SOA

자바 웹 서비스로 통하는
서비스 지향 아키텍처

목 차

옮 긴 이 글

장동준

"처음이란 말은 나를 참 설레게 하는 말이다. 그렇지만, 시작이란 말은 나를 참 긴장하게 만든다. 성공이나 실패에 대한 두려움보다 끝을 예측할 수 없기에, 나는 처음 해보는 시작을 좋아한다. 그래서, 흔쾌히 번역에 참여를 하게 됐다. 어느덧 시작에 대한 책임을 질 때가 왔다. 졸역에 대한 독자들의 따끔한 질책과 혹평을 겸허히 받을 준비를 해야겠다. 스터디를 통해 도움을 많이 주신 창신님을 비롯한 모든 공역자 분들께 진심어린 감사를 드리며, 번역을 위해 주말을 흔쾌히 허락해 준 아내 종숙과 아영, 신희에게 심심한 고마움을 전한다."

오픈마루에서 자아의 발현을 위해 노력하는 범인이고, 속내를 볼 수 있는 자바를 좋아한다. XML 문서 관리, 이메일 응대, 검색, 지능형 에이전트, 플랫폼 프로바이더 관련 일들을 하였다.

조지훈

"그리 길지 않은 시간이었는데, 시작할 때와는 많은 것들이 달라졌다. 번역은 영어가 아니라 국어 실력에 좌우된다는 점을 새삼 깨닫게 해준 첫 번역 작업, 존경하는 선배님들과 함께 할 수 있었기에 더욱 의미 있는 추억으로 남는다. 언제나 믿음과 기대로 살펴주시는 선배님들께 부끄럽지만 또한 설레는 마음으로 이 책을 전해드리고 싶다. 그리고 아직은 좋은 아빠, 능력 있는 아빠에도 부족하기만 한 내게 '세상을 사는 이유, 그 자체'가 되어 준 첫 딸 '은'이와 가족들에게 감사의 마음을 전한다."

NHN 웹플랫폼개발팀에서 전사 플랫폼 개발과 자바 관련 기술지원 및 교육을 담당하고 있다. J2eeSTUDY(http://www.j2eestudy.co.kr)를 통해 '사람 냄새나는 따뜻한 개발자 커뮤니티'를 이루려는 꿈많은 개발자이다.

정지웅

"좋은 책, 무엇보다 평소에 존경해오던 분들과 함께 작업하며 많은 것을 배웠던 소중한 경험이었던 것 같습니다. Saas와 같은 낯선 단어들이 새로이 떠오르는 요즘, SOA가 말하는 비전은 이제부터가 본격적인 시작이 아닌가 하는 생각을 해봅니다. 물론 그 중심에는 더욱 성숙해진 자바 웹 서비스가 자리잡고 있을 것이고, 이 책이 가지는 의미도 그런 시작을 알린다는 의미에서 더 각별한 것이 아닐지요. 창신님을 비롯해 가르침 주신 많은 공역자 분들께 감사드리며, 아무쪼록 이 책이 더 즐거운 웹 세상을 만드는데 작은 보탬이 되었으면 하는 바람입니다."

오픈마루 스튜디오에서 즐거운 웹을 만들기 위해 노력하고 있는 개발자다. 주된 관심사는 데이터 중심의 웹 , 정보조직화 등이며, 블로그(http://humbleprogrammer.net/blog)에 관련된 이야기를 풀어놓고 있다. 역서로 '자바 개발자를 위한 레일스'가 있다.

성명식

"읽어보려 마음먹었던 책이었기에 겁 없이 달려들었습니다. 늘 하던 분야이기에 쉽게 생각했었습니다. 고된 작업이었습니다. 하지만, 많은 것을 얻었습니다. 명세가 알려주지 못하는 많은 것들을 이 책에서 볼 수 있어서 제 일에도 많은 도움이 되었습니다. 시간을 내서 다시 한 번 읽어볼 생각입니다. 이 책을 추천해 주시고 이 작업에 동참하도록 이끌어 주신 호경님과 창신님께 감사를 전합니다."

티맥스소프트에서 JEUS 웹 서비스 모듈을 담당하고 있는 개발자이다. Java EE 웹 서비스를 비롯하여 SOA, ESB 등 서비스 아키텍쳐에 관심을 두고 있다.

박호경

"2007년 샌프란시스코에서 열렸던 자바원 컨퍼런스의 한켠에 자리잡은 북쉐프에서 처음 이 책을 발견했었을 때, 최근 웹 서비스 분야에서 많은 개발자들의 관심사와 중요한 변화가 잘 반영되어 있는 아주 알차게 정리된 책이라는 인상을 받았습니다. 이 책은 SOA 시대의 한 축을 담당하고 있는 웹 서비스의 주요 기반 기술들에 대해서 보다 쉽게 설명하고 있으며, 최근 발전되어 온 여러 명세들에 대한 개념, 필요성과 그 적용에 대해 잘 설명하고 있습니다. 이 책을 독자들에게 전달할 수 있게 되어서 아주 기쁘게 생각합니다. 번역에 있어서는, 그간의 현장에서의 개발 경험을 살려 본 책의 내용을 가능한 알기 쉽게, 그리고 내용이 변질되지 않게 전달하려고 노력하였습니다. 앞으로 웹 서비스 기술을 적용 혹은 개발하려는 사람들이 보다 본질적인 내용을 이해하는데 도움이 되었으면 합니다. 오랜 시간 함께 웹 서비스를 연구해 오며, 책의 내용에 대해서 많은 가르침과 의견을 제시해 준 명식님과 창신님께 감사를 전합니다."

티맥스소프트에서 JEUS 웹 서비스 모듈 개발을 담당하였다. Java EE 웹 서비스, 웹 서비스 보안과 웹 서비스 트랜잭션 모듈 개발에 주로 관여하였다.

강규영

"보다 충실한 번역을 위해 원서를 가지고 역자들끼리 사전 스터디를 하였는데, 덕분에 훌륭한 분들과 함께 공부할 기회를 얻게 되었습니다. 개인적으로는 이것만으로도 충분히 보람 있는 일이었습니다. 이 책이 독자 여러분의 공부에 조금이나마 도움을 드릴 수 있다면 더욱 기쁘고 보람찬 일이 되겠지요."

오픈마루 스튜디오에서 스프링노트 개발에 참여하고 있다. 자바스크립트용 행동 주도 개발 프레임워크인 JSSpec, 웹 기반 WYSIWYG 편집기인 Xquared 등의 오픈소스 프로젝트를 리드하고 있으며 현재 개인 위키 http://jania.pe.kr 과 블로그 http://alankang.tistory.com 을 운영하고 있다.

이창신

"수년간 티맥스에서 미쳐(?) 살았던 분야라는 원서의 치명적인 매력 덕분에 근 1년간 이 책의 번역에 매달리고 말았습니다. 「퍼펙트 JSP」 개정판 이래 두 번째로 코디네이터 역할까지 감행할 정도로, 제 개발자 인생의 중간평가 같은 심정이라면 너무 솔직한 거 아닌지 모르겠네요. SOA도 자바 웹 서비스도 이 책을 통해 어렵고 복잡하다는 오명을 씻을 수 있으면 좋겠습니다. 전 동료(박호경, 성명식, 조지훈)와 현 동료(장동준, 정지웅, 강규영)가 함께했다는 것도 특별하기 그지없습니다. 자바, XML, 웹 서비스를 마음껏 토론하고 소통할 수 있었다는 사실만으로, 저는 늘 그렇듯이 팀 모두에게 고마울 뿐입니다. 이렇게 제 10번째 책이 나왔다는 것이 믿기지 않네요. 작년에 참 많이도 다른 분들의 책에 추천사를 썼는데, 제 책이다 보니 참 쑥스럽습니다. 끝으로, 이 책을 내주신 위키북스와 베타리더 이동국님께 충심으로 감사드립니다."

엔씨소프트 오픈마루 스튜디오에서 OTO(Open Technology Officer)로 일하고 있다. OpenID, OAuth, OpenSocial 등 공개 표준 기술에 대한 연구, 구현, 보급에 힘쓰고 있다. http://www.iasandcb.pe.kr

추천 자 서문

과거 마이크로소프트 소속이었던 Pat Helland는 상호운영성interoperability을 논할 때 단어의 앞글자만 따서 즐겨쓰는 약어가 있다. HST, 즉 "싹엮기Hooking Stuff Together"(실은, 그는 중간에 완전히 다른 단어(Shit이다—옮긴이주)를 쓰지만, 이 책은 온 가족이 함께 볼 수 있는 등급이므로 완곡하게 표현했다)이다. 아무리 현란한 용어와 복잡한 도표로 포장해도, 상호운영성은 그냥 "싹엮기"이고, 웹 서비스Web Services도 바로 그런 것이다.

세상에 두 번째 컴퓨터가 온라인에 등장한 이래, 진정한 상호운영성은 여전히 우리에게 미완의 과제를 남기고 있다. IT 환경은 서로 다른 기술의 광대한 배열의 중심에 서서 각자 유용한 기능을 제공하고 있다(거나 적어도 나는 그렇게 들었다). 비록 다양한 업체가 애플리케이션을 구축하는 (그리고 이식하는) 용도로 자신들의 도구를 유일한/지배적인 선택의 자리에 오르게 하려고 애썼지만, IT 세계의 기술들이 통합되기는커녕 더욱 다양해지고 있다. 수많은 솔루션들이 기반 언어, 플랫폼, 운영체계, 하드웨어에 무관하게 'A'프로그램이 'B'프로그램에 말을 걸 수 있게 하는 까칠한 문제의 '정답'임을 자처해왔다. 하지만 '모 아니면 도' 식의 사고방식을 요구하거나 가장 간단한 상황만 대처할 수 있을 뿐, 그 이상은 안 되는 솔루션을 제공하는 등 어떤 것도 완전히 성공적임을 입증하지는 못했다.

