WSDL 각각의 파일 들에 대해서 <webservice-description>이 하나씩 존재해야 한다. 예를 들면 2개의 다른 Java 클래스 Endpoint가 각각 WSDL을 별도로 가지고 하나의 WAR 파일 안에 존재 한다면 각각의 JSE를 서술하기 위해서 <webservice-description>은 2개가 존재해야 한다.

<webservice-description>는 Java EE Endpoint를 WSDL 포트 정의, DD 구현, JAX-RPC 매핑 파일 그리고 Endpoint 인터페이스로 바인딩시켜주는 역할을 하며, 본 장에서는 이들의 관계와 필요성에 대해서 설명 한다.

• <wsdl-file>

<wsdl-file>는 WSDL 문서의 WAR나 EJB JAR 파일 내에서의 상대적인 위치를 나타내며, webservices.xml 파일은 WSDL 파일과 같은 WAR나 JAR 파일 내에 위치해야 한다.

• <jaxrpc-mapping-file>

<jaxrpc-mapping-file>는 JAX-RPC 매핑 파일의 위치를 지정한다. JAX-RPC 매핑 파일은 WSDL 파일과 Java EE Endpoint 간의 매핑을 정의한다. JAX-RPC 매핑 파일에 관한 더 자세한 설명은 “24.2. 웹 서비스의 메핑 파일 작성”을 참고한다.

• <port-component>

<port-component>는 특정한 Java 클래스나 EJB Endpoint를 WSDL 문서 내의 특정한 port element로 매핑하는 역할을 한다. Java 클래스의 경우에 WSDL port 정의와 바인딩 정의는 웹 서비스를 호스트하기 위해 필요한 서블릿을 생성하는 데 사용된다.

EJB Endpoint의 경우에는 port와 바인딩 정의는 EJB 컨테이너가 SOAP 메시지를 무상태 Bean 객체에 보내기 위해 마샬링 하는 데 사용된다.

– <port-component-name>

<port-component-name>는 특정한 Java 클래스나 EJB Endpoint를 지정하기 위한 이름을 제공한다. 이 이름은 webservices.xml 파일 내에서 유일해야 한다.

– <service-endpoint-interface>

Java 클래스를 Back-end로 가지는 웹 서비스의 경우, SEI의 이름을 지정하는 element이다.
EJB Endpoint 웹 서비스의 경우에도 마찬가지로 이 element에서 정의하며, 추가적으로 ejb-jar.xml의 <service-endpoint>에서도 같은 이름으로 정의해야 한다.

- <service-impl-bean>

<service-impl-bean>는 서비스 배치가 수행되는 시점에 어떤 서비스의 로직 구현 정의가 Java EE Endpoint가 될 것인지 정의한다. Java 클래스의 경우에는 web.xml의 서블릿 정의를 가리키고, EJB Endpoint의 경우에는 ejb-jar.xml의 session 정의를 가리킨다.

JSE의 경우에는 <servlet-link>에 정의되며 예는 다음과 같다.

[예 24.2] DocLitEchoService : <<webservices.xml >>

```xml
<?xml version="1.0"?>
<webservices ...>
  <webservice-description>
    <webservice-description-name>DocLitEchoService</webservice-description-name>
    <wsdl-file>WEB-INF/wsdl/DocLitEchoService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>WEB-INF/DocLitEchoService-mapping.xml</jaxrpc-mapping-file>
  </webservice-description>
</webservices>
```

<servlet-link>의 값은 web.xml 내의 <servlet-name>의 값과 일치해야 한다.

[예 24.3] DocLitEchoService : << web.xml >>

```xml
<?xml version="1.0"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee">
  <servlet>
    <servlet-name>EchoServlet</servlet-name>
    <servlet-class>jeustest.webservices.java2wsdl.doclit.EchoImpl</servlet-class>
  </servlet>
</web-app>
```
위와 마찬가지로 EJB Endpoint 웹 서비스의 경우 <ejb-link>를 사용하여 정의한다.


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

<ejb-link>의 값은 ejb-jar.xml 파일 내의 <ejb-name>의 값과 일치해야 한다.

[예 24.5] AddressBookService : << web.xml >>

```xml
<?xml version="1.0" encoding="UTF-8"?>
<webservices ...
  <webservice-description>
    <webservice-description-name>AddressBookService</webservice-description-name>
    <wsdl-file>META-INF/wsdl/AddressBookService.wsdl</wsdl-file>
    <jaxrpc-mapping-file>META-INF/AddressBookService-mapping.xml</jaxrpc-mapping-file>
    <port-component>
      <port-component-name>AddressBookIFPort</port-component-name>
      <wsdl-port xmlns:ns2="urn:AddressBookService">
        ns2:AddressBookIFPort
      </wsdl-port>
      <service-endpoint-interface>address.AddressBookIF</service-endpoint-interface>
      <service-impl-bean>
        <ejb-link>AddressEJB</ejb-link>
      </service-impl-bean>
    </port-component>
  </webservice-description>
</webservices>
```
한 가지 유의해야 할 사항은 <servlet-link>나 <ejb-link>를 통해 연결될 대상들은 webservices.xml를 포함하는 압축 파일 안에 존재해야 한다는 것이다.

- <handler>

 메시지 핸들러는 Java EE Endpoint가 보내고 받는 메시지를 간접하여 가공하는 역할을 한다.
webservices.xml 파일 안에서 이 메시지 핸들러들을 설정하고, 처리 순서를 지정할 수 있다. 메시지 핸들러에 대한 더 자세한 설명은 "제23장 JAX-RPC 웹 서비스 SOAP 메시지 핸들러 생성"을 참고한다.

다음은 핸들러를 설정한 webservices.xml의 작성 예이다.


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

d음은 <handler>의 하위 항목에 대한 설명이다.

<table>
<thead>
<tr>
<th>항목</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;handler-name&gt;</td>
<td>정의되는 값은 webservices.xml 파일 안에서 유일한 값이 되어야 하며, 설정한 핸들러에 대한 식별자 역할을 한다.</td>
</tr>
<tr>
<td>&lt;handler-class&gt;</td>
<td>핸들러를 구현한 클래스를 정의한다.</td>
</tr>
<tr>
<td>항목</td>
<td>설명</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>설명항목</td>
<td>정의되는 핸들러 클래스는 javax.xml.rpc.handler.Handler 인터페이스를 직접 또는 간접적으로 구현해야 한다.</td>
</tr>
<tr>
<td>&lt;init-param&gt;</td>
<td>필수 사항은 아니며 메시지 핸들러의 생성 주기의 시작 단계에서 핸들러를 초기화하기 위해 전달될 파라미터를 정의한다. 런터임에 HandlerInfo.getHandlerConfig() 메소드를 실행해서 얻어지는 java.util.Map 객체로부터 파라미터들을 가져올 수 있다.</td>
</tr>
<tr>
<td>&lt;soap-header&gt;</td>
<td>핸들러가 처리해야 하는 SOAP Header 블록의 규 네임(Q Name : Qualified Name) 값을 정의하며, &lt;handler&gt;는 하나 이상의 &lt;soap-header&gt;를 포함할 수 있다.</td>
</tr>
<tr>
<td>&lt;soap-role&gt;</td>
<td>Header 블록을 처리할 때 핸들러가 정하는 역할을 정의한다. &lt;soap-role&gt; 값이 &lt;soap-header&gt;와 짝을 이루어서 정의될 때에만 어떤 Header 블록을 핸들러가 처리하는지 정확하게 알 수 있다. SOAP 메시지는 각각의 Header 블록을 처리하기 위한 역할을 명시적으로 지정할 수 있다. actor 속성이 정의되어 있으면, 정의된 역할을 수행하는 노드만이 Header 블록을 처리하고, actor 속성이 정의되지 않은 경우에는 최종적인 메시지 수신자가 header 블록을 처리한다.</td>
</tr>
</tbody>
</table>

### 24.2. 웹 서비스의 매핑 파일 작성

JAX-RPC 매핑 파일은 JEUS 웹 서비스에 내장된 JAX-RPC 컴파일러가 WSDL 문서와 웹 서비스 Endpoint 를 나타내는 Java 인터페이스의 관계를 이해하는 것을 도와준다. 많은 경우에 WSDL과 Java의 매핑은 매핑 파일이 없어도 큰 문제가 없지만 명시적인 정의가 필요할 때가 있으며, 그러한 필요에 의해서 JAX-RPC 매핑 파일이 도입되었다.

JAX-RPC 매핑 파일은 Java EE 웹 서비스 Endpoint나 Java EE 웹 서비스 클라이언트를 사용할 때마다 필요하며, WSDL 문서와 JAX-RPC 매핑 파일은 1대 1로 대응한다. 즉, 각각의 WSDL 파일에는 하나씩의 JAX-RPC 매핑 파일이 존재한다.

### 24.2.1. JAX-RPC 매핑 파일 내용

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

- XML 복합 타입(Complex Type)과 Java Bean
- Fault 메시지와 예외 클래스
- WSDL의 portType 정의와 SEI(Service Endpoint Interface)
- WSDL의 서비스 정의와 서비스 인터페이스

JAX-RPC 매핑 파일에 정의되지 않은 것들은 WSDL과 XML의 Java로의 표준 매핑 범칙을 따라서 매핑된다. 그리고 매핑 파일에서 정의 되는 것들은 항상 표준 매핑 범칙에 우선한다.
24.2.2. JAX-RPC 매핑 파일 작성

매핑 파일은 상당히 내용이 복잡해 보이고, 크기에 있어서도 단일 설정 파일 중에 큰 편이다. 전체적인 설정 파일의 구성은 다음과 같다.

[예 24.7] JAX-RPC 매핑 파일의 구조

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

- **<java-wsdl-mapping>**
  JAX-RPC 매핑 파일의 최상위 element이며, 다른 매핑 element들을 포함하고 있다.

- **<package-mapping>**
  JAX-RPC 컴파일러가 WSDL에 정의된 여러 가지 타입에 대해 Java 클래스와 인터페이스 정의를 생성하려고 할 경우 사용하는 항목으로 반드시 설정되어야 한다.

  `<package-type>`의 값은 Java 패키지의 이름이며, `<namespaceURI>`의 값은 지정된 Java 패키지로 매핑되어야 하는 XML 이름 공간이다.

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

- **<java-xml-type-mapping>**
XML Schema 복합 타입(Complex Type)이나 단순 타입(Simple Type)을 사용할 때 필요하며, XML Schema와 Java 타입 간의 관계를 정의한다. XML Schema에 정의된 built-in 타입이 Java로 표준적인 매핑이 된다면 이 element를 사용하지 않아도 된다.

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

● <exception-mapping>
WSDL 결함(fault) 메시지를 Java 예외 클래스로 매핑한다.

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

● <service-interface-mapping>
WSDL의 서비스 정의를 JAX-RPC 서비스 인터페이스 타입으로 매핑한다.
<service-interface>는 WSDL의 서비스 정의를 나타내는 Java 인터페이스의 클래스 이름을 정의하고, 서비스 인터페이스의 패키지 이름은 <package-mapping>에 정의된 패키지 이름과 일치해야 한다.

<port-mapping>는 서비스 인터페이스의 getPortName() 메소드의 포트 이름을 WSDL의 <port>와 대응 하게 정의한다.

위와 같이 정의된 경우 생성되는 Java 서비스 인터페이스 정의는 다음과 같다.

[예 24.8] << AddressBookService.java >>

```java
package address;

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

● <service-endpoint-interface-mapping>

SEI를 WSDL의 portType과 바인딩 정의로 매핑한다. 이 element는 JAX-RPC 컴파일러가 적절한 Endpoint Stub과 Endpoint 인터페이스를 생성할 때 필요한 정보를 제공한다. 또한 WSDL의 operation과 message part 정의가 어떻게 Java Endpoint 메소드의 정의로 매핑될 것인지에 대한 상세 정보를 제공한다.

다음은 AddressBookService의 매핑 파일에 정의된 예이다.

```xml
<java-wsdl-mapping...>
    ...
</java-wsdl-mapping>
```
<service-endpoint-interface>
  address.AddressBookIF
</service-endpoint-interface>
<ns4:wsdl-port-type
  xmlns:ns4="http://java.sun.com/xml/ns/j2ee"
  xmlns="urn:AddressBookService">
  AddressBookIF
</ns4:wsdl-port-type>
<ns5:wsdl-binding
  xmlns:ns5="http://java.sun.com/xml/ns/j2ee"
  xmlns="urn:AddressBookService">
  AddressBookIFSoapBinding
</ns5:wsdl-binding>
<service-endpoint-method-mapping>
  <java-method-name>add</java-method-name>
  <wsdl-operation>add</wsdl-operation>
  <wrapped-element/>
  <method-param-parts-mapping>
    <param-position>0</param-position>
    <param-type>address.PersonInfo</param-type>
    <wsdl-message-mapping>
      <ns6:wsdl-message
        xmlns:ns6="http://java.sun.com/xml/ns/j2ee"
        xmlns="urn:AddressBookService">
        addRequest
      </ns6:wsdl-message>
      <wsdl-message-part-name>parameters</wsdl-message-part-name>
      <parameter-mode>IN</parameter-mode>
    </wsdl-message-mapping>
  </method-param-parts-mapping>
  <wsdl-return-value-mapping>
    <method-return-value>
      address.PersonInfo[]
    </method-return-value>
    <ns7:wsdl-message
      xmlns:ns7="http://java.sun.com/xml/ns/j2ee"
      xmlns="urn:AddressBookService">
      addResponse
    </ns7:wsdl-message>
    <wsdl-message-part-name>parameters</wsdl-message-part-name>
    <wsdl-message-part-name></wsdl-message-part-name>
    <parameter-mode>OUT</parameter-mode>
  </wsdl-return-value-mapping>
</service-endpoint-method-mapping>

...
제24장 JAX-RPC 웹 서비스 설정 파일 작성 299
제25장 JAX-RPC 웹 서비스 데이터 타입

본 장에서는 JAX-RPC 웹 서비스에 관한 여러 가지 데이터 타입 문제에 관해 설명한다.

표준 Java/XML 데이터 타입 매핑(type mapping)과 사용자 정의 클래스들에서 웹 서비스 파라미터로 사용 할 JAX-RPC의 value 타입에 대해서 설명한다. 그 다음으로 출력 또는 입/출력 파라미터들을 위한 JAX-RPC의 Holder 클래스를 설명하고, 마지막으로 에러 정보를 웹 서비스 클라이언트로 어떻게 전송하는지를 설명한다.

25.1. 개요

본 절에서는 웹 서비스와 웹 서비스 클라이언트에서 사용하는 데이터 타입에 관한 이슈에 대해서 설명한다.

다음은 웹 서비스 데이터 타입의 특징이다.

- **Java와 XML 타입 매핑**
  XML 인스턴스를 Java 객체로 직렬화 또는 역직렬화((de)serialize)하기 위해서는 XML의 태그와 Java 클래스들의 태그들과의 타입 매핑이 필요하다.
  JEUSS 웹 서비스는 JAX-RPC 스펙에 설명된 Java와 표준 XML 태그 매핑을 따르고 있다.

- **JAX-RPC Value 타입 사용**
  JAX-RPC Value 타입은 웹 서비스의 요청이나 응답에 값으로써 전달될 수 있는 타입을 말한다.
  JAX-RPC Value 타입은 일반적으로 사용자 정의 JavaBeans 클래스로 표현된다. JEUSS 웹 서비스는 사용자가 작성한 JavaBeans 태그를 웹 서비스의 파라미터와 반환 값으로서 사용할 수 있도록 하고 있다.

- **입출력(In/Out) 파라미터로서 Holder의 사용**
  입출력(In/Out) 파라미터는 웹 서비스를 실행할 때 입력은 물론 출력에서도 사용되는 파라미터이다. 웹 서비스가 여러 출력값들을 반환해야 할 경우 입/출력 파라미터를 사용할 수 있다.
  JEUSS 웹 서비스는 입/출력 포맷과 표준 JAX-RPC Holder 클래스를 사용하여 지원하고 있다.

- **SOAP Fault로서의 Exception 사용**
  웹 서비스 실행 중 어떤 에러가 발생하게 되면 웹 서비스 클라이언트에게 그 내용이 전달되어야 한다.
  SOAP 표준에서는 이런 목적을 위한 SOAP Fault를 정의하고 있다. JEUSS 웹 서비스는 표준 SOAP Fault를 지원한다.
25.2. Java와 XML 타입 매핑

지금까지 하나의 타입만(String만)을 파라미터와 반환값으로 사용하는 웹 서비스 예제를 보았다. JEUS 웹 서비스는 Java 타입을 XML 또는 WSDL 정의로 매핑한다. 예를 들어 JEUS 웹 서비스는 java.lang.String 클래스를 XML xsd:string 데이터 타입으로 매핑한다.

본 절에서는 JEUS 웹 서비스에서 지원하는 데이터 타입들의 종류와 어떤 데이터 타입이 JEUS 웹 서비스에서 사용되기 위해 필요한 요구 사항인지에 대해서 설명한다.

참고
애플리케이션 개발자들은 Java와 XML 타입의 매핑 과정에 대해서 자세히 알아야 할 필요는 없다. 그러나 모든 Java 클래스가 파라미터와 반환 타입으로 사용될 수 있는 것은 아님에 주의한다.

25.2.1. 내장 타입 매핑

다음은 Java와 XML의 데이터 형과 JEUS 웹 서비스의 내장 타입 매핑(Built-in Type Mapping)을 정리한 표이다.

<table>
<thead>
<tr>
<th>XML 데이터 타입</th>
<th>Java 데이터 타입</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd:string</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>xsd:boolean</td>
<td>boolean, java.lang.Boolean *</td>
</tr>
<tr>
<td>xsd:double</td>
<td>double, java.lang.Double *</td>
</tr>
<tr>
<td>xsd:float</td>
<td>float, java.lang.Float *</td>
</tr>
<tr>
<td>xsd:int</td>
<td>int, java.lang.Integer</td>
</tr>
<tr>
<td>xsd:integer</td>
<td>java.math.BigInteger</td>
</tr>
<tr>
<td>xsd:long</td>
<td>long, java.lang.Long *</td>
</tr>
<tr>
<td>xsd:short</td>
<td>short, java.lang.Short *</td>
</tr>
<tr>
<td>xsd:byte</td>
<td>byte, java.lang.Byte</td>
</tr>
<tr>
<td>xsd:Decimal</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>xsd:base64Binary</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>xsd:hexBinary</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>xsd:QName</td>
<td>javax.xml.rpc.namespace.QName</td>
</tr>
<tr>
<td>xsd:dateTime</td>
<td>java.util.Calendar</td>
</tr>
<tr>
<td>xsd:gYearMonth</td>
<td>java.util.Calendar</td>
</tr>
<tr>
<td>xsd:gYear</td>
<td>java.util.Calendar</td>
</tr>
<tr>
<td>xsd:gMonthDay</td>
<td>java.util.Calendar</td>
</tr>
</tbody>
</table>

302  JEUS Web Service 안내서
<table>
<thead>
<tr>
<th>XML 데이터 타입</th>
<th>Java 데이터 타입</th>
</tr>
</thead>
<tbody>
<tr>
<td>xsd:anyURI</td>
<td>java.net.URI (JDK 1.4 or over) / java.lang.String (JDK 1.4)</td>
</tr>
<tr>
<td>xsd:duration</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>xsd:name</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>xsd:NCName</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>xsd:QName</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>xsd:NCName</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>xsd:SToken</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>SOAP-ENC:base64</td>
<td>byte[]</td>
</tr>
<tr>
<td>SOAP-ENC:string</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>SOAP-ENC:boolean</td>
<td>boolean, java.lang.Boolean *</td>
</tr>
<tr>
<td>SOAP-ENC:double</td>
<td>double, java.lang.Double *</td>
</tr>
<tr>
<td>SOAP-ENC:float</td>
<td>float, java.lang.Float *</td>
</tr>
<tr>
<td>SOAP-ENC:int</td>
<td>int, java.lang.Integer *</td>
</tr>
<tr>
<td>SOAP-ENC:long</td>
<td>long, java.lang.Long *</td>
</tr>
<tr>
<td>SOAP-ENC:short</td>
<td>short, java.lang.Short *</td>
</tr>
<tr>
<td>SOAP-ENC:byte</td>
<td>byte, java.lang.Byte *</td>
</tr>
<tr>
<td>SOAP-ENC:integer</td>
<td>java.math.BigInteger</td>
</tr>
<tr>
<td>SOAP-ENC:decimal</td>
<td>java.math.BigDecimal</td>
</tr>
</tbody>
</table>

참고
1. ‘xsd’ 접두어는 XML namespace URI(http://www.w3.org/2001/XMLSchema)를 나타낸다.
3. 만약 WSDL에서 어떤 객체가 null이 될 수 있다고 정의되어 있다면 서비스 호출자는 xsd:nil을 데이터로 보내거나 받을 때 사용할 수 있다. 그러한 Java primitive 타입은 Wrapper 클래스로 교체된다. 위의 표에서 Java 타입 뒤에 붙은 ***는 이것을 나타낸다.
DII 클라이언트에서는 어떤 타입을 지정할 때, XML의 QName을 사용할 수 있다.

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

25.2.2. 배열

JEUS 웹 서비스는 JAX-RPC 태그에서 정의한 배열들을 지원한다. 예를 들어 int[]와 String[]이나 다차원 배열인 java.math.BigDecimal[][]도 지원한다.

25.2.3. 사용자 정의 타입 : JAX-RPC Value Type

JEUS 웹 서비스는 애플리케이션을 위해 작성한 사용자 정의 타입을 지원한다. JAX-RPC 스펙에는 이러한 클래스들을 Value Type이라고 한다.

JEUS 웹 서비스에서 이것을 지원하기 위해서는 사용자 정의 클래스들은 다음 규칙을 따라야 한다.

- 파라미터 없는 public default 생성자를 가져야 한다.
- 직접 또는 간접적으로 java.rmi.Remote를 구현해서는 안 된다.
- 멤버 필드들의 타입은 JEUS 웹 서비스가 지원하는 타입이어야 한다.

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

- public 필드는 final이나 transient가 될 수 없다.
- public 필드가 아닌 것은 관련된 getter와 setter 메소드를 가져야 한다.

위의 규칙을 따르는 JavaBeans 컴포넌트 또한 지원된다.

25.3. JAX-RPC Value 타입의 사용

"25.2.3. 사용자 정의 타입 : JAX-RPC Value Type"에서 설명한 규칙들을 따르는 사용자 정의형들은 웹 서비스의 파라미터나 반환 타입으로써 사용이 가능하다. 이런 종류의 사용자 정의 타입은 JAX-RPC Value 타입이라고 한다.

본 절에서는 JAX-RPC Value 타입을 사용하는 예제를 설명한다. CalcService는 2개의 숫자와 하나의 연산자를 받아서 결과를 숫자로 넘겨주는 예제이다. 그리고 CalcService를 위한 웹 서비스 클라이언트를 작성한다.
25.3.1. JAX-RPC Value 타입을 사용하는 웹 서비스 생성

다음은 CalcService 소스 코드인 Calculator.java 코드이다.

[예 25.1] <<Calculator.java>>