1998년, 돈 박스Don Box와 데이브 위너Dave Winer 그리고 마이크로소프트, IBM, Lotus의 몇몇 사람들이 모여 앉아 원격 프로시저 호출remote procedure call, RPC 층을 XML 메시지로 옮기는 아이디어를 기술하는 짧은 문서를 썼다. 착상은 간단했다. 당시 나왔던 모든 분산 오브젝트 툴킷distributed object toolkit—DCOM, 자바 RMI, CORBA가 주된 관심사였던—이 하나의 공통된 네트워크 전송 형식을 공유한다면, 엔터프라이즈 IT 프로그래밍의 성배Holy Grail인 진정한 상호운영성을 이룩하는 것은 간단한 일이 될 거란 것이었다.

처음에는, SOAP이 더 간단하고 나은 HST의 길을 열 것을 약속했다. HTTP 위에서 전달되며, 데이터 표현의 만국공통어인 XML 기반으로, 기존의 분산 오브젝트 프로그래밍 모델의 모든 시멘틱semantic으로 무장된 오브젝트 전쟁의 다크 호스였다. 쉽게 성공할 것처럼 보였다. 분산

오브젝트의 생성된 코드 깊숙이, 아무도 뒤져 보지 않을 곳에 XML을 슬쩍 흘려 놓으면 그만이었다. 당시 우리가 깨닫지 못했던 것은, 안타깝게도 이 전망이 너무도 얄팍했고 무서우리만치 순진했다는 것이다. 확실히 비슷한 (Java와 .NET 같은) 환경에서는 얼추 맞아 들어갈지도 모른다. 하지만, 그런 경우에서조차 XML이 간단히 해결해주지 못하는 차이로 인해 문제가 생기기 쉽다. 소위 표준이라는 것도 사실 어떤 법적 표준 단체의 소산이 아니었다는 사실과 더불어, 업체들은 'WS-어쩌고저쩌고' 명세를 자고 일어나면 찍어내기 바빴고, SOAP이 추구했던 간단명료함은 한마디로 실종되었다. 솔직히, 좀 험하게 말하면, 오랜 동안 전체 웹 서비스 스토리는 '메시지'라기보다는 '메스꺼움'(원어는 mess 엉망진창이었지만 운율을 맞추기 위해 메스꺼움으로 옮겼다—옮긴이주)이 더했다.

이 책의 1장에서 저자 마크 헨센Mark Hensen은 "웹 서비스는 쉽지 않다"고 썼다. 대체 'SOAP'의 'Simple'은 어떻게 된 걸까?

얄궂게도, 웹 서비스가 '저속어dirty word' 상태를 떠안고 출발했지만, EJB와 COBOL과 마찬가지로 그 메시지는 점차 분명해지고 있으며 '바른 말 고운 말'의 기회도 이토록 고조된 적이 없다. (SOAP 1.2 명세와 REST 논문을 제대로 읽어본 사람이라면 누구라도 지적할 수 있는, 진정 논쟁이라고 할 수 없는) SOAP 대 REST 논쟁과 같은 물타기는 차치하더라도, 마침내 여러 업체와 업계 단체가 단지 "나는 너한테 문자열을 넘길 테니 너는 파싱parsing하고…" 라는 것보다 더욱 의미 있는 방식으로 실질적인 HST를 할 수 있는 지점에 이르고 있다.

나는 자바, .NET, XML을 가르쳤던 DeveloperMentor에서 지도자로서, 돈 박스나 마틴 가진Martin Gudgin과 같이 W3C 대표로 SOAP과 스키마 명세를 쓰던 사람들과 SOAP, WSDL, 웹 서비스를 배울 기회를 가졌다. 자바, .NET, 그리고 다른 플랫폼 간의 상호운영성에 초점을 맞춘 업계 컨설턴트로서, 나는 실세계의 상호운영성 문제를 보는 독특한 1인칭 시점을 지니고 있다. 그리고 또한 독립 강연자 겸 멘토로서, 여러 상호운영성 도구를 공부하고 얼마나 잘 돌아가는지 파악하게 된다.

모두 그런 기회를 가지는 것은 아니지만, 나 같이 저수준의 '삽질' 오타쿠여서 밤하늘의 별만큼이나 많은 WS-* 관련 명세를 읽으며 자학적인 쾌락을 추구하지 않는 이상, SOAP이나 WSDL은 불가사의로 남을 것이고, 척 보기에 누구도 마음속에 배울 엄두가 안 나는 지고한 주제로서, 자바 코드를 다른 플랫폼과 통할 수 있게 하는 거구나 하고 감을 잡는 정도가 될 것이다. 괜찮다. 아주 솔직히, 파고 들 필요는 없다. 주어진 프로그래밍 환경을 쓰기 위해 밑바닥부터 죄다 꿰고 있어야만 한다면, 글쎄, 뭔가 잘못된 것이다.

JAX-WS와 JAXB 표준은 웹 서비스 삽질의 그런 모든 저수준의 난맥상을 원하지 않는 한 알 필요 없도록 고안되었다. 마크의 책은 JAX-WS와 JAXB의 얽혀있는 실타래를 푸는 것을 그가

풀어본 경험을 바탕으로 도와줄 것이다. 그는 메시지에 이르기 위해 메스꺼움을 이겨내야만 했고, 이제 그는 뒤돌아서 험난한 여정을 거칠 독자를 안내하려 한다.

종국에는 이 모든 것이 '싹엮기' 이야기가 되기 때문이리라.

—테트 뉴워드Ted Neward, 자바, .NET, XML 서비스 컨설팅, 교육, 강연, 저술 www.tedneward.com

저자 서문

2006년, 자바는 서비스 지향 아키텍처Service Oriented Architecture, SOA를 위한 강력한 플랫폼으로 탈바꿈했다. 2006년 5월 발표된 Java EEEnterprise Edition 5는 애플리케이션 서버상의 웹 서비스 역량을 심대하게 향상했다. 그리고 Java SEStandard Edition 6이 2006년 12월에 공개되어 자바 프로그래밍 언어의 표준 판에 그러한 기능의 대부분을 포용했다.

강건한 웹 서비스 기술은 SOA 구현의 근간이기 때문에, 이제 이 시대의 엔터프라이즈 시스템이 자바 애플리케이션을 SOA 하부구조infrastructure로 통합하는 데 필요한 도구를 자바가 제공하게 된 셈이다.

물론, 자바에는 한동안 기초적인 웹 서비스 기능이 있었다. JAX-RPC 1.0은 2002년 6월에 나왔다. 2003년 11월 최종규격이 나온 J2EE 1.4에는 JAX-RPC 1.1이 있었다. 그렇다면 최신 버전의 자바 웹 서비스Java Web Services, JWS API는 뭐가 그토록 달라진 걸까?

답은 강력함과 편의성이다. 프로그래머는 J2EE 1.4 때보다 Java EE 5에서 엔터프라이즈급 애플리케이션 개발이 훨씬 쉬워졌음을 발견할 것이다. 그 증거가 9장과 10장에 있는데, 거기서 내가 개발한 이베이, 야후 쇼핑, 아마존을 아우르는 온라인 쇼핑 통합 애플리케이션을 선보인다. SOAShopper라고 불리는 이 애플리케이션은 순수 Java EE 5 애플리케이션으로, 여러 쇼핑 사이트의 REST와 SOAP 서비스를 호출한다. SOAShopper는 또한 자체적으로도 SOAP과 REST 엔드포인트를 제공하여 크로스플랫폼 검색과 Ajax 프론트엔드를 제공한다. SOAShopper는 J2EE 1.4와 JAX-RPC로 개발할 때에는 장난이 아니었다. 새로운 자바 웹 서비스 표준으로는, 짜는 것이 즐거움 그 자체였다.

이 책은 새로운 자바 웹 서비스를 구성하는 다음과 같은 표준들에 초점을 맞춘다.

- JAX-WS 2.0 [JSR 224] - Java API for XML-Based Web Services. JAX-RPC의 후계자로서, 자바로 웹 서비스를 만들고 쓸 수 있게 해준다.

- JAXB 2.0 [JSR 222] - Java Architecture for XML Binding. JAX-WS와 긴밀히 연계되어, 자바 오브젝트가 XML로 어떻게 표현되는가를 제어한다.
- WS-Metadata [JSR 181] - Web Services Metadata for the Java Platform. WS-Metadata는 자바 웹 서비스의 유연한 정의와 배포를 돕는 어노테이션annotation을 제공한다.
- WSEE 1.2 [JSR 109] - Web Services for Java EE. WSEE는 Java EE 컨테이너에서의 웹 서비스의 프로그래밍 모델과 런타임 동작을 정의한다.

이러한 표준은 몇 가지 큼직한 발전과 더불어 많은 자잘한 향상을 통해 확실히 더욱 강력한 웹 서비스 프로그래밍 플랫폼으로 올라섰다. 예를 들어 새로운 어노테이션은 웹 서비스 애플리케이션 작성을 더 쉽게 해준다. 그리고 JAX-WS 2.0[JSR 244]에서 자바/XML 바인딩을 JAXB 2.0 [JSR 222]로 위임한 것은 JAX-RPC에 비해 크게 JAX-WS의 사용성을 증대했다. 배포 모델 또한 WS-Metadata [JSR 181]로 매우 간단해지고 나아진 WSEE 1.2 [JSR 109] 버전이다.

1장과 2장은 이런 JWS 표준을 자세히 살펴보고 어떻게 이전의 JWS 표준을 향상시켰는지 설명한다. 3장부터 10장까지는 코드 작성에 주력한다. 새 자바 웹 서비스의 강력함과 편리함을 진정 이해하기 위해서는, 코드를 짜볼 필요가 있다. 그리고 그것이 이 책의 주안점이다. 3장부터 10장까지는 강력한 기능을 최대한 이용하며 함정에 빠지지 않고 일부 한계를 극복하는 방법을 보여주는 예제 코드로 꽉 차있다.

11장은 향후 전망을 바라보며 웹 서비스 개발에 있어 WSDL 중심적인 접근법에 대한 몇 가지 아이디어를 프로토타입과 함께 선사하여 SOA 플랫폼으로서 JWS가 향후 발전할 가능성을 모색해본다.

나는 이 책을 JAX-RPC가 처음 업계에 등장한 2002년에 쓰기 시작했다. 프로그래머를 위한 책을 쓰려했고, 좋은 예제 코드를 JAX-RPC로 짜는 데 애를 많이 먹으면서 곧 난관에 봉착했다. 4년이 지난 후, 글래스피시 Java EE 5 애플리케이션 서버의 시험판을 만지작거리기 시작하면서, 세상이 많이 변했음을 직감했다. 자바로 웹 서비스 프로그램을 짜는 것이 즐거워졌고 다시 이 책의 마무리에 파고 들 수 있었다.

그 결과로 자바 프로그래밍 언어 안으로부터 SOAP, WSDL, 그리고 REST를 다루는 방법을 보여주는 많은 코드로 채워진 책이 나왔다. 이 코드와 부속 설명은 자바 웹 서비스를 완전히 익히고 SOA를 위한 강력한 플랫폼으로 자바를 쓰는 데 도움을 주기를 바라 마지않는다.

이 책에 대해

SOA를 위한 자바 웹 서비스로의 편견 없는 안내

이 책의 주된 목적은 SOA를 위해 자바 웹 서비스JWS를 쓰는 데에 있어 균형 잡힌 안내를 제공하는 것이다. 물론, 어떤 저자도 편견을 가지고 있으며, 나 또한 JWS 표준이 상당히 훌륭하다는 믿음을 실토하지 않을 수 없다. 그렇지 않다면, 이런 책을 썼을 리 없지 않은가.

편견을 실토하고 나니, JWS는 특히 'WSDL과 자바로부터 시작'으로 알려진 개발 방식의 취약함도 솔직히 인정하려 한다. 이 책에서 다른 많은 방식으로 설명되겠지만, JWS 표준은 웹 서비스를 위한 자바 중심적인 방식을 제안한다. 이미 구축된 SOA 표준과 협업하고 기존의 XML 스키마 문서와 WSDL에 자바 애플리케이션을 맞춰야 할 때 그 방식이 발목을 잡을 수도 있다.

그런 상황에서, 웹 서비스 개발을 위한 WSDL 중심적인 방식을 취할 수 있는 것이 해법이 될 수 있다. WSDL 중심적인 영역에서는 JWS가 덜 강력하다. 이 책을 통해, 그 단점을 지적하고 극복할 전략을 소개하려 한다. 심지어 11장에서는 SOA-J라는 이름의 프로토타입 프레임워크를 통해 자바 웹 서비스에 WSDL 중심적인 방식의 대안을 제시한다.

자바 개발자와 설계자architect를 위한 저작

이 책은 코드에 관심이 있는 사람을 위한 것이고, 따라서 주로 시스템을 작성하는 개발자와 시스템을 설계하는 설계자를 대상으로 한다. 내려 받아 설치하고 실행할 수 있는 다량의 코드 예제가 있다.

웹 서비스를 다루는 자바 프로그래머를 위한 책이 되기 위해, 이 책에 담긴 설명과 예제는 독자가 자바에 대한 실제적인 지식과 더불어 XML 및 XML 스키마에 대한 기본적인 이해를 갖고 있음을 가정하고 있다. 이 책을 시작하기 위해 SOAP과 WSDL을 많이 알고 있을 필요는 없다. 하지만 책을 읽어 감에 따라, 웹 서비스 기초에 대해 탄탄한 개념을 잡고 싶을 때 WSDL과 XML 스키마에 대한 입문서를 훑어 볼 필요가 있을 수 있다. 이 책을 통해, 복습을 할 수 있는 참고 자료를 제공할 것이다.

J2SE 5.0의 숙지가 바탕

이 책은 독자가 J2SE 5.0, 특히 제네릭스generics와 어노테이션과 같은 자바 언어 확장을 기본적으로 이해하고 있음을 가정하고 있다. 만약 그렇지 않다면, http://java.sun.com 에 있는 무료 문서와 입문서를 통해 필요한 것을 다 배울 수 있다.

이런 주제에 대해 겁먹을 필요는 없다. 제네릭스와 어노테이션은 숙달하기 어렵지 않으며, Java EE 5와 SE 6로 웹 서비스를 할 때 알아야만 한다. 이 책의 일부 지면을 할애하여 제네릭스와 어노테이션에 대한 안내를 쓰지 않은 이유는 웹에 이미 좋은 무료 정보가 널렸기 때문이다. 내가 이 책에서 목표로 삼은 것은 온라인 입문서나 문서로 무료 제공되는 것 이상을 제공하는 것이다.

어째서 글래스피시인가?

이 책의 모든 예제 코드는 글래스피시 [GLASSFISH] 오픈소스 Java EE 5 참조 구현체reference implementation를 써서 개발과 테스트를 했다. 집필할 당시, 글래스피시는 사용 가능한 구현체로 유일했다. 인쇄가 들어갈 무렵에는 구현체가 더 많아졌고, 예제는 수정 없이 모든 플랫폼에서 동작해야 한다. 바꿀 곳이 있다면 글래스피시에 특화된 도구(예를 들어 WSDL에서 자바로 컴파일하는 `wsimport`나 `asadmin` 배포 도구)가 쓰이는 빌드 과정 정도이다.

JBoss, BEA, IBM 등의 벤더가 JWS 표준을 지원함에 따라 예제 코드를 실행할 수 있는 안내를 제공할 예정이다. 이 책의 공식 사이트 http://soabook.com 에서 최신 소식을 확인하기 바란다.

글래스피시를 써본 적이 없다면, https://glassfish.dev.java.net 을 통해 한번 써보기를 권한다. 글래스피시는 Java EE의 첨단을 걷고 있으며 커뮤니티도 대단하다. 특히, 메일링 리스트를 통해 받은 기술 지원이 좋았다. 질문을 올렸더니 JSR 명세 리더가 몇 분 내에 답해주는 것은 흔한 일이 아니다!

이 책에서 다루지 않는 주제들

SOA와 웹 서비스 둘 다 광범위한 주제이다. 논의를 자바 기술로 한정해도, 한 책에 전부 다루는 것은 불가능하다. 이런 현실에 직면하다 보니, 자바 개발자와 설계자에게 중요한 핵심 사안이라고 여겨지는 것에 집중하기로 했다. 느슨하게 결합되는 SOA 애플리케이션으로 분해

할 수 있는 식으로 웹 서비스를 만들고, 배포하고, 호출하는 일이 핵심 사안과 관련되어 있다.

책의 초점을 맞추기 위해, 일부 독자를 실망시키게 되는 것을 피할 수 없는데 몇몇 흥미로울 수 있는 주제를 다루지 않았기 때문이다. 이 책의 검토자들이 지적해준 그런 주제들을 내가 왜 다루지 않았는지 그 이유와 함께 여기 나열해보았다.

SOA 설계 방침

SOA의 이면에 있는 개념과 설계 철학을 이 책은 다루지 않는다. 이 책은 자바 웹 서비스로 SOA 컴포넌트를 짜는 법을 자바 개발자에게 가르쳐주는 방법서이다. SOA 개념과 설계에 대한 완전한 입문서로, 토마스 얼Thomas Erl의 Service-Oriented Architecture [Erl]을 추천한다.

UDDI

UDDI는 매우 중요하다. 그리고 Java EE 5는 UDDI 저장소에 대해 JAXR 표준 인터페이스를 제공한다. 그러나 JAXR은 J2EE 1.4 이래로 바뀌지 않았다. 게다가 다른 많은 책과 온라인 입문서에서 잘 다루어져 있다. 그래서, 이 책을 감당할 수 있는 크기로 유지하게 위해, 과감히 생략했다.

엔터프라이즈 메시징Enterprise Messaging

엔터프라이즈 메시징에 대한 장을 넣었으면 하는 마음은 있었다. 어찌되었건 SOA의 기본 토대이기 때문이다. 하지만, 이 책의 범위를 JWS가 제공하는 기능으로 한정했다. JWS는 WS-ReliableMessaging [WS-RM]이나 어떤 다른 SOAP/WSDL 지향적인 신뢰적 메시징 방법을 지원하지 않는다. 물론, Java EE 5에는 Java Message Service API JMS가 들어있다. 그리고 JMS는 SOA 애플리케이션을 구현하는 데에 유용한 도구이다. 그러니 그 자체로서 JMS는 웹 서비스 도구가 아니다. 그래서 초점을 흐리지 않기 위해 역시 뺐다.

WS-Addressing, WS-Security, 그리고 다른 많은 WS-* 표준들

웹 서비스 기반을 위해 고안된 수많은 표준을 설명하려면 수천 페이지가 들 것이다. WS-* 표준은 JWS에 속하지 않기 때문에, 따라서 다루지 않았다. 거기에, 내 생각에는 대부분의 자

바 개발자는 여전히 HTTP 상의 SOAP을 떼고 있는 중이다. 프로그래머를 위한 WS-* 안내서의 시장은 아마 수년 뒤의 일일 것이다.