```java
package calc;

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

        if (op.equals("plus")) {
            ret = num1 + num2;
        } else if (op.equals("minus")) {
            ret = num1 - num2;
        } else if (op.equals("mult")) {
            ret = num1 * num2;
        } else if (op.equals("div")) {
            if (num2 != 0)
                ret = num1 / num2;
        }
        return ret;
    }
}
```
calc() 메소드는 CalcData형을 인자로 받고, 에러가 발생하면 -9999.0을 반환한다. 이후에 설명하는 예제에서는 좀 더 확장된 에러 처리방법을 설명할 것이다.

다음은 CalcData.java의 소스 코드이다.

[예 25.2] <<CalcData.java>>

```java
package calc;

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

    public CalcData() { }
    public double getNum1() { return num1; }
    public double getNum2() { return num2; }
    public String getOp() { return op; }
```
public void setNum1(double n) { num1 = n; }
public void setNum2(double n) { num2 = n; }
public void setOp(String s) { op = s; }
}

CalcData 클래스는 JAX-RPC Value Type 요구 사항을 따르고 있다. 이것은 CalcData의 인스턴스는 값으로서 전달될 수 있음을 의미한다.
다음은 Service Endpoint Interface 파일 CalculatorIF.java 코드이다.

[예 25.3] <<CalculatorIF.java>>

```java
package calc;

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

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

이 파일들을 컴파일하기 위해서 다음과 같이 명령을 수행한다.

```bash
ant compile
```
위 명령은 결과 클래스 파일들을 build 디렉터리 아래에 옮겨놓을 것이다.
배치 가능한 EAR 파일을 생성하기 위해서는 다음 명령을 수행한다.

```bash
ant wsear
```
웹 서비스 모듈을 배치하면 다음과 같은 주소로 서비스에 접근할 수 있다.

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

### 25.3.2. JAX-RPC Value 타입을 사용하는 웹 서비스 클라이언트 생성

다음의 명령을 수행하면 CalcService 웹 서비스를 위한 프록시 클라이언트를 생성한다.

```bash
ant wsd12java
```
생성된 Stub 코드의 패키지 이름은 com.test.calc로 가정한다.
다음은 클라이언트 프로그램의 소스 코드이다.

[예 25.4] <<CalcClient.java>>

```java
import com.test.calc.*;
import javax.xml.rpc.soap.SOAPFaultException;
```
public class CalcClient {
    public static void main(String[] args) {
        CalcClient calc = new CalcClient();

        if (args.length != 3) {
            System.out.println("usage: java CalcClient num1 op num2");
            System.out.println(" where op is one of "+"'plus', 'minus',
            'mult', 'div'");
            System.exit(1);
        }
        try {
            calc.run(args);
        } catch (SOAPFaultException e) {
            System.err.println("faultcode = " + e.getFaultCode());
            System.err.println("faultString = " + e.getFaultString());
        } catch (Exception e) {
            System.err.println(e.toString());
            e.printStackTrace();
        }
    }

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

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

클라이언트 코드 구현이 완료되면 다음 명령을 통해 클라이언트 코드와 Stub 코드를 컴파일한다.

ant build

클라이언트를 실행하기 위해서 다음과 같이 Ant Task를 실행한다.

ant runclient

성공하면 다음과 같은 결과가 출력된다.

2.0
25.4. Holder 클래스

웹 서비스가 여러 개의 값을 반환하기를 원한다면 JAX-RPC Value 타입을 정의하거나 출력 또는 입/출력 파라미터를 하나 이상 지정해야 한다. Holder 클래스는 출력 또는 입/출력 파라미터로 사용되는 Helper 클래스이다.

25.4.1. 내장 Holder 클래스

JEUS 웹 서비스는 단순 데이터 타입의 JAX-RPC Holder 클래스들을 제공한다. JAX-RPC가 지원하는 표준 Holder 클래스들은 다음과 같다.

[표 25.2] 내장 Holder 클래스

<table>
<thead>
<tr>
<th>Holder Class</th>
<th>Java Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>javax.xml.rpc.holders.BooleanHolder</td>
<td>boolean</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.ByteHolder</td>
<td>byte</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.ShortHolder</td>
<td>short</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.IntHolder</td>
<td>int</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.LongHolder</td>
<td>long</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.FloatHolder</td>
<td>float</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.DoubleHolder</td>
<td>double</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.BigDecimalHolder</td>
<td>java.math.BigDecimal</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.BigIntegerHolder</td>
<td>java.math.BigInteger</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.ByteArrayHolder</td>
<td>byte[]</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.CalendarHolder</td>
<td>java.util.Calendar</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.QNameHolder</td>
<td>javax.xml.namespace.QName</td>
</tr>
<tr>
<td>javax.xml.rpc.holders.StringHolder</td>
<td>java.lang.String</td>
</tr>
</tbody>
</table>

각 Holder 클래스들이 가지고 있는 값을 액세스하기 위해 value 필드를 사용한다.

다음은 Calculator.java를 수정한 소스이다.

[예 25.5] << Calculator.java >>

```java
package calc;

import javax.xml.rpc.holders.DoubleHolder;

public class Calculator implements CalculatorIF {
    public Calculator() {
    }
```
소스 코드는 내장 Holder 클래스인 javax.xml.rpc.holders.DoubleHolder를 import하였다.

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

calc() 메소드의 signature 또한 수정되었다. 리턴 타입은 void이고, 두 번째 파라미터로 Holder 객체를 받는다.

```java
public void calc(CalcData data, DoubleHolder result){
    // ...
    result.value = ret;
}
```

 Holder 객체가 가진 객체 값을 접근하기 위해서는 Holder 클래스의 value 필드를 이용한다.

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

웹 서비스를 위한 웹 서비스 클라이언트를 생성하기 전에 이 웹 서비스를 패키지로 생성하고 배치해야 한다. 또한 클라이언트 프로그램 CalcClient.java도 수정되어야 한다.

다음은 클라이언트 소스 코드 CalcClient.java의 run() 메소드의 내용이다.

[예 25.6] << CalcClient.java의 run() 메소드 >>

```java
public void run(String[] args) throws Exception {
    CalculatorIF port = new CalcService_Impl().getCalculatorIFPort();
    CalcData data = new CalcData();
    data.setNum1((new Double(args[0])).doubleValue());
    data.setNum2((new Double(args[2])).doubleValue());
    data.setOp(args[1]);
    // ...
    result = port.calc(data);
    // ...
}
```
DoubleHolder ret = new DoubleHolder();
port.calc(data, ret);
System.out.println(ret.value);
}

주의
CalcClient.java를 컴파일하기 전에 프록시 소스 코드를 다시 생성하는 것을 잊지 않도록 주의한다.

25.4.2. 사용자 정의 타입을 위한 Holder 클래스 작성

다음은 사용자가 작성한 클래스의 Holder 클래스를 작성하는 과정이다.
1. javax.xml.rpc.holders.Holder 인터페이스를 구현한다.
2. 사용자가 작성한 클래스를 TypeHolder라고 명명한다.
   Type은 Holder 객체가 가지게 될 클래스의 이름이다. 예를 들어 CalcData 클래스를 위한 Holder를 작성하기 위해서는 Holder 클래스의 이름은 CalcDataHolder가 된다.
3. 작성한 Holder 클래스를 public으로 선언한다.
4. value라는 이름을 가진 public 필드를 생성한다. 이 데이터 타입은 Holder 클래스의 타입과 같다.
5. value 필드를 기본값으로 초기화하는 파라미터 없는 default 생성자를 생성한다.
6. 파라미터 값으로 value 필드를 설정하는 생성자를 생성한다.

JAX-RPC Value 타입의 Holder 클래스를 사용하는 방법을 설명하기 위해서 CalcService 예제를 수정한다. 다음 예제에 클래스 CalcData는 계산 결과값을 멤버로 포함한다. private 변수인 result와 getter/setter 메소드를 멤버로 추가하였다.

[예 25.7] << CalcData.java >>

```java
package calc;

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

    public CalcData() { }
    public double getNum1() { return num1; }
    public double getNum2() { return num2; }
    public String getOp() { return op; }
    public double getResult() { return result; }
}
```
public void setNum1(double n) { num1 = n; }
public void setNum2(double n) { num2 = n; }
public void setOp(String s) { op = s; }
public void setResult(double n) { result = n; }
}

CalcData의 Holder 클래스는 CalcDataHolder이다. CalcDataHolder 클래스는 데이터 타입이 CalcData인
public의 value 필드와 value 필드를 초기화하는 2개의 생성자를 가지고 있다.

[예 25.8] << CalcDataHolder.java >>

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

public class CalcDataHolder implements Holder {
    public CalcData value;

    public CalcDataHolder() {
    }

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

Holder 클래스를 사용하는 Calculator.java 소스 코드는 다음과 같다. calc() 메소드는 데이터 타입이 Calc
DataHolder인 파라미터를 지정하고 있다. Holder 객체가 가지고 있는 객체를 접근하기 위해 public의 value
필드를 사용하고 있다.

package calc;

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

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

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

        System.out.println("Result: "+ ret);
    }
}
다음은 클라이언트 소스 코드의 내용이다. `Ant wsd2java` 명령 또한 WSDL 문서로부터 클라이언트를 위한 Holder 클래스들을 생성한다. `Ant wsd2java` 명령으로부터 생성된 Holder 클래스들은 사용자가 작성한 것과는 다르다는 것을 유념한다.

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

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

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

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

    public void run(String[] args) throws Exception {
        CalculatorIF port = new Calculator3Service_Impl().getCalculatorIFPort();
        CalcData data = new CalcData();
        CalcDataHolder dataHolder = new CalcDataHolder(data);

        data.setNum1((new Double(args[0])).doubleValue());
        data.setNum2((new Double(args[2])).doubleValue());
    }
}
```
25.5. Exception과 SOAP Fault


CalcService 예제에서 에러 상황이 발생할 경우 -9999.0 값을 반환값으로 사용하였던 소스 코드는 다음과 같이 변경할 수 있다.

[예 25.9] << Calculator.java >>

```
package calc;

import java.rmi.RemoteException;

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

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

## 25.6. MIME 타입을 DataHandler 타입으로 매핑

웹 서비스 클라이언트는 Attachment를 SOAP 메시지에 첨부하여 전송할 수 있다. JAX-RPC 스펙은 Attachment의 MIME 타입에 상응하는 Java 타입 매핑을 정의하고 있지만, 경우에 따라 웹 서비스 클라이언트가 MIME 타입에 관계없이 항상 javax.activation.DataHandler 타입으로 매핑하여 사용할 수도 있다.

본 절에서는 WSDL-to-Java 매핑 툴을 사용하여 MIME 타입을 항상 DataHandler 타입으로 매핑하는 방법을 설명한다.

### 25.6.1. Wsdl2java에서 dataHandlerOnly 옵션 사용

다음과 같이 MIME part를 가진 웹 서비스의 WSDL이 있다.

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

314  JEUS Web Service 안내서
기본적으로 이러한 WSDL을 가지고 wsdl2java 를 사용하여 SEI를 생성하면 다음과 같이 MIME 타입 "application/xml"은 javax.xml.transform.Source 타입으로 매핑된다.

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

Ant Task, wsdl2java에서 attribute, dataHandlerOnly="true"로 설정하거나 Command Line 툴에서 --data handleronly 옵션을 사용하면, 다음과 같이 MIME 타입에 상관없이 part는 항상 javax.activation.DataHandler 타입으로 매핑된다.

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

따라서 웹 서비스 클라이언트 개발자는 DataHandler 타입으로 Attachment를 송부해야 한다.

다음은 DataHandler 타입을 사용한 웹 서비스 클라이언트 예제이다.

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

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

25.7. Doc/Literal에서 데이터 바인딩을 사용하지 않기

JAX-RPC 스펙에는 XML 타입에 대한 Java 타입 매핑을 정의하고 있다. 그러나 WSDL의 타입에 의해 매핑된 Java 타입을 사용하기 보다는 SOAPElement를 직접 구성하여 메시지를 전송하는 것이 더욱 편리할 수도 있다.

본 절에서는 WSDL-to-Java 매핑 를 사용하여 XML 타입에 상관없이 javax.xml.soap.SOAPElement를 사용하는 방법을 설명한다.

25.7.1. Wsdl2java에서 noDataBinding 옵션 사용

다음과 같은 Document/Literal로 기술된 WSDL이 있다.
이 WSDL로부터 wsdl2java로 생성한 SEI는 다음과 같이 Object 타입의 Input 파라미터를 갖는다.

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

만약 wsdl2java로 생성한 SEI를 생성할 때 Ant Task, wsdl2java에서 attribute, noDataBinding="true"로 설정하거나, Command Line 툴에서 –nodatabinding 옵션을 사용하면 다음과 같이 XML 태입에 상관없이 Input 파라미터 및 Return value 타입은 javax.xml.soap.SOAPElement이 된다.
public interface BookQuote extends java.rmi.Remote {
    public javax.xml.soap.SOAPElement
        getBookPrice(javax.xml.soap.SOAPElement book)
        throws java.rmi.RemoteException;
}

wsdl2java의 nodatabinding 옵션은 Document/Literal의 WSDL에서만 유효하다.
이 경우 웹 서비스 클라이언트 개발자는 SOAPElement 타입으로 메시지를 직접 구성해야 한다.
다음은 SOAPElement 타입을 사용한 웹 서비스 클라이언트 예제이다.

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

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

SOAPElement title = factroy.createElement("title");
title.addTextNode("Sample for a option: nodatabinding");
book.addChildElement(title);

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

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

// Get a Service port
BookQuote port = new BookQuoteService_Impl().getBookQuotePort();
SOAPElement price = port.getBookPrice(book);
System.out.println("price = " + price.getValue());
제26장 JAX-RPC 웹 서비스의 보안

본 장에서는 JEuS에서 JAX-RPC 웹 서비스에 보안을 적용하기 위한 방법을 설명한다.

26.1. 개요

전통적으로 웹 서비스에 보안을 적용하기 위해서는 다음과 같은 2가지 방식이 존재한다.

- 전송 수준 보안 (Transport-level Security)
  SSL을 이용하여 클라이언트와 웹 서비스 사이의 연결의 보안을 보장한다.
  전송 수준 보안을 적용하려면 클라이언트와 JEuS 서버 간의 연결을 SSL을 이용하여 안전하게 할 수 있다. 그러나 이러한 방식은 연결 자체만을 안전하게 하며 클라이언트와 JEuS 서버 사이에 라우터나 메시지 큐와 같은 매개체 (Intermediary)가 있다면 매개체는 SOAP 메시지를 암호화되지 않은 원래의 텍스트 문서 형태로 가질 수 있다. 또 전송 수준 보안은 전체적인 메시지를 다루므로 메시지 일부에 의한 보안 적용이 불가능하다.

- 메시지 수준 보안
  SOAP 메시지를 전자 서명이나 암호화한다.
  메시지 수준 보안은 SSL의 보안상의 장점을 포함하면서 보안적인 유연성을 제공한다. 메시지 수준 보안은 메시지 전달 과정에서 하나 이상의 매개체가 존재하더라도 보안이 유지되는 End-to-End 보안이며, 연결 자체의 보안 유지되는 SOAP 메시지 자체의 서명과 암호화를 의미한다. 또한 부분적인 서명과 암호화가 가능한다는 장점이 있다.

26.2. 전송 수준 보안

웹 서비스의 전송 수준 보안은 웹 서비스 클라이언트 응용 프로그램과 웹 서비스 간의 연결을 SSL을 이용하여 안전하게 하는 것을 의미한다.

전체적인 절차는 다음과 같다.

1. JEuS 서버의 SSL 설정한다.
   웹 서비스 개발 부분은 추가 작업이 필요하지 않다. JEuS 서버에서의 SSL 설정에 대한 자세한 내용은 "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. wsdl2java를 이용하여 WSDL로부터 Stub을 생성하거나 클라이언트에서 웹 서비스를 호출하기 위해 클라이언트를 실행하려고 할 때 시스템 프로퍼티 값을 다음과 같이 설정한다.

```java
-Djavax.net.ssl.trustStore=keystore_name
-Djavax.net.ssl.trustStorePassword=keystore_password
```

d. Ant가 아닌 wsdl2java 콘솔 툴을 사용한다면 다음과 같은 환경변수 설정이 필요하다.

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

26.3. 메시지 수준 보안

메시지 수준의 보안은 웹 서비스와 웹 서비스를 호출하는 클라이언트 간의 SOAP 메시지가 서명이나 암호화되는 것을 의미한다.

JEUS 웹 서비스는 다음과 같은 OASIS 웹 서비스 보안 표준 1.0을 지원한다.

- OASIS Standard 1.0
  - SOAP Message Security V1.0
  - Username Token Profile V1.0
  - X.509 Token Profile V1.0

위와 같은 표준들은 인증, 권한 부여, 데이터 무결성 및 기밀성 보장에 관한 요구 조건을 명시하며 각종 기업 애플리케이션들에 존재하는 보안 쟁점들을 해결할 수 있게 한다.

26.3.1. 웹 서비스 보안 적용

JEUS 웹 서비스에서는 다음과 같이 적용이 가능하다.

- 클라이언트는 X.509 인증서를 사용하여 SOAP 메시지의 암호화와 서명을 하여 보안이 적용된 웹 서비스를 호출할 수 있고, 웹 서비스는 다시 X.509 인증서를 이용하여 SOAP 메시지를 암호화와 서명을 하여 응답으로 내보낼 수 있다.

SOAP 메시지는 메시지 안에 모든 보안 관련 정보들을 포함하고 있으므로, 클라이언트와 웹 서비스 사이의 매개체들은 SOAP 메시지에서 필요한 부분만 클라서 처리하게 할 수 있다.

- 기본적으로 JEUS 웹 서비스는 암호화하는 부분을 별도로 지정하지 않을 경우 SOAP 메시지의 Body 부분을 암호화하지만 어떤 부분이든지 암호화할 수 있다. 서명도 마찬가지이며, 이 2가지는 서로 혼용될 수 있다.
26.3.2. 웹 서비스 보안 아키텍처

다음은 JEUS 웹 서비스 보안 아키텍처를 나타내는 그림이다.