글꼴과 특수 문자

Courier 글꼴은 자바 타입, XML 스키마 컴포넌트, 그리고 본문 중의 모든 예제 코드를 표시하는 데 쓰인다. 예를 들어

`java.lang.String` - 자바 클래스(완전 수식)

`MyPurchaseOrder` - 자바 클래스

`xs:string` - XML 스키마 타입

`po:billTo` - XML 스키마 전역 요소

Courier 글꼴은 또한 소프트웨어 환경에 특화된 항목(즉 경로, 디렉토리, 환경 변수등)과 컴퓨터와의 문자 입출력을 강조하는 데에 쓰인다. 예를 들어

`JAVA_HOME` - 환경 변수

`$JAVA_HOME/bin` - 디렉토리

`mvn install` - 콘솔에 입력할 명령

〈〉은 환경에 특화된 디렉토리 위치를 가리킨다. 예를 들어

`<AppServer>` - Java EE 5 애플리케이션 서버가 설치된 곳의 위치

`<book-code>` - 책의 예제가 설치된 곳의 위치

본문 중의 부분 코드

본문에는 논의를 전개하기 위해 많은 부분 코드가 있다. 각 부분 코드의 밑에는 그 코드가 어디에 있는지를 나타내는 파일 경로가 있다. 그래서 예를 들어 아래 부분 코드는 `<book-code>/chap03/eisrecords/src/xml/order.xml`에서 왔다는 식이다. 더욱이, 왼쪽의 줄 번호는 그 코드가 파일의 어느 부분인지를 보여준다.

```
4  <Order xmlns="http://www.example.com/oms"
5    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6    xsi:schemaLocation="http://www.example.com/oms
7    http://soabook.com/example/oms/orders.xsd">
```



```
8   <OrderKey>ENT1234567</OrderKey>
9   <OrderHeader>
10    <SALES_ORG>NE</SALES_ORG>
11    <PURCH_DATE>2005-12-09</PURCH_DATE>
12    <CUST_NO>ENT0072123</CUST_NO>
13    <PYMT_METH>PO</PYMT_METH>
14    <PURCH_ORD_NO>PO-72123-0007</PURCH_ORD_NO>
15    <WAR_DEL_DATE>2005-12-16</WAR_DEL_DATE>
16  </OrderHeader>
17  <OrderItems>
18    <item>
19      <ITM_NUMBER>012345</ITM_NUMBER>
20      <STORAGE_LOC>NE02</STORAGE_LOC>
21      <TARGET_QTY>50</TARGET_QTY>
22      <TARGET_UOM>CNT</TARGET_UOM>
23      <PRICE_PER_UOM>7.95</PRICE_PER_UOM>
24      <SHORT_TEXT>7 mm Teflon Gasket</SHORT_TEXT>
25    </item>
26    <item>
27      <ITM_NUMBER>543210</ITM_NUMBER>
28      <TARGET_QTY>5</TARGET_QTY>
29      <TARGET_UOM>KG</TARGET_UOM>
30      <PRICE_PER_UOM>12.58</PRICE_PER_UOM>
31      <SHORT_TEXT>Lithium grease with PTFE/Teflon</SHORT_TEXT>
32    </item>
33  </OrderItems>
34  <OrderText>This order is a rush.</OrderText>
35 </Order>
```

book-code/chap03/eisrecords/src/xml/order.xml

감사의 글

많은 유능한 분들의 도움과 지원이 아니었다면 이 책의 저술을 마칠 수 없었을 것이다. 특히, 글래스피시 프로젝트 커뮤니티의 모든 분께 이 책을 통해 드러나는 값진 통찰력을 제공해준 데에 감사 드리며, Stephen DiMilla, Jerome Dochez, Joseph Fialli, Mike Grogan, Doug Kohlert, Kohsuke Kawaguchi, Jitendra Kotamraju, Bhakti Mehta, Carla Mott, Dhiru Pandey, Vivek Pandey, Dinesh Patil, Eduardo Pelegri-Llopart, Vijay Ramachandran, 그리고 Kathy Walsh에 각별한 정을 표하고 싶다. 그중 썬 마이크로시스템스의 Vijay Ramachandran과 Doug Kohlert에는 WS-Metadata, WSEE, 그리고 JAX-WS 관련 부분을 검토해준 데에 더욱 감사 드린다.

웹 서비스를 이용한 프로세스와 데이터 통합에 대한 연구를 수행했던 MIT에서 교환 교수로 Stuart Madnick 교수로부터 초청을 받았던 시절 이 프로젝트(이 책의 집필)을 처음 구상했다. 그를 비롯한 그의 연구 팀과의 협업은 자바 웹 서비스에 대한 내 관심에 불을당겼고, 마침이 이 책으로 결실을 본 셈이다.

Bruce Scharlau, Art Sedighi, 그리고 Matt Anderson은 초고를 검토해주며 많은 유익한 조언을 해주어 이 책으로 승화했다.

11장에 설명된 SOA-J 프로젝트에 값진 기여를 해주었던 인도 뱅갈로에 있는 내 친구들인 Kishore Gopalakrishna와 그의 팀원인 Rohit Agarwal, Vinit Sharma, 그리고 Rohit Choudhary에게도 감사를 표하고 싶다.

Ted Neward는 통찰력 넘치는 비평을 해주었으며 고맙게도 추천문을 써주었다. 이 책을 통해 그와 함께 한 것은 대단한 영광이다.

이 책은 Prentice Hall 출판사의 편집자 Greg Doench로부터 인고의 지도가 없었다면 결코 나올 수 없었다. 그의 지혜와 경험은 값으로 따질 수 없다. 또한 이 책을 출판의 길로 인도해준 Michelle Housley, Julie Nahil, Dmitri Korzh, 그 밖의 모든 모든 출판사 스태프들에게 감사 드리고 싶다.

집으로 돌아와, 내 아이들, Elizabeth, Eric, 그리고 Emily는 책을 쓰는 동안 잦은 포옹과 장난기 어린 방해로 나에게 힘을 주었다. 마지막으로 가장 중요한, 나의 아내 Lorraine의 사랑과 격려가 이 책을 가능하게 했다. 아내의 인내와 이해가 없었더라면, 이 작업은 결코 마칠 수 없었다.

저자에 대해

마크 D. 한센Mark D. Hansen 박사는 소프트웨어 개발자이자 컨설턴트이며 기업가이다. 그의 회사인 Javector Software는 웹 서비스에 특화된 컨설팅과 소프트웨어 애플리케이션 개발을 제공한다. 마크는 또한 글래스피시 프로젝트의 컨텐츠 개발자이며 WSDL 중심적인 웹 서비스 개발을 위한 SOA-J 오픈소스 애플리케이션 프레임워크를 개발해오고 있다.

이전에, 마크는 웹 서비스 기술을 통한 프로세스와 데이터 통합을 위한 MIT 연구 애플리케이션 팀의 방문 교수였다. 그 전에는 eBusiness 컨설팅 서비스의 손꼽히는 제공사 중 하나인 Xpedir, Inc. 의 책임 부사장을 역임했다.

마크는 포춘 500대 기업을 대상으로 맞춤 인터넷 솔루션을 개발을 위해 1993년 Kinderhook을 창립했다. Kinderhook Systems의 설립 전에는, 기업형 데이터 웨어하우스에서의 데이터 정합성 관리용 도구를 제공하는 매사추세츠주 캠브리지에 근거한 소프트웨어 회사인 QDB Solutions, Inc.의 창업자이자 부사장으로 일했다. QDB Solutions는 1997년 Prizm Technologies에 합병되었다.

마크의 저술은 Wall Street Journal, Information Week, Computer World, Database Management, Database Programming and Design, Business Communications Review, EAI Journal, 그리고 IntelligenceEnterprise와 같은 간행물에 실렸다.

마크는 코넬 대학에서 수학 학사 학위를, 시카고 대학에서 수학 석사 학위를, MIT 경영대학원에서 석사 학위를, MIT 컴퓨터 과학 연구소에서 박사 학위를 취득했다.

마크와 그의 아내 로렌은 뉴욕 근거에 3명의 자녀 엘리자베스, 에릭 그리고 에밀리와 함께 거주하고 있다

01장.
자바 웹 서비스와 함께하는 서비스-지향 아키텍처

요즈음의 엔터프라이즈 자바 애플리케이션들은 너 나 할 것 없이 모두 서비스-지향 아키텍처 Service-Oriented Architecture, SOA를 지원해야 하는 요구에 맞닥뜨리게 된다. 대부분의 SOA 애플리케이션의 근본은 웹 서비스다. 그렇기 때문에, 엔터프라이즈 자바 개발자라면 Java EE 5와 Java SE 6에 포함된 웹 서비스 표준들에 능숙해지고 싶은 욕구를 가지고 있으리라 생각된다. 이 표준에는 JAX-WS(이전에는 JAX-RPC였던) [JSR224], JAXB [JSR222], Web Services Metatdata WS-Metadata [JSR181], SOAP with Attachment API for Java (SAAJ) [JSR67], 그리고 Web Services for Java EE WSEE [1] [JSR 109] 등이 포함된다. 이제부터 이 표준들을 한데 묶어 자바 웹 서비스 Java Web Service, JWS라 부르도록 하겠다.

SOA 애플리케이션은 느슨하게 결합된loosely coupled 웹 서비스들로 구성된다. 우리가 엔터프라이즈 자바 개발자이니만큼 SOA 애플리케이션을 개발하는데 JWS 도구들을 사용하고자 한다. 더구나, 주요 엔터프라이즈 자바 벤더들이 SOA 애플리케이션을 위한 개발 플랫폼으로써 JWS 기술들을 지지하고 있는 상황이기도 하다.

그래서 JWS 표준이 매우 중요한 것이다. 이 표준들은 엔터프라이즈 자바환경에서 SOA 개발을 하는 데 기초가 된다. 느슨하게 결합된 SOA 애플리케이션을 통해 비즈니스 프로세스는 보다 유연하게 만들어진다. 이 결과로, 기업은 급속히 변화하는 세계시장에 보다 잘 적응할 수 있게 되므로, 결국 이 모든 것은 기업 차원의 경쟁력 제고에 있어서도 매우 중요한 역할을 차지하게 된다.