[그림 26.1] JEUS 웹 서비스 보안 아키텍처


JEUS Java EE 웹 서비스 클라이언트와 웹 서비스 간의 보안 메시지의 송수신

웹 서비스에 보안을 적용하는 경우 jeus-webservices-dd.xml에 `<security>`를 설정하고 Keystore를 생성해야 한다. 그리고 Keystore에 접근하기 위한 암호나 사용자명 토큰(UsernameToken)에 사용할 암호를 설정하기 위해서 Callback 클래스를 하나 생성해야 한다. 이 애플리케이션을 JEUS 서버에 재배치함으로써 보안 설정이 적용된다.

Java EE 클라이언트는 jeus-web-dd.xml에 `<security>`를 설정해야 하며, 클라이언트의 키를 저장할 Keystore를 생성해야 한다. 사용자명 토큰(UsernameToken)에 사용할 암호와 Keystore의 개인 키를 가져오기 위한 암호를 설정하기 위해서 Callback 클래스를 하나 생성해야 한다.

Java EE 클라이언트 애플리케이션이 실행되면 웹 서비스 클라이언트 런타임은 jeus-web-dd.xml에서 `<security>`의 설정된 값에 따라 보안 로직을 동작시키며. 클라이언트 Keystore에서 개인 키를 가져서 보안을 하거나 서버의 공개 키를 가지고 메시지를 암호화하여 JEUS 서버의 웹 서비스 서버 런타임으로 메시지를 보낸다.

서버 런타임은 jeus-webservices-dd.xml에서 `<security>`에 설정된 값을 바탕으로 보안 로직을 실행시킨다.

- 메시지가 서버에 도달한 경우, 메시지에 포함된 정보를 바탕으로 클라이언트의 공개 키를 사용하여 서버의 Keystore에서 인출하여 서명을 검증한다.

- 메시지가 암호화되어 있을 경우, 메시지에 포함된 암호화 정보를 바탕으로 서버의 Keystore에서 사용하는 개인 키를 인출하여 암호화를 해독한다.

서버에서 웹 서비스 호출이 끝나고 응답을 돌려 보내는 경우 보안을 적용할 때는 위의 과정과 유사하게 진행된다.
웹 서비스는 jeus-webservices-dd.xml에 <security>에 설정된 값으로 Keystore에서 서버의 개인 키나 클라이언트의 공개 키를 꺼내어서 응답 메시지의 일부 또는 전체를 서명하거나 암호화한다.

클라이언트 런타임은 jeus-web-dd.xml의 <security>에 설정된 값을 바탕으로 클라이언트 Keystore에서 서버의 공개 키나 클라이언트의 개인 키를 꺼내어 이를 사용하여 응답 메시지를 검증하고 해독한다.

### JEOS와 타 벤더 간 보안 메시지의 상호 호환 및 운영성

웹 서비스의 보안은 메시지 수준의 보안을 적용하고 사용하게 되어 있다. 보안이 적용된 SOAP 메시지를 내부에는 보안에 관련된 모든 정보가 포함되어 있으며 이러한 정보들이 OASIS의 웹 서비스 보안 표준을 만족하게 될 경우, 여러 벤더 간의 상호 운영성이 보장된다.

JEUS는 OASIS의 보안 표준의 최신 버전인 웹 서비스 보안 표준 1.0을 준수하고 있으며, 따라서 이 표준을 준수하는 벤더 간의 메시지 운송 및 처리를 할 수 있다.

### 26.3.3. JEOUS 웹 서비스 보안 설정

JEUS 웹 서비스와 클라이언트에서 보안 메시지를 송수신하기 위해서는 JEUS 웹 서비스는 jeus-webser-vices-dd.xml에 <security>를 추가 설정하고, 클라이언트는 jeus-web-dd.xml에 <security>를 추가 설정해야 한다.

### 서버 설정


다음은 서버 설정에 대한 예이다.

[예 26.1] <<jeus-webservices-dd.xml>>

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

<security>의 하위 element인 <request-receiver>에 서버가 클라이언트로부터 받는 메시지의 보안 처리를 위해 필요한 정보를 설정해야 한다. <response-sender>에는 서버가 클라이언트로 보내는 메시지의 보안 처리를 위해 필요한 정보를 설정해야 한다.
클라이언트 설정


다음은 클라이언트 설정을 구현한 예이다.

[예 26.2] <<jeus-web-dd.xml>>

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

<security>의 하위 element인 <request-sender>에는 클라이언트가 서버로 보내는 메시지의 보안 처리를 위해 필요한 정보를 설정해야 한다. <response-receiver>에는 클라이언트가 서버로부터 받는 메시지의 보안 처리를 위해 필요한 정보를 설정해야 한다.

26.3.4. 패스워드 Callback 클래스 생성


다음은 패스워드 Callback 클래스를 구현한 예이다.

[예 26.3] <<PWCallback.java>>

```
package jeustest.webservices.wssec.doall;

import com.tmax.ws.security.WSPasswordCallback;
```
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import java.io.IOException;

public class PWCallback implements CallbackHandler {
    public void handle(Callback[] callbacks)
    throws IOException, UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof WSPasswordCallback) {
                WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];
                checkPassword(pc);
            } else {
                throw new UnsupportedCallbackException(
                    callbacks[i], "Unrecognized Callback");
            }
        }
    }

    public void checkPassword(WSPasswordCallback pc) {
        String userId = pc.getIdentifier();
        if (pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN) {
            if (userId.equals("16c73ab6-b892-458f-abf5-2f875f74882e")) {
                pc.setPassword("security2");
            } else if (userId.equals("key_tmax1")) {
                pc.setPassword("keypass_tmax1");
            }
        } else if (pc.getUsage() == WSPasswordCallback.DECRYPT) {
            if (userId.equals("16c73ab6-b892-458f-abf5-2f875f74882e")) {
                pc.setPassword("security");
            } else if (userId.equals("key_tmax1")) {
                pc.setPassword("keypass_tmax1");
            }
        } else if (pc.getUsage() == WSPasswordCallback.SIGNATURE) {
            if (userId.equals("16c73ab6-b892-458f-abf5-2f875f74882e")) {
                pc.setPassword("security");
            } else if (userId.equals("key_tmax1")) {
                pc.setPassword("keypass_tmax1");
            }
        }
    }
}
WSPasswordCallback.getUsage()는 인자로 넘겨 받는 Callback 파라미터가 어떤 경우에 사용되기 위한 것인지에 대한 값을 리턴한다. 사용자명 토큰의 암호를 설정하려면 USERNAME_TOKEN이라는 값을 가지고 있고, 암호화된 메시지를 해독하기 위해 사용할 개인 키에 대한 암호를 설정하려면 DECRIPT 값을 리턴한다. 그 외에도 서명을 하기 위해 필요한 개인 키를 Keystore에서 가져오기 위한 암호를 설정하거나 세션 키를 설정하려 할 경우에도 이러한 구현이 필요하다.

이렇게 생성된 Callback 클래스는 jeus-webservices-dd.xml이나 jeus-web-dd.xml의 <password-callback-class>에 클래스 이름을 설정함으로써 보안에 적용이 가능하다.

26.3.5. JEUS 웹 서비스 서버 보안 적용 예제

스트링을 주고받는 웹 서비스를 보안을 적용은 다음의 과정으로 구현되어야 한다.
1. SOAP 메시지를 암호화하고 서명하는 데 필요한 Keystore 생성한다.
2. Java 클래스 작성한다.
3. 웹 서비스 DD를 작성한다.
4. 패키징과 배치를 한다.

본 절에서는 각 과정에 대해서 설명한다.

26.3.5.1. Keystore 생성

SOAP 메시지를 암호화하고 서명하는 데 필요한 키의 생성이 필요하며 Java에서 제공하는 Keytool을 이용하여 간단하게 구현할 수 있다. keytool 외에도 여러 가지 방법이 있지만, 여기에서는 keytool을 이용하도록 한다. 좀 더 세세한 설명은 keytool의 도움말을 참고한다.

샘플이 존재하는 폴더에서 keygen을 수행하면 다음과 같은 작업을 수행한다.

1. server-keystore.jks라는 Keystore 파일을 생성하고 Keystore의 암호는 keystore_password로 한다. Keystore에는 JEUS_SERVER라는 별칭을 가진 개인 키를 RSA 알고리즘으로 생성하며 개인 키의 암호는 key_password이다. 이때 JEUS_SERVER라는 개인 키의 공개 키도 쌍으로 생성된다.

   keytool -genkey -alias JEUS_SERVER -keyalg rsa -keypass
   key_password -keystore server-keystore.jks -storepass
   keystore_password


   keytool -genkey -alias JEUS_CLIENT -keyalg rsa -keypass
   key_password -keystore client-keystore.jks -storepass
   keystore_password

3. 서버의 Keystore로 사용할 server-keystore.jks라는 Keystore에서 JEUS_SERVER의 공개 키를 가져온다.
4. 클라이언트 Keystore의 client-keystore.jks에 JEUS_SERVER라는 별칭으로 저장한다.

```
keytool -import -file jeus_server.cert -keystore client-keystore.jks -storepass keystore_password -alias JEUS_SERVER
```

5. 클라이언트 Keystore로 사용할 client-keystore.jks라는 Keystore에서 JEUS_CLIENT의 공개 키를 꺼낸다.

```
keytool -export -alias JEUS_CLIENT -storepass keystore_password -keystore client-keystore.jks -file jeus_client.cert
```

6. 서버 Keystore인 server-keystore.jks에 JEUS_CLIENT라는 별칭으로 저장한다.

```
keytool -import -file jeus_client.cert -keystore server-keystore.jks -storepass keystore_password -alias JEUS_CLIENT
```

### 26.3.5.2. Java 클래스 작성

기본적인 웹 서비스 구현은 보안이 적용되지 않았을 경우와 차이점이 없다.

**[예 26.4] << Ping.java >>**

```java
package ping;

public interface Ping extends java.rmi.Remote {
    public String ping(String arg) throws java.rmi.RemoteException;
}
```

**[예 26.5] << PingImpl.java >>**

```java
package ping;

public class PingImpl implements Ping {
    public String ping(String arg) throws java.rmi.RemoteException {
        return arg;
    }
}
```

여기에 추가적으로 개인 키를 꺼내오거나 사용자명 토큰의 암호를 설정하기 위한 Callback 클래스를 생성해야 한다.
package ping;

import com.tmax.ws.security.WSPasswordCallback;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import java.io.IOException;

public class PingPWCallback implements CallbackHandler {
    public void handle(Callback[] callbacks) throws IOException, UnsupportedCallbackException {
        for (int i = 0; i < callbacks.length; i++) {
            if (callbacks[i] instanceof WSPasswordCallback) {
                WSPasswordCallback pc = (WSPasswordCallback) callbacks[i];

                if (pc.getUsage() == WSPasswordCallback.USERNAME_TOKEN) {
                    pc.setPassword("usertoken_password");
                }

                if (pc.getUsage() == WSPasswordCallback.DECRYPT) {
                    pc.setPassword("key_password");
                }

                if (pc.getUsage() == WSPasswordCallback.SIGNATURE) {
                    pc.setPassword("key_password");
                }
            }
        }
    }
}

26.3.5.3. DD 파일 작성

Keystore를 생성하고 Java 클래스 작성이 완료되면 웹 서비스 DD를 작성해야 한다. 다른 부분은 일반적인 웹 서비스 DD 작성과 동일하며, JEUS 특정 설정을 담당하는 jeus-webservices-dd.xml에서만 추가 작업을 한다.

다음은 서버에서 클라이언트의 메시지를 처리하는 부분의 설정이다. 파일 전체의 내용은 샘플을 참고한다.

[예 26.7] << jeus-webservices-dd.xml >>

```xml
...<security>
    <request-receiver>
        <action-list>
```
26.3.5.4. 패키징과 배치

웹 서비스 보안 모듈의 패키징과 배치 방식은 기존의 웹 서비스 모듈의 방식과 크게 다르지 않다. 추가적으로 Keystore(server-keystore.jks)를 컨텍스트의 클래스 경로 안에 위치시켜야 하는 점이 다르다. 또는 jeus-webservices-dd.xml의 <keystore-filename>에 절대 경로를 포함한 Keystore 파일 이름을 적어주는 방법도 있다.

샘플 흘인 pingSecurityService 디렉터리에서 다음과 같이 입력하면 패키징 작업과 패키징된 모듈에 대해 JEOS 서버 배치 작업이 완료된다.

```bash
ant
```

이제 보안이 적용된 웹 서비스를 시작할 수 있다.
26.3.6. JEUS 웹 서비스 클라이언트 보안 적용 예제

JEUS의 웹 서비스 Java EE 클라이언트의 경우도 서버의 설정과 유사하게 일반적인 웹 서비스 클라이언트 생성 작업에 Callback 클래스와 설정 파일 작업, Keystore 패키징 작업이 추가적으로 더해진다.

JEUS 웹 서비스 클라이언트 보안 적용은 다음의 과정으로 구현되어야 한다.
1. SOAP 메시지를 암호화하고 서명하는 데 필요한 Keystore 생성한다.
2. Java 클래스 작성한다.
3. 웹 서비스 DD를 작성한다.
4. 패키징과 배치를 한다.

본 절에서는 각 과정에 대해서 설명한다.

26.3.6.1. Keystore 생성

Keystore는 "26.3.5.1. Keystore 생성"에서 생성한 Keystore 중 client-keystore.jks를 사용한다.

26.3.6.2. Java 클래스 작성

다음은 클라이언트인 pingClient.jsp 파일의 일부이다. 웹 서비스의 보안 적용은 웹 서비스의 포트 단위로 적용되므로 반드시 호출할 포트의 이름을 명시해야 한다.

[예 26.8] << pingClient.jsp >>

```java
... InitialContext jndiContext = new InitialContext(); ...
Service service = (Service)jndiContext.lookup("java:comp/env/service/PingSecurityService");
QName portName = new QName("urn:PingSecurityService","PingPort");
java.rmi.Remote port = service.getPort(portName, Ping.class);
Ping pingPort = (Ping)port;
ret = pingPort.ping(msgToSend);
...```

클라이언트도 서버 설정과 마찬가지로 Callback 클래스를 생성한다. 여기서 사용하는 Callback 클래스는 서버 설정할 때 생성했던 Callback 클래스를 재사용하도록 한다.

26.3.6.3. DD 파일 작성

다른 설정 파일은 일반적인 웹 서비스 클라이언트 DD 파일 작성과 동일하며, JEUS 특성 설정을 담당하는 jeus-web-dd.xml에서만 추가 작업을 한다.
다음은 클라이언트에서 서버로 전송할 메시지를 처리하는 부분의 설정이다. 파일 전체의 내용은 샘플을 참고한다.

[예 26.9] << jeus-web-dd.xml >>

```xml
<security>
  <request-sender>
    <action-list>
      UsernameToken Signature Encrypt
    </action-list>
    <password-callback-class>
      ping.PingPWCallback
    </password-callback-class>
    <user>JEUS_CLIENT</user>
    <signature-infos>
      <signature-info>
        <keyIdentifier>DirectReference</keyIdentifier>
        <keystore>
          <key-type>jks</key-type>
          <keystore-password>
            keystore_password
          </keystore-password>
          <keystore-filename>
            client-keystore.jks
          </keystore-filename>
        </keystore>
      </signature-info>
    </signature-infos>
    <encryption-infos>
      <encryption-info>
        <encryptionUser>JEUS_SERVER</encryptionUser>
        <keyIdentifier>DirectReference</keyIdentifier>
        <keystore>
          <key-type>jks</key-type>
          <keystore-password>
            keystore_password
          </keystore-password>
          <keystore-filename>
            client-keystore.jks
          </keystore-filename>
        </keystore>
      </encryption-info>
    </encryption-infos>
  </request-sender>
</security>
```
26.3.6.4. 패키징과 배치

웹 서비스 Java EE 클라이언트 보안 모듈의 패키징과 배치 방식은 기존의 웹 서비스 Java EE 클라이언트 모듈의 방식과 크게 다르지 않다. 추가적으로 Keystore(client-keystore.jks)를 컨텍스트의 클래스 경로 안에 위치시켜야 하는 점이 다르다. 또는 jeus-web-dd.xml에 <keystore-filename>에 절대 경로를 포함한 Keystore 파일 이름을 적어주는 방법도 있다.

샘플 홈인 pingSecurityServiceJavaeeClient 디렉터리에서 다음과 같이 입력하면 패키징 작업과 JEUS 서버 배치 작업이 완료된다.

```ant
```
패키징하여 배치까지 완료되면 보안이 적용된 웹 서비스 클라이언트를 시작할 수 있다.

26.3.7. 보안 API를 이용한 JEUS 웹 서비스 클라이언트의 작성

JEUS 웹 서비스는 Java EE 환경과 Java SE 환경에서 모두 적용이 가능한 웹 서비스 클라이언트 보안 API를 제공하여 보다 쉽게 웹 서비스 클라이언트 작성을 가능하게 한다.

다음은 API를 이용한 웹 서비스 보안 클라이언트이다.

[예 26.10] << pingClient.jsp >>

```<%@ page language="java" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
<%@ page import="java.rmi.RemoteException" %>
<%@ page import="java.util.*" %>
<%@ page import="javax.naming.InitialContext" %>
<%@ page import="javax.xml.rpc.Service" %>
<%@ page import="javax.xml.namespace.QName" %>
<%@ page import="jeus.webservices.wssecurity.SecurityConfiguration"%>
<%@ page import="jeus.webservices.wssecurity.Keystore" %>
<%@ page import="ping.*" %>
<%@ page errorPage="/error.html" %>

%! String msgToSend = "msg_sent_by_jspClient";
    String ret=null;
    String exceptionString="";
%
```
try {

(1) Keystore keystore = new Keystore ("JKS", "keystore_password", "client-keystore.jks");
Keystore truststore = new Keystore ("JKS", "keystore_password", "client-keystore.jks");

InitialContext jndiContext = new InitialContext();

Service service = (Service) jndiContext.lookup("java:comp/env/service/PingSecurityService");
QName portName = new QName("urn:PingSecurityService", "PingPort");

(2) SecurityConfiguration sconfig =
    new SecurityConfiguration(service, portName);
(3) sconfig.setUsername("JEUS_CLIENT");
(4) sconfig.setRequestPasswordCallbackClass("ping.PingPWCallback");
(5) sconfig.addUTRequest(true, true);
(6) sconfig.addSignRequest(null, "DirectReference", keystore);
(7) sconfig.addEncryptRequest(null, "DirectReference",
    truststore, "JEUS_SERVER", null);
(4) sconfig.setResponsePasswordCallbackClass("ping.PingPWCallback");
    sconfig.addUTResponse();
    sconfig.addSignResponse(truststore);
    sconfig.addDecryptResponse(keystore);

java.rmi.Remote port =
    service.getPort(portName, Ping.class);
Ping pingPort = (Ping) port;

((javax.xml.rpc.Stub) pingPort)._setProperty(
    javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY,
    "http://localhost:8088/PingSecurity="/PingSecurityService";

System.out.println("Send : " + msgToSend);
ret = pingPort.ping(msgToSend);
System.out.println("You received : " + ret);
} catch (Exception e) {
    exceptionString = e.toString();
e.printStackTrace();
}
}%>
위의 예제 중 굵은 글자로 표시된 부분이 웹 서비스 보안 설정에 관한 부분이다.


그 외에 사용자 이름 토큰이나 서명, 암호화에 관한 설정을 추가할 때 각각의 API를 이용하여 추가할 수 있다. 이때 Keystore에 대한 설정이 필요한 경우에는 jeus.webservice.wssecurity.Keystore 클래스를 생성하여 설정이 가능하다.

- (1) Keystore 객체 생성

웹 서비스 보안 API를 적용하기 위해서는 서명 작업이나 암호화 작업의 진행을 위해 Keystore 객체의 생성이 필요하다.

```java
public Keystore(String keyType, String keystorePassword, String keystoreFilename)
```

<table>
<thead>
<tr>
<th>파라미터</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>keystoreFilename</td>
<td>Keystore의 파일 이름을 입력한다.</td>
</tr>
<tr>
<td>keystorePassword</td>
<td>Keystore 파일의 암호를 입력한다.</td>
</tr>
<tr>
<td>keyType</td>
<td>Keystore가 저장하고 있는 키의 타입을 입력한다. 지원되는 키의 타입은 'JKS'나 'PKCS12'들 중에 하나이다.</td>
</tr>
</tbody>
</table>

- (2) SecurityConfiguration 객체 생성

웹 서비스 보안 API를 적용하기 위해 가장 먼저 생성해야 하는 객체이다.