하지만 불행히도, 내 경우와 비슷한 전철을 밟게 된다면 자바 웹 서비스의 학습곡선이 다소 가파르다는 사실을 금새 알게 될 것이다. 단순히, 자바 클래스를 웹 서비스로 배포하거나 그러한 서비스들을 소비할 수 있는 간단한 클라이언트를 생성하는 데에도 많은 종류의 강력하고 복잡한 기술들이 요구된다. 물론, Java EE 튜토리얼에 간단한 "Hello World" 애플리케이션이 나와 있긴 하다.

---

1) EE 컨테이너에 대한 배포 표준인 WSEE는 Java SE가 아닌, Java EE에서만 지원된다.

그러나 구매 주문 시스템 같은 것을 배포해야 할 필요가 있다면 일은 좀 더 복잡해진다. WSDL에서부터 시작하는 경우라면, 실제 구매 시스템에 수동으로 패키징 되고 매핑되어야 하는 온갖 종류의 기괴한 클래스들이 컴파일 될 것이며, 자바 클래스들에서부터 시작하는 경우에도 WSDL은 여러분이 원하는 방식대로 생성되지 않을 것이다. 만약 여러분이 이런 종류의 문제들로 당황해본 적이 있다면, 나는 그 심정을 충분히 이해한다 말할 수 있다. 나 또한 그래왔고, 그것이 내가 이 책을 쓰게 된 직접적인 동기가 되었기 때문이다.

1.1 내가 멍청한 걸까? 아니면 자바 웹 서비스가 진짜 어려운 걸까?

처음엔, 그저 내가 멍청하다고 생각했다. 자바 웹 서비스에 관계된 일을 하기 전까지 나는 컨설팅 업체를 경영하고 있었다. 지난 수년 동안 관리자로서 일해왔기 때문에, 기술자로서의 두뇌는 좀 굳어져 있는 상태였다. '계속 그 일을 해!' 나는 거듭해서 내 자신에게 말했다. '그러면 결국 이 기술에 통달할 수 있을 거야.' 그게 대략 3년 전쯤의 일이었고, 나는 JWS 어노테이션과, 배포 기술자deployment descriptors, WSDL 프록시proxy, 스키마 컴파일러schema compiler 같은 것들과 씨름해왔으며, 그러는 가운데 배운 경험들을 바탕으로 이 책을 펴내기에 이르렀다.

지난 3년간, 나는 자바 제네릭스Generics, 리플렉션Reflection, 영속성persistence, 병행성concurrency과 같은 주제들을 습득했다. 아파치 Axis 소스코드를 공부했으며, 몇 개의 패치를 제출하기도 했다. 그리고선 이내, 내 자신이 그리 멍청하지는 않다는 사실을 깨달을 수 있었다. 여태껏 JWS 표준들에 대한 직관적인 이해를 향상시키려 했던 노력들은, 내게는 긴 투쟁의 연속이나 마찬가지였다. 그리고 이런 경험을 한 사람이 비단 나 하나만이 아니었다.

저명한 학자인 리처드 몬슨-헤펠Richard Monson-Haefel은 2003년 말에 자바 웹 서비스 표준의 J2EE 1.4 버전에 기초한, 960쪽짜리 책을 출간했다. 무려 960쪽! 이 사실 하나만으로도, 우리는 JWS와 관계된 학습 곡선에 대한 의미심장한 결과를 파악할 수 있다. 어떤 특정 주제가 매우 어려운 것이 아니고, JAX-WS API 자체가 무엇을 하는지 파악하는 것도 그다지 어렵지 않다. 그러나 진정 어려운 부분은 이 모든 API들이 기반 웹 서비스 표준(예를 들어 XML, WSDL, SOAP 같은), HTTP 프로토콜 그리고 다른 Java EE 컨테이너 서비스(예를 들면, 의존성 주입Dependency Injection)와 어떻게 연관되는지를 이해하는 일일 것이다. 기반이 되는 WSDL, SOAP, XML 그리고 HTTP 등이 자바 표준 가운데서 어떻게 녹아들었는지 연결 지어보려 하면 할수록 JWS 표준을 사용해서 작업한다는 사실 자체가 어색하고 부자연스럽게 느껴지기 마련이다.

지난 2년 동안, 학자들은—몬슨 헤펠도 그들 중 하나였다—자바 웹 서비스 표준을 맹렬히 비난했다. 그들의 관점은, 구버전의 JWS인 J2EE 1.4 버전상에서 작업했던 그들의 경험들에서 비롯된 것이다. 나 또한 그런 예전 버전의 API로는 유용한 SOA-스타일의 개발이 힘들다는데

동의한다. 하지만, 특별히 Java EE 5와 Java SE 6에 내장된 최신 버전을 놓고 볼 때, 나는 표준 자체가 문제라는 사실에 대해서는 동의하기 어렵다. 대신에 자바 웹 서비스 개발을 위한 범용적인 프레임워크를 생성하는 문제 자체가 원래 복잡한 것이 원인이 아닌가 의심해보고 싶다.

리처드 몬슨-헤펠은 2006년 4월 22일에 아래와 같은 이메일 내용을 그의 블로그에 포스팅한 바 있다. 이 글은 우리 가운데 얼마나 많은 사람들이 자바 웹 서비스에 많은 시간을 소모하고 있는지를 단적으로 잘 나타내는 비유라 할 수 있다.

이 책의 명제 중 하나는, 간단히 말해 웹 서비스가 근본적으로 어렵다는 것이다. 우리는 이 사실을 인정하고 넘어가야 한다. 웹 서비스는 분산 컴퓨팅에 기반하기 때문에 어려운 것이고, 분산 컴퓨팅은 컴퓨터과학에 있어서도 가장 어려운 문제이기 때문이다.

데이브 포드너의 웹 서비스를 대하는 다섯 가지 단계

1. 부정—단순한 객체 접근 프로토콜Simple Object Access Protocol, SOAP이군. 그렇지?
2. 난처함—좋아, SOAP, WSDL, WS-I BP, JAX-RPC, SAAJ, JAX-P 등의 표준에 대해 읽어봐야겠어. 그 다음에 위키를 확인하고, 마지막으로 서비스와 클라이언트 측을 다룬 예제들을 따라해 봐야지.
3. 분노—이것들이 왜 그리 어렵게 만들어졌는지 도무지 알 수가 없어!
4. 죄책감—모두들 웹 서비스를 잘 사용하고 있는데, 분명 내 탓일 거야. 내가 뭔가 실수하고 있는 게 틀림없어
5. 납득—웹 서비스는 단순하지도, 쉽지도 않은 것이군.

그래서 이 책은 JWS 표준을 사용하면 SOA 스타일의 애플리케이션을 쉽게 만들 수 있다는 따위의 과장광고는 하지 않는다. 대신에, 이 책은 JWS를 살펴보고 이 컴포넌트 기술들의 장점과 단점들을 이해할 수 있도록 도와준다. 그 과정에서 어떻게 JWS가 웹 서비스를 효율적으로 배포deploy하고 소비consume할 수 있는 강력한 SOA-스타일의 애플리케이션을 만드는 데 사용될 수 있는지에 대해, 내가 여태껏 배워온 것들을 여러분과 공유할 것이다. 이 여행의 대미는 9장과 10장에서 다루게 되는 이베이, 아마존, 그리고 야후! 쇼핑과 통합된 쇼핑엔진을 구현한 SOAShoper 예제 애플리케이션을 구축하는 대목에 있다. SOAShopper는 REST와 SOAP 엔드포인트 둘 모두를 발행publish하고, REST와 SOAP 엔드포인트를 소비하며, Ajax 프론트 엔드를 제공한다.

1.1.1 기술에 쉽게 현혹되지 말라

2001년 초반 Ariba, IBM 그리고 마이크로소프트가 W3C 노트W3C Note로서 WSDL 1.1을 발표했을 때만 해도, 웹 서비스는 분산 컴퓨팅을 보다 쉽게 만들어 줄 수 있는 방법으로써 계획되었다. 개발자들은 더 이상 크로스 플랫폼Cross Platform 분산 애플리케이션을 생성하기 위해 CORBA 표준 같은 것을 이해할 필요가 없을 것으로 보였다. 더 나아가, 웹 서비스는 분산 컴퓨팅을 인터넷상에 존재하는 모든 대상에 적용할 수 있는 기술로써 기대를 받았다.

나처럼 대부분의 자바 개발자들은, 이러한 초기의 웹 서비스 비전들을 신봉했다. 1990년대에 우리가 인터넷을 통해 경험한 것에 따르면 그 비전들은 충분히 납득이 가는 것이었기 때문이다. 일찍이 HTTP상에서 동작하는 HTML이 대중을 위한 분산 컴퓨팅 플랫폼인 월드 와이드 웹의 놀라운 성장에 원동력이 되었던 바 있다. HTTP상의 XML을 위한 표준들도 이와 마찬가지로 웹 서비스—비즈니스 애플리케이션을 위한 분산 컴퓨팅 플랫폼—성장의 원동력이 되어 줄 것으로 믿었다.

결국, 우리는 모두 그 환상에 매혹되고 말았다. 웹 서비스가 분산 컴퓨팅을 보다 쉽게 만들어줄 것이라 믿었던 것이다.

엔터프라이즈 자바 업계의 리더들은 웹 서비스의 비전을 구현하는 일에 착수했다. 이 비전을 실현하는 과정에서, 자바 업계는 그들이 끔찍한 표준들을 몇 개 만들어냈다는 사실을 발견했다. 초기의 JAX-RPC, JAXB나 다른 표준들을 읽어본 사람들은—나를 포함해서—점차 불안해졌다. 무언가가 잘못 돌아가고 있다는 걸 느낄 수 있었다. 이 표준을 주도하고 있는 전문가 그룹이 뭔가 궤도를 이탈했다고 추측했다. 웹 서비스가 약속했던 것들이 사라지는 것에 대해 점차 씁쓸함과 환멸마저 느끼게 되었다. 우리들 사이에서도 SOAP과 REST에 대한 논쟁이 시작되었고, 이런 자바 웹 서비스 표준의 복잡성이 누구의 탓인지를 두고 설왕설래가 계속되었다.

하지만, 비단 복잡성 때문에 SOAP 대신 REST를 선택하는 결과가 벌어지는 것은 아니다. 그렇다고, 전문가 그룹이 기술을 과용over engineering한 탓도 아니다. 전문가 그룹은 정석을 따랐을 뿐이고, 웹 서비스의 비전을 실현하려고 노력했을 뿐이다. 결국 그들은 분산 컴퓨팅이 진정 어려운 문제라는 사실을 재발견해낸 셈이 되었다. SOAP, WSDL, XML, 그리고 심지어 REST조차도 결코 분산 컴퓨팅을 쉽게 만들어주진 못했다.

물론, JWS 표준에는 결함이 있었다. 하지만 그건 이미 예견된 결과였을 뿐이다—새로운 기술은 종종, 다루기 힘든 (EJB를 보라) 급격한 변화와 특이점을 동반하기 마련이다. 이 문제들은 다음 버전의 [2] 개선점을 통해 수정되기 마련이다.

2) EJB 3.0은 비난세력들이 요구해왔던 진보된 기능들, 가령 관점 지향 프로그래밍(Aspect Oriented Programming)과 제어 역전(Inversion Of Control)과 같은 것들을 구현하면서 계속해서 발전을 거듭하고 있다.

JWS 명세Specification가 얼마나 개선되었는지를 보여주는 일례로 JAX-WS 2.0을 들 수 있다. 6장과 7장에서 이 명세를 자세히 다루고 있기 때문에, 지금은 이것이 JAX-RPC 1.1에 비해 어떻게 큰 개선을 가져왔는지에 대해 맛보기 정도로만 살펴보기로 하자. 첫 번째, JAX-RPC 데이터 바인딩Data Binding은 삭제되었고, 명세는 REST 엔드포인트를 지원하는 것과 동시에, WSDL과 자바 간의 매핑에 초점을 맞추는 방향으로 좀 더 단순화되었다. 자바 데이터 바인딩은 JAX-RPC에 있던 XML 스키마로부터 더 개선되고 널리 쓰일 수 있는 JAXB 2.0으로 대체되었다. 두 번째, JAX-WS는 어노테이션을 통해 자바 인터페이스로부터 생성되는 WSDL의 형태를 제어할 수 있게 해준다. 이런 어노테이션의 사용을 통해 JAX-RPC 서비스를 배포할 때 필요하던 배포 기술자를 좀 더 적게 사용할 수 있다. 세 번째로, JAX-WS는 프로그래머가 XML에 직접 작업할 수 있는 인터페이스(클라이언트 측은 `Dispatch`, 서버 측은 `Provider`)를 제공한다—사용하고 싶지 않을 때는 효율적으로 JAXB 데이터 바인딩 자체를 생략할 수도 있다.

확실히, JAX-WS 2.0에는 아직 더 개선의 여지가 남아 있다. 내가 생각할 수 있는 가장 큰 개선점은 개발자가 WSDL과 관련된 스키마에 명시된, 네이티브 XML 타입을 직접 다룰 수 있도록 하는 다른 바인딩 방법(JAXB 외에 추가적으로)을 제공하는 것이다. XJ [XJ] 같은 자바의 XML 확장들이 그런 역할을 할 수 있다. 개발자들이 JAX-WS로 작업하면서 경험하는 복잡성과 혼란의 대부분은, JAX-WS WSDL 컴파일러가 만드는 클래스들, 즉 JAX-WS/JAXB에 의해 생성되는 클래스들을 WSDL에 명시된 XML 메시지에 매핑하는 방법을 결정하는 어려움에 기인한다. 그러나 이 주제는 앞으로도 많은 난관을 헤쳐나가길 기다려야 하는 별도의 연구 영역(네이티브 XML 타입으로 직접 프로그래밍하는 일을 단순화시켜 줄 수 있는 언어를 만드는 것)이다. 즉, 내가 여기서 말하고자 하는 바는 JAX-WS가 이상적이라는 얘기가 아니라, EJB 3.0이 EJB 2.1에 비해 개선된 것처럼 단순히 JAX-RPC에 비해서 개선된 기술이라는 것이다.

요약해 보자면, WSDL 명세가 나온 이후 엔터프라이즈 자바 커뮤니티는 웹 서비스 기술에 기반한 분산 컴퓨팅을 위한 자바 중심의 플랫폼을 처음부터 만들기 시작했다. 이는 분명 방대한 작업이었고 동시에 명세가 이해하기 어렵다는 사람들의 동요 또한 잠재워야 하는 작업이었다. 이런 관점에서 보면 JWS 표준들은 그리 엉망은 아닌 셈이다. 사실, 자바가 SOA 개발 플랫폼이 되려면 아직 많은 일들이 남아 있다. 이 표준들은 그 과정에서 우리가 웹 서비스 개발의 복잡성과 씨름하는 데 필요한 API들을 제공한다

그러면 왜 우리가 이런 환멸을 느끼는 걸까? 환멸에 빠진 만큼, 거기서 우리가 배워야만 하는 교훈은 무엇일까? 나는 그것이 이 업계에서 우리가 몇 번이고 반복해서 배우게 되는 것—'결코 기술에 쉽게 매혹되지 말라!'는 교훈의 일종이라고 생각한다. 만약 웹 서비스가 분산 컴퓨팅의 만병통치약이 될 것이라고 가정하지 않았다면, 우리는 이런 혼란에 빠지지 않았을 것이다. 대신에, 우리는 좀 더 낙관적인 입장을 취할 수 있었을 것이다.

1.1.2 JWS는 도구 모음이지 애플리케이션이 아니다

웹 서비스가 본질적으로 어렵다는 것을 깨달은 뒤로, 나는 JWS 명세들에 대해 가지고 있던 전제들을 하나 둘씩 다시 검토해 보기로 했다. 나는 더 이상 이 명세들이 대폭 단순화되리라고는 믿지 않았다. 그런 복잡성이, 그 기반에 있는 분산 컴퓨팅이란 문제에서 비롯된 본질적인 것이라고 받아들이게 된 것이다.

JWS를 SOA 스타일 개발의 애플리케이션 프레임워크로서 바라보는 대신, 웹 서비스—SOA 기반의 분산 컴퓨팅 환경의 컴포넌트들—를 소비하고 배포하는 하나의 도구 모음Tool Set으로 인식하기로 했다. 문제는 내가 멍청해서가 아니라 도구에게 너무 많은 것을 기대했기 때문이다. JWS 기술로 SOA 애플리케이션을 생성하는 데는 몇 가지 규칙과 설계에 대한 이해가 필요하다. 이 책의 전반에 걸쳐 JWS 애플리케이션 개발을 좀 더 쉽게 만들 수 있는 좋은 설계들에 대한 예제가 여럿 제시될 것이다.

예를 들어, 4장에서는 '관심사의 분리separation of concerns'를 촉진하는 매개로써 중앙집중식의 스키마 라이브러리의 사용에 대해 설명하고 있다. 그런 라이브러리는 타입 정의 과정(웹 서비스로 SOA 애플리케이션을 생성하는 데 꼭 필요한 부분)을 인터페이스 정의 과정(예: 개별 SOA 컴포넌트들의 WSDL 표현 생성)으로부터 분리하게 된다. 다른 예로, 5장에서는 SOA 시스템에 타입 매핑 시스템을 도입해서, JWS가 생성한 클래스들을 애플리케이션의 다른 부분들로부터 분리시키는 방법을 보여주고 있다. 이러한 기법은 이후에 소개되는 9장의 SOAShopper 구현에서도 사용된다.

좋은 설계와 함께, 보다 쉬운 프로그래밍을 이끌어 낼 수 있는 방법 중의 하나는 애플리케이션 프레임워크를 사용하는 것이다. 이를 테면, 아파치 스트럿츠 프레임워크는 모델 2 또는 모델 뷰 컨트롤러MVC 프레임워크에 기반한 웹 애플리케이션 개발을 도와준다. 프레임워크는 복잡한 도구 모음들의 상위에서 추상화 계층을 제공한다. 추상화 계층은 특정한 방식으로 프로그래밍할 수 있도록 도와주는 역할을 한다. 프레임워크는 프로그래밍 시에 취하는 선택의 폭을 검증된 패턴들의 부분집합으로 제한함으로써 작업을 보다 쉽고, 보다 덜 혼란스럽게 해준다.

애플리케이션 프레임워크는 좋은 설계를 도와주는 역할도 하게 된다. 그 결과로써, 좋은 SOA 프레임워크는 XML 스키마 라이브러리의 사용을 장려하고, WSDL 문서들 간의 스키마 재사용을 촉진하도록 해야만 한다. 좋은 SOA 프레임워크는 반드시 컴파일된 스키마와 WSDL을 다른 애플리케이션 클래스들로부터 분리시켜야만 한다.

애플리케이션 프레임워크는 도구 모음을 이용하지만 사실, 도구 모음 이상의 것을 지향한다. 프레임워크는 도구 모음을 사용하는 데 있어서 특정한 방법을 권장한다. 예를 들어 스트럿츠는 여러 다른 도구모음 중에서 서블릿Servlet과 자바 서버 페이지Java Server Pages, JSP를 사용한다. 이러한 도구 모음 위에서, 스트럿츠는 MVC 프레임워크에 따라 애플리케이션을 만들기 위한 클래스들의 프레임워크(예: `Action, ActionMapping`)를 제공한다.

스트럿츠와 웹 서비스 간의 공통점을 생각해보면서, 나는 JWS가 애플리케이션 프레임워크가 아닌 도구 모음을 제공하는 것이라는 사실을 깨달았다. 우리는 SOA 비즈니스 애플리케이션을 개발하기 위해서 단순히 기반이 되는 도구 모음이 아닌 스트럿츠 같은 애플리케이션 프레임워크를 간절히 원했던 것이다. SOA는 특히 WSDL 중심적이기 때문에, 이상적으로 WSDL 중심 개발을 할 수 있게 해주는 프레임워크를 원하고 있었다.