```java
public SecurityConfiguration(javax.xml.rpc.Service service, QName portName)
```

```
throws ConfigurationException
```

<table>
<thead>
<tr>
<th>파라미터</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>service</td>
<td>Java EE 클라이언트에서 작성할 경우에는 JNDI를 통해 얻어오는 Service 객체를 넣어주고, Java SE 클라이언트에서 작성할 경우에는 Service_Impl() 객체를 통해 생성한 객체를 인자로 넣어준다.</td>
</tr>
<tr>
<td>portName</td>
<td>WSDL에 공개된 포트의 QName을 넣어준다.</td>
</tr>
</tbody>
</table>

- (3) 사용자 이름 설정

웹 서비스 보안 요청 메시지를 작성하기 위해서는 사용자명 토큰에 들어갈 이름이나 서명에 사용될 개인 키의 이름이 필요하며 이를 이 함수를 통해 설정할 수 있다.

```java
public void setUsername(String username)
```
• (4) Callback 클래스 이름 설정
웹 서비스 보안 요청이나 응답 메시지를 주고받는 과정 중 사용자명 토큰에 사용되는 암호 설정이나, 개인 키에 대한 암호 설정을 할 경우 패스워드 Callback 클래스들을 생성해야 하며, 생성된 Callback 클래스의 이름을 이 함수들을 사용하여 지정한다.

public void setRequestPasswordCallbackClass(String classname)
public void setResponsePasswordCallbackClass(String classname)

• (5) UsernameToken 처리
다음 함수는 웹 서비스 보안 요청 메시지에 UsernameToken element를 추가할 경우 사용한다.

public void addUTRequest(boolean addNonceCreated, boolean passwordDigest)
패스워드는 평문이나 다이제스트 형태로 생성될 수 있으며, 평문으로 생성할 경우에는 Nonce와 Created 항목을 추가할 것인지 선택할 수 있다.
다음 함수는 웹 서비스 보안 응답 메시지에 UsernameToken element가 있기를 기대하는 경우에 사용 한다.

public void addUTResponse()

• (6) 서명 처리
다음 함수는 웹 서비스 보안 요청 메시지에 특정 부분에 서명할 때 사용한다.

public void addSignRequest(QName signPart, String keyIdentifier, Keystore keystore)
throws SecurityConfigurationException

<table>
<thead>
<tr>
<th>파라미터</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>signPart</td>
<td>null로 설정하면 기본적으로 SOAP 메시지의 body 부분을 기본적으로 서명한다.</td>
</tr>
<tr>
<td>keyIdentifier</td>
<td>다음 중 하나를 입력한다.</td>
</tr>
<tr>
<td></td>
<td>- IssuerSerial</td>
</tr>
<tr>
<td></td>
<td>- DirectReference</td>
</tr>
<tr>
<td></td>
<td>- SKIKeyIdentifier</td>
</tr>
<tr>
<td></td>
<td>- X509KeyIdentifier</td>
</tr>
<tr>
<td>keystore</td>
<td>서명에 필요한 개인 키를 저장하고 있는 Keystore 객체를 입력한다.</td>
</tr>
</tbody>
</table>

• (7) 암호화 처리
다음 함수는 웹 서비스 보안 요청 메시지에 특정 부분을 암호화할 때 사용한다.

public void addEncryptRequest(QName encPart, String keyIdentifier, Keystore keystore, String encryptUser, String algorithm)
throws SecurityConfigurationException
<table>
<thead>
<tr>
<th>파라미터</th>
<th>설명</th>
</tr>
</thead>
<tbody>
<tr>
<td>encPart</td>
<td>암호화하는 SOAP 메시지의 부분으로 QName 형태로 입력한다. 만약 null로 설정될 경우 기본적으로 SOAP 메시지의 body 부분을 기본적으로 암호화 한다.</td>
</tr>
</tbody>
</table>
| keyIdentifier | 다음 중 하나를 입력한다.  
- IssuerSerial  
- DirectReference  
- SKIKeyIdentifier  
- X509KeyIdentifier  
- EmbeddedKeyName |
| keystore | 암호화에 필요한 공개 키를 저장하고 있는 Keystore 객체를 입력한다. |
| encryptUser | 암호화에 사용할 공개 키의 별칭을 입력한다. |
| algorithm | 암호화에 사용될 알고리즘으로 다음 중에 하나를 입력한다.  
- AES_128  
- AES_256  
- TRIPLE_DES  
- AES_192 |

다음 함수는 웹 서비스 보안 요청 메시지에 특정 부분을 암호화하고자 할 때 사용하며, 특히 특정 Callback 클래스에 Byte의 배열로 키를 설정하여 그 키를 이름으로 불러내려 할 때, 이 함수를 사용한다.

```java
public void addEncryptRequest(QName encPart,
    String keyIdentifier, String embeddedKeyCallbackClass,
    String keyName, String encryptUser, String algorithm)
```

다음 함수는 웹 서비스 보안 응답 메시지에 암호화 된 부분이 있기를 기대할 때 사용한다. keystore 파라미터에는 암호화를 풀기 위한 개인 키를 가지고 있는 Keystore 객체를 입력한다.

```java
public void addDecryptResponse(Keystore keystore)
```

[참고]
다음은 본 예제에는 적용되지 않았으나 추가로 사용할 수 있는 기능에 대한 설명이다.

- **TimeStamp 처리**

  다음 함수는 웹 서비스 보안 요청 메시지에 TimeStamp element를 추가할 경우 사용한다.

  ```java
  public void addTimeStampRequest()
  ```

  다음 함수는 웹 서비스 보안 요청 메시지에 유효 기간을 별도로 설정할 경우 사용한다. 초(second)단위이며, 별도로 설정하지 않았을 경우 기본 설정값은 300초이다.


```java
public void addTimeStampRequest(int timeToLive)

다음 함수는 웹 서비스 보안 응답 메시지에 TimeStamp element가 있기를 기대하는 경우 사용한다.

public void addTimeStampResponse()

다음 함수는 웹 서비스 보안 응답 메시지의 TimeStamp element에 설정된 유효 기간이 인자로 입력한 시간 이내여야 한다.

public void addTimeStampResponse(int timeToLive)
```

26.4. 접근 제어 설정

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. 본 절에서는 웹 서비스 접근 제어 설정법과 접근 제어 설정이 된 웹 서비스를 호출하는 방법에 대해 설명한다.

26.4.1. Java 클래스 웹 서비스 접근 제어 설정

 특정한 URL로의 접근을 제한함으로써 웹 서비스에 접근할 수 있으면 접근 제어 설정을 할 수 있으며, 이를 위해서는 web.xml 과 jeus-web-dd.xml 같은 웹 응용 프로그램의 DD에 보안 정보를 추가해야 한다. 또한 보안 도메인의 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.

[예 26.11] << accounts.xml >>

```
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <users>
    <user>
      <name>jeus</name>
      <password>{SHA}McbQlyh13yi0G1HGt8DQVWkyhg=/</password>
    </user>
  </users>
</accounts>
```
DD 파일 작성

jeus-web-dd.xml에 다음과 같이 <role-mapping>을 추가한다.

[예 26.12] << jeus-web-dd.xml >>

```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.

[예 26.13] << web.xml >>

```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'이며 암호가 'jeus'일 경우에만 접근이 허용된다.
26.4.2. EJB 웹 서비스의 접근 제어 설정

EJB 웹 서비스의 접근 제어 또한 Java 클래스 웹 서비스의 접근 제어와 마찬가지로 특정한 URL로의 접근을 제한함으로써 웹 서비스로의 접근을 제한할 수 있다. 이를 위해서는 ejb-jar.xml과 jeus-ejb-dd.xml 같은 EJB DD와 jeus-webservices-dd.xml과 같은 웹 서비스 DD에 보안 정보를 추가해야 한다. 또한 보안 도메인 설정이 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.

[예 26.14] accounts.xml

```xml
<accounts xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
  <users>
    <user>
      <name>jeus</name>
      <password>{SHA}McbQlyhI3yi0G1HG7g8DQVWkyh9</password>
    </user>
  </users>
</accounts>
```

DD 파일 작성

jeus-ejb-dd.xml에 다음과 같이 <role-permission>을 추가한다.

[예 26.15] jeus-ejb-dd.xml

```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>
</jeus-ejb-dd>
```
그리고 ejb-jar.xml에 다음과 같이 보안 관련 설정을 추가한다.

[예 26.16] << ejb-jar.xml >>

```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 다음과 같은 설정을 웹 서비스 DD에 할 수 있으며, 별도로 하지 않을 경우에는 기본적으로 설정된 값으로 수행된다. SSL을 사용하여 주고받는 데이터의 무결성을 보장하려면 ejb-transport-guarantee를 'INTEGRAL' 값을 설정하고, 데이터의 기밀성과 무결성을 모두 보장하려면 'CONFIDENTIAL'로 설정해야 한다. 물론 이 경우에는 HTTPS로 통신을 주고받도록 별도의 HTTP 리스너의 설정이 필요하다.

[예 26.17] << jeus-webservices-dd.xml >>

```xml
<jeus-webservices-dd xmlns="http://www.tmaxsoft.com/xml/ns/jeus">
    <ejb-context-path>webservice</ejb-context-path>
    <ejb-login-config>
        <auth-method>BASIC</auth-method>
        <realm-name>default</realm-name>
    </ejb-login-config>
    <service>
        <webservice-description-name>...</webservice-description-name>
        <port>
            <port-component-name>...</port-component-name>
            <ejb-endpoint-url>...</ejb-endpoint-url>
            <ejb-transport-guarantee>
                CONFIDENTIAL
            </ejb-transport-guarantee>
        </port>
    </service>
</jeus-webservices-dd>
```
26.4.3. 접근 제어가 설정된 웹 서비스 호출

 접근 제어(Basic Authentication)가 설정된 웹 서비스의 호출을 하기 위해서는 WSDL에 접근하여 그 내용을 바탕으로 Stub을 생성하고, Stub을 이용하여 웹 서비스를 호출해야 하는데, 이 경우에 사용자 이름과 암호를 요구할 경우 설정해 주는 작업이 필요하다.

WSDL로부터 Stub 생성

콘솔 툴을 이용할 경우 다음과 같이 설정한다.

```
wsdl2java -gen:client -username jeus -password jeus
```

접근 제어가 설정된 웹 서비스의 클라이언트 작성

JAX-RPC 웹 서비스 클라이언트가 스스로 인증하기 위해서는 다음과 같은 2가지의 설정이 클라이언트 프로그램 내에 삽입되어야 한다.