불행히도, 이 책을 쓰기 전까진 자바 웹 서비스에서 스트럿츠에 비견될 만한 인기 있는 프레임워크가 나타나지 않았다. 나는 그 시발점으로 WSDL 중심 개발을 위한 프레임워크인 SOA-J라는 것을 직접 만들었다. 특별히 호기심 많은 독자들을 위해 귀띔해 주자면 SOA-J에 대한 소개 내용은 11장에 있다.

1.1.3 깨달음

JWS가 애플리케이션이 아니라 도구 모음이라는 것을 이해한 것이야말로 나의 진정한 깨달음이라고 할 수 있다. 이런 사실을 모르고 JWS로 성공적인 개발을 하려 했다면 도구 모음들이 어떻게 동작하는지 알기 위해 그리고 친숙해지기 위해 수많은 시간을 소모했을 것이라는 사실을 그제서야 깨달을 수 있었다. 이 책은 내가 얻은 그런 경험들을 여러분에게 전수하는 책이다. 이 책은 다양한 작업들을 수행하는 방법에 대한 수많은 예제로 가득 차있다(예: REST 엔드포인트를 발행하고, JAXB 바인딩을 Castor와 같은 다른 것들로 대체하거나, WSDL 없이 웹 서비스를 소비하는 등의). 여러분이 많은 코드 예제가 있는 쪽을 좋아한다면 결코 실망하지 않을 것이다.

코드를 본격적으로 파고 들기 전에, 웹 서비스의 개발과 배포를 위해 [3] 모든 플랫폼에 일반적으로 들어가는 여러 가지 컴포넌트들을 살펴보고자 한다. 이를 위해서 몇 가지 공통적인 용어들을 소개할 필요가 있다. 우선 그런 플랫폼들을 웹 서비스 플랫폼이라 칭하기로 하자. 다음 절에서는 웹 서비스 플랫폼들에 대해 알아보기 위해 내가 웹 서비스 플랫폼 아키텍처Web Services Platform Architecture, WSPA라고 부르는, 이 책 전반에 걸쳐 언급되는 플랫폼이 등장한다. WSPA를 우리가 사용할 레퍼런스 아키텍처로 간주해도 좋다. 자바 웹 서비스를 다루는 과정에서 WSPA를 참조해가며 그 장점과 단점을 논할 것이다.

---

3) 웹 서비스 배포를 위한 Java EE 5 외의 다른 플랫폼으로 Axis, Systinet Server 그리고 XFire를 들 수 있다

1.2 웹 서비스 플랫폼 아키텍처

웹 서비스 플랫폼은 특정 프로그래밍 언어를 사용해서 웹 서비스를 호출하고 배포하는 데 쓰이는 도구들의 집합이다. 나는 초점을 자바에 맞추고 있긴 하지만, 이 절에서 다루는 개념들은 여러 언어에 적용될 수 있다.

플랫폼은 서버 측 컴포넌트와 클라이언트 측 컴포넌트들을 가지고 있다. 서버 측 컴포넌트들은 보통 특정한 컨테이너와 함께 패키징된다(예: Java EE 애플리케이션 서버 또는 서블릿 엔진). 클라이언트 측 컴포넌트들은 보통 웹 서비스에 바인딩 되는 자바 인터페이스의 인스턴스에 접근하는 도구로써 패키징된다. 아파치 Axis, XFire, Systinet Server, JWS 또는 그 어떠한 플랫폼이든 간에, 다음의 세 가지 핵심적인 하부시스템을 제공해야만 한다: 호출Invocation, 직렬화[4] 그리고 배포Deployment. 이해를 돕기 위해 이 하부시스템들에 대한 기본적인 사안들을 간추려 살펴보면 JWS가 무엇을 하도록 설계되었는지 이해하고, 그 동작을 논하기 위한 용어들을 파악하는 데 분명 도움이 될 것이다.

1.2.1 호출

서버와 클라이언트 측 모두 각각 다른 호출 방식을 가진다. 서버 측 호출 방식은 다음과 같은 것들을 책임진다.

서버 측 호출

1. 전송계층을 통해 SOAP 메시지를 수신한다(예: HTTP나 JMS 엔드포인트로부터).
2. 메시지의 전처리preprocess 프로세스를 담당할 핸들러를 호출한다(예: 신뢰성을 위한 메시지의 저장이나, SOAP 헤더header의 처리 같은).
3. 타겟 서비스의 메시지를 결정한다—달리 말해, 메시지가 호출하고자 하는 WSDL 오퍼레이션operation이 어떤 것인지를 결정한다.
4. 타겟 WSDL 오퍼레이션을 통해 어떤 자바 클래스/메소드가 호출될지를 결정한다. 이것을 자바 타겟Java Target이라 부르겠다. 자바 타겟을 결정하는 일을 디스패칭dispatching이라고 부른다.
5. SOAP 메시지를 직렬화 하부시스템Serialization subsystem이 처리하게 하고 자바 타겟에 파라미터로 전달할 수 있도록 자바 객체로 역직렬화deserialize한다.
6. 직렬화 하부시스템에 의해 생성된 파라미터를 사용해 자바 타켓을 호출하고, 타겟 메소드에 의해 반환되는 자바 객체를 획득한다.

4) 여기서는 '직렬화(Serialization)와 역직렬화(Deserialization)'를 통칭해서 부르는 줄임말로써 '직렬화'라는 용어를 사용했다.

7. 반환된 객체를 직렬화 하부시스템이 처리하게 하고, 그로 인해 타겟 WSDL 오퍼레이션에 명시된 반환 메시지를 따르는 XML요소로 직렬화하게 한다.
8. 반환된 XML 요소를 타겟 WSDL 오퍼레이션을 다루는 SOAP 메시지 응답SOAP Message Response으로 포장한다.
9. 메시지 전달을 위해 SOAP응답을 전송계층으로 보낸다.

이 프로세스의 각 단계에서, 호출 하부시스템은 예외를 반드시 처리해야 한다. 예외가 발생할 때, 호출 하부시스템은 종종 예외를 SOAP 오류 메시지SOAP fault Message로 패키징해서 클라이언트에게 반환한다. 실제 호출 프로세스는 여기서 설명한 것보다 좀 더 미묘하고 복잡한 것이 사실이다. 하지만, 여기에 개괄적으로 다룬 단계들은, 우리가 자바 웹 서비스 아키텍처를 논하는데 있어서 좋은 출발점이 되어줄 것이다. 이후 장—특별히 JAX-WS를 연구해보는 6장과 7장, 그리고 SOA-J[5] 호출 방식을 설명하는 11장 등에서—에서 이 내용을 좀 더 자세하게 다룰 것이다.

여기서 본 것처럼, 호출 프로세스invocation process는, 결코 단순하지 않다. 이 복잡성의 일부는 SOAP을 지원하는 데서 비롯된 것이다. 대신 3장에서는 REST라고 불리는 좀 더 간단한 대안을 살펴볼 것이다. 그러나 REST를 적용하더라도 호출은 여전히 복잡하다. 웹 서비스의 XML 기술description을 자바 타겟에 매핑시키고, XML 메시지로 타겟을 호출하는 등의 일반적인 문제와는 달리, 호출은 결코 풀기 쉬운 문제가 아니다.

자바 인터페이스를 사용해서, 웹 서비스를 호출하고 싶다면, 클라이언트 측의 호출 프로세스도 서버 측 프로세스와 유사하다. 사실 이 방식이 웹 서비스—이 방식은 여러분이 해결하고자 하는 문제 자체에 많이 의존하게 된다—호출에 항상 가장 적합한 방식은 아니다. 여러분의 클라이언트가 XML로 작업을 한다면, 그냥 XML로부터 SOAP 메시지를 구성하고 웹 서비스에 전달하는 편이 더 쉬울 것이다. 다른 한편에서는, 클라이언트가 자바 객체로 작업한다면 JWS의 방식처럼 클라이언트 측 호출 하부시스템이 그 역할을 수행하게 된다.

클라이언트 측 호출

1. 서비스 엔드포인트 인터페이스service endpoint interface SEI라 불리는 자바 인터페이스를 구현해서 웹 서비스 엔드포인트의 인스턴스를 생성한다. 호출 하부시스템은 SEI 인터페이스들을 생성하기 위한 하나 이상의 팩토리를 가지고 있다. 이 팩토리 인스턴스들은 직접 생성할 수도 있고 JNDI를 사용해서 접근할 수도 있다. 일반적으로 SEI 인터페이스는 자바 프록시와 호출 핸들러Invocation Handler를 사용해서 구현된다. 6장에서 이 매력적인 주제를 다뤄볼 것이다.
2. SEI 인터페이스의 호출을 처리한다.

5) 1.5절에서, SOA-J가 JWS기반의 애플리케이션 프레임워크라고 소개한 바 있다.

3. SEI로 전달되는 파라미터들을 받아서, 직렬화 하부시스템에 전달하게 된다. 직렬화 하부시스템은 파라미터로 전달받은 내용들을, 타겟 서비스의 WSDL에 명시된 XML 스키마를 따르는 XML 요소로 직렬화한다
4. 타겟 서비스의 WSDL에 기반해서 파라미터 요소들을 SOAP 메시지로 포장한다.
5. 메시지를 서비스의 품질 또는 다른 요구사항에 기반해 후-처리post-process(예: 신뢰성을 위해, 메시지를 별도로 저장하거나, SOAP 헤더에 설정한 경우)하는 핸들러를 호출한다.
6. 타겟 웹 서비스로의 전달을 위해 메시지를 전송계층으로 보낸다.
7. 전송 계층으로부터 SOAP 메시지 응답을 수신한다.
8. SOAP 메시지를 직렬화 하부시스템으로 보내 SEI의 반환 타입에 명시된 클래스의 인스턴스인 자바 객체로 역직렬화한다.
9. 역직렬화된 SOAP 응답을 반환해서, SEI 호출을 완료한다.