- javax.xml.rpc.Stub.USERNAME_PROPERTY
- javax.xml.rpc.Stub.PASSWORD_PROPERTY

다음 예는 실제로 프로그램 내에 어떻게 위 설정들이 삽입될 수 있는지를 보여준다.

```
Echo port = // ... 서비스 Endpoint 인터페이스의 획득
((javax.xml.rpc.Stub) port)._setProperty(javax.xml.rpc.Stub.USERNAME_PROPERTY, "jeus");
((javax.xml.rpc.Stub) port)._setProperty(javax.xml.rpc.Stub.PASSWORD_PROPERTY, "jeus");
String s = port.echoString("JEUS");
```
용어해설

DII 클라이언트
  Dynamic Invocation Interface 클라이언트의 약어이다. DII 클라이언트는 JAX-RPC 클라이언트 API에 의해 작성된 웹 서비스 클라이언트이다.

EJB
  Enterprise Java Beans이다.

In/Out Parameter
  In/Out Parameter는 입력과 출력에 사용되는 웹 서비스 오폐레이션의 파라미터이다.

JAX-RPC
  Java API for XML-Based RPC의 약어이다. JAX-RPC는 JCP(Java Community Process)에서 정의한 Java 웹 서비스를 위한 표준 API이다.

JAX-RPC Holder Class
  JAX-RPC Holder Class는 In/Out Parameter(입출력 파라미터)를 지원하는 Java 클래스이다.

JAX-RPC Value Type
  JAX-RPC 값 타입은 웹 서비스의 요청과 응답값에 의해 네트워크로 전송되는 내용의 데이터 타입이다.

JEUS
  Java Enterprise User Solution의 약어이다. 본 안내서에서 소개하는 JEUS 7은 Java EE 6 호환 WAS이다.

Proxy Client
  Proxy Client는 wsdl2java Ant Task에 의해 WSDL 문서로부터 생성된 Stub 코드를 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.
Symbols
@BindingType Annotation, 81, 96
@javax.jws.WebService Annotation, 36, 38
@javax.xml.ws.soap.MTOM Annotation, 96
@jeus.webservices.annotation.EndpointDescription Annotation, 36, 37
@jeus.webservices.jaxws.api.AsyncWebService Annotation, 93
@JMSWebService Annotation

A
Action MAP, 125
AddNumbers 클래스, 29
AddNumbersResponse 클래스, 29
All Operator, 118
Attachment, 273

B
Back-end 구성 요소, 237

C
Content Negotiation 기법, 109

D
DII, 257, 261
DII 클라이언트, 257

E
EWS, 2
ExactlyOne Operator, 118

F
Fast Infoset 표준, 107

H
Holder, 301
Holder 클래스 작성, 310
In/Out Parameter, 301

J
Java API for XML Processing, 227
Java Architecture for XML Binding, 227
Java Transaction API(JTA), 141
Java 스프링 XML 파서, 234
Java 클래스 웹 서비스 Back-end, 8
java.net.Authenticator, 195
javax.activation.DataSource, 77
javax.security.auth.callback.CallbackHandler, 168, 172
javax.xml.rpc.handler API
javax.xml.rpc.handler. GenericHandler, 276
javax.xml.rpc.handler. Handler, 276
javax.xml.rpc.handler. HandlerChain, 276
javax.xml.rpc.handler. HandlerInfo, 276
javax.xml.rpc.handler. HandlerRegistry, 276
javax.xml.rpc.handler. MessageContext, 276
javax.xml.rpc.handler.soap. SOAPMessageContext, 276
javax.xml.soap.SOAPMessage, 276
javax.xml.rpc.Stub.PASSWORDPROPERTY, 340
javax.xml.rpc.Stub.USERNAMEPROPERTY, 340
javax.xml.transform.Source, 77
javax.xml.ws.BindingProvider.PASSWORDPROPERTY, 195
javax.xml.ws.BindingProvider.USERNAMEPROPERTY, 194
javax.xml.ws.handler.LogicalHandler, 67
javax.xml.ws.handler.SOAPHandler, 67
JAX-RPC, 2, 4
JAX-RPC Value Type, 304
JAX-RPC Value 타입, 301, 304
JAX-RPC 매핑 파일, 289, 294
   <exception-mapping>, 296
   <java-wsdl-mapping>, 295
   <java-xml-type-mapping>, 295
   <package-mapping>, 295
   <service-endpoint-interface-mapping>, 297
   <service-interface-mapping>, 296
JAX-RPC 방식, 15
JAX-RPC 웹 서비스 호출, 257
JAX-RPC 웹 서비스의 보안 방식
   메시지 수준 보안, 319
   전송 수준 보안, 319
JAX-WS, 2, 4
JAX-WS JMS 기반 전송, 113
JAX-WS 방식, 15
JAXB, 2, 4, 227
JAXB 기능, 227
JAXP, 227, 234
JAXP의 주요 API
   DOM, 234
   SAX, 234
   StAX, 234
   TrAX, 234
JEUS UDDI Explorer, 203
JEUS UDDI 서버 실행, 202
JEUS 웹 서비스, 7
JEUS 웹 서비스의 구조, 7
jeus-web-dd.xml의 <security>, 323
javax.xml.soap.SOAPMessage, 77

K
Keystore, 333
   keystoreFilename, 333
   keystorePassword, 333
   keyType, 333

M
MAP(Message Addressing Properties), 123
MIME Attachment, 95
MTOM, 2, 95

N
Negotiation Pessimistic 방식, 109

O
Optional Operator, 118

R
RPC 방식, 9

S
SAAJ, 2, 5, 242, 273
Schemagen, 231
SecurityConfiguration, 333
   portName, 333
   service, 333
SEI, 87
Service Endpoint Interface, 87
Service.Mode.Message, 74
Service.Mode.PAYLOAD, 73
ServiceFactory, 265
SJSXP, 227
SOA, 1
SOAP, 2, 3
SOAP Fault, 301, 313
SOAP 메시지 헤더의 생성 방식
   JAX-RPC 방식, 11
   JAX-WS 방식, 10
SOAP 메시지의 인코딩
   RPC 방식, 5
   문서(Document) 방식, 5
SOAP 특성, 3
SOAP 헤더(SOAP Handler), 63
SOAPBody 타입, 275
SOAPBodyElement 타입, 275
SOAPElement 타입, 274
SOAPEnvelope 타입, 274
SOAPHeader 타입, 274
SOAPHeaderElement 타입, 274
SOAPPart, 274
Stateless Session Bean 웹 서비스 Back-end, 8
StAX, 4, 234
Streaming API for XML, 2
STS(Security Token Service), 148
Stub 클라이언트, 257
Sun Java Streaming XML Parser, 227

U
UDDI, 2, 3, 197
UDDI Data 구조, 197
기술 모델(tModel), 198
비즈니스 Entity, 198
비즈니스 서비스(businessService), 198
서비스 바인딩(bindingTemplate), 198
UDDI DataStore, 199
UDDI Deploying, 199
UDDI Explorer, 209
UDDI Property Key
uddi.auth, 200
uddi.dataSource, 200
uddi.operatorName, 200
uddi.operatorURL, 200
UDDI Querying, 203
UDDI Registry, 204
UDDI subscription, 216
UDDI subscription API, 216
UDDI user, 200
UDDI WSDL Publishing, 223
UDDI 서버, 199
UDDI 서버 구성 설정, 200
UDDI 클라이언트, 211
UDDI 클라이언트 XML 서명 검증, 215
UDDI 클라이언트 XML 서명 검증 API
find-relatedBusinesses API, 216
get_bindingDetail, 216
get_businessDetail, 215
get_serviceDetail, 215
get_tModelDetail, 216
UDDI 클라이언트 XML 서명 생성, 214
UDDI 클라이언트 XML 서명 생성 API
add_publisherAssertion, 214
save_binding, 214
save_business, 214
save_service, 214
save_tModel, 214
set_publisherAssertion, 214
UDDI 표준, 4
uddi.properties 파일 설정, 200

W
Web Service Metadata for the Java Platform, 2
web.xml의 <async-supported>, 93
webservices.xml
<handler-class>, 293
<handler-name>, 293
<init-param>, 294
<jaxrpc-mapping-file>, 290
<port-component>, 290
<port-component><handler>, 293
<port-component><port-component-name>, 290
<port-component><service-endpoint-interface>, 290
<port-component><service-impl-bean>, 291
<soap-header>, 294
<soap-role>, 294
<wsdl-file>, 290
WS-Addressing, 2
WS-AtomicTransaction, 2
WS-Coordination, 2
WS-MetadataExchange, 3
WS-Policy, 3
WS-Reliable Messaging, 133
WS-ReliableMessaging, 2
WS-SecureConversation, 3
WS-Security, 3
WS-Security Policy, 3
WS-SecurityPolicy 설정, 177
WSDL, 2, 3, 4
wsdl2uddi, 224
WSDL로부터 웹 서비스 구현, 10
wsi:swaRef 스키마, 101

X
XJC, 228
xmime:expectedContentType, 96
WS-Policy, 3
WS-ReliableMessaging, 2
WS-SecureConversation, 3
WS-Security, 3
WS-Security Policy, 3
WSDL, 3
XML, 2
XML Infoset, 2
XML Namespace, 2
XML Schema, 2
웹 서비스의 전송 수준 보안, 143, 319
일관성, 139

웹 서비스의 전송 수준 보안, 143, 319
정책 연산자(Operator), 118
정책 컨테이너(Policy Container), 117
지속성, 139

코디네이터 서비스, 141

트랜잭션 스펙
  WS-AtomicTransaction, 139
  WS-Coordination, 139

표준 Holder 클래스
  javax.xml.rpc.holders.BigDecimalHolder, 308
  javax.xml.rpc.holders.BigIntegerHolder, 308
  javax.xml.rpc.holders.BooleanHolder, 308
  javax.xml.rpc.holders.ByteArrayHolder, 308
  javax.xml.rpc.holders.ByteHolder, 308
  javax.xml.rpc.holders.CalendarHolder, 308
  javax.xml.rpc.holders.DoubleHolder, 308
  javax.xml.rpc.holders.FloatHolder, 308
  javax.xml.rpc.holders.IntHolder, 308
  javax.xml.rpc.holders.LongHolder, 308
  javax.xml.rpc.holders.QNameHolder, 308
  javax.xml.rpc.holders.ShortHolder, 308
  javax.xml.rpc.holders.StringHolder, 308

프로바이더(Provider) 인터페이스, 73

 yürüt

핸들러 인터페이스 함수
  destroy(), 278
  getHeaders(), 278
  handleFault(), 278
  handleRequest(), 278
  handleResponse(), 278
  init(), 278

색인 347