보다 간단한 설명을 위해, 여기서도 예외 처리 프로세스에 대한 설명은 생략하겠다. 일반적으로 클라이언트 측 호출은 서버 측 호출의 역순으로 이루어진다. 서버 측 호출 서비스 시스템의 최전선에는 WSDL에서 정의된 프록시 SOAP 오퍼레이션이 담긴 자바 메소드가 위치하게 된다. 이 시스템은 자바 메소드를 호출해서 WSDL 오퍼레이션을 실행한다. 클라이언트 측의 호출 하부시스템의 최전선에는 WSDL에서 정의한 SOAP 오퍼레이션이 프록시 자바 인터페이스와 함께 위치하게 된다. 이 시스템은 WSDL 오퍼레이션을 실행해서 자바 메소드 호출을 처리한다. 그림 1.1에 거울에 비친 듯이 대칭적인 이 두 동작이 묘사되어 있다.

여기서 한 가지 흥미로운 점은, 그림 1.1의 중간 부분에 위치하는 WSDL에 의해 명시된 SOAP 요청, 응답 부분이다. 양 끝단의 자바 메소드 호출은 웹 서비스의 관점에서는 임의적인 것이나 다름없기 때문이다. 사실, 서버 측에서는 클라이언트 측과는 완전히 다른 메소드 시그니처를 가지게 된다. 대부분의 경우에 클라이언트와 서버 간의 메소드 시그니처는 서로 다른 편이고, 사용되는 프로그래밍 언어도 다를 수 있다. 하지만 만약 양측이 동일한 자바 클래스 라이브러리로 작업하게 된다면 자바 RMI를 통한 호출을 수행할 수도 있다.

그림의 1-1은 클라이언트와 서버 측의 호출이 마치 거울에 비친 것처럼 유사하다는 사실을 묘사한 것뿐 이라는 점을 명심해두어야 한다. 실제로는, 이 그림의 어느 한 쪽은 자바 메소드 호출을 수행하지 않을 수도 있다. 예를 들어, 웹 서비스이기 때문에 SOAP/HTTP상에서 CICS 트랜잭션(주로 IBM 메인프레임 시스템 등에서 동작하는 트랜잭션 서버—옮긴이주)을 호출하는 자바 클라이언트를 사용할 수 도 있다. 이런 시나리오라면, 클라이언트 측에서만 자바로 된 호출 하부시스템을 가지게 되고 다른 쪽에는 SOAP을 서버 측의 CICS로 변환하게 되는 무엇인가를 사용할 수도 있다.

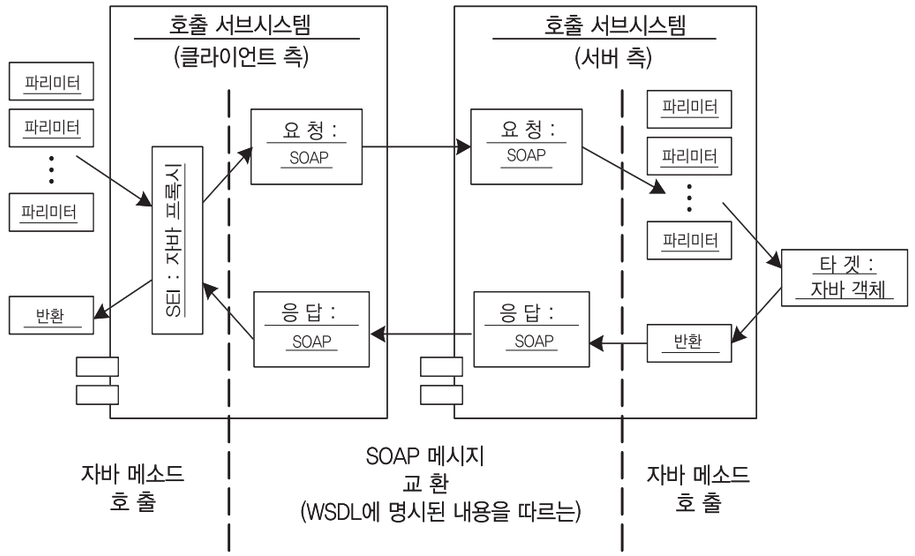


그림 1.1 클라이언트 측 호출 하부시스템은 SEI 프록시상의 메소드 호출을 SOAP 요청/응답으로 변환한다. 서버 측 호출 하부시스템은 SOAP 요청/응답을 자바 타겟의 메소드 호출로 변환한다.

1.2.2 직렬화

직렬화Serialization는 자바 클래스의 인스턴스를 XML 요소Element로 변환하는 과정을 말한다. 역으로, XML 요소를 자바 클래스의 인스턴스로 변환하는 프로세스는 역직렬화Deserialization라고 부른다. 이 책에서는 직렬화와 역직렬화 모두를 일컬어 간단히 '직렬화'라고 부르도록 하겠다.

이론의 여지는 있지만, 직렬화는 모든 자바 웹 서비스 플랫폼에서 가장 중요한 역할을 수행하는 컴포넌트이다. 그림 1.2는 직렬화가 해결해주는 문제들을 보여주고 있다. 이 그림을 설명하기 위해서 직렬화가 WSDL, SOAP과 어떤 연관이 있는지 이 시점에서 좀 더 자세히 다루어보도록 하겠다. [6] 이런 세부적인 사항(아직 1장이긴 하지만!)을 소개하는 과정은 WSPA의 직렬화 하부시스템이 정확히 어떤 일을 하는지 이해하기 위해서도 반드시 필요하다.

웹 서비스 컨테이너 안에는 많은 SOAP 엔드포인트—각각은 웹 서비스들의 그룹에 대응하게 된다—들이 상주해 있다. 엔드포인트는 수행할 수 있는 오퍼레이션들을 정의하는 역할을 하는, WSDL을 하나씩 가지고 있다.

---

6) 이 설명을 이해하기 위해, WSDL과 SOAP을 좀 더 들여다 볼 필요가 있지 않느냐는 의문이 든다면, 그런 걱정은 일찌감치 붙들어 매두어도 좋다. 4장에서 좀 더 자세히 다룰 예정이니까. 지금은 단순히 일반적인 개념들을 이해하는데만 초점을 맞추면 된다.

그림 1.2에서 우측 하단의 설명선으로 된 상자는, 그러한 WSDL 인터페이스의 부분 코드를 보여주고 있다. 이 부분 코드들을 살펴보면서 <types> 요소에 주목해보자. 이 요소는 WSDL 문서의 나머지 부분에서 정의되는 웹 서비스에 의해 사용되는 XML 스키마 타입 정의를 포함하고 있다. 이 부분 코드는 customerPurchase라는 이름의 요소에 대한 정의를 보여주고 있다. 이 요소의 완전한qualified name 이름은 wrapper:customerPurchase다. 요소는 onCustomerPurchase의 메시지 정의의 한 부분으로써 사용된다. 좀 더 살펴보면, CustomerPurchase라는 portType은 입력으로 사용되고 있다. 그리고 이 portType은 입력으로 OnCustomerPurchase 메시지를 사용하는 processCustomerPurchase란 오퍼레이션에 의해 정의되고 있다.

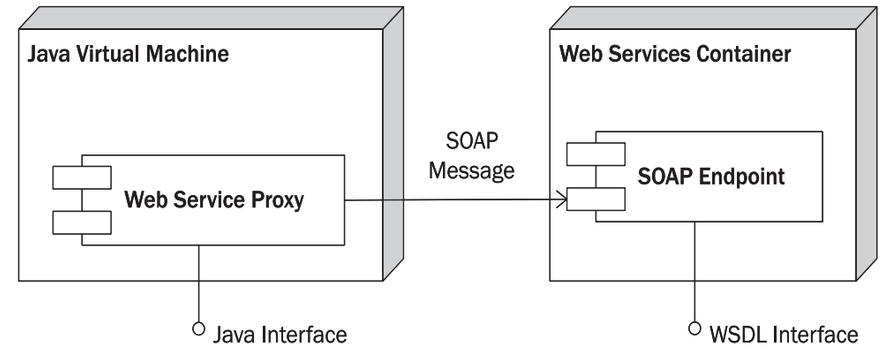
그래서 이 부분 코드는 wrapper:customerPurchase 요소의 한 인스턴스를 포함하는 입력 메시지가 필요한 processCustomerService라는 웹 서비스를 정의한다. 웹 서비스를 호출하면 그 결과로 wrapper:customerPurchase의 인스턴스를 포함하고 있는 SOAP 메시지의 생성을 요구한다. WSDL 부분 코드에 있는 wrapper:customerPurchase에 대한 정의가 imported:customer와 imported:po의 두 요소를 참조하고 있는 것에 주목하자. 이 두 요소에 대한 스키마는 보이지 않지만, 접두어prefix의 이름(imported)으로부터 충분히 WSDL의 어느 부분쯤에서 불려와 졌으리라는 사실을 추측할 수 있다. 그래서 SOAP 메시지의 생성에는 imported:customer와 imported:po의 인스턴스 생성이 필요하다.

자, 이제 그림 1.2의 좌측 하단에 위치한, 설명선으로 된 상자에 있는 자바 부분 코드를 살펴보고, 여기에서 불려진 com.soabook.sales.Customer와 com.soabook.purchasing.PurchaseOrder 클래스에 주목하자. 이 클래스들은 newPurchase 메소드의 파라미터 클래스들로써 사용되었다. 그림 1.2에 보여진 웹 서비스 프록시는 WSDL의 오퍼레이션인 processCustomerPurchase와 자바 인터페이스 메소드인 newPurchase를 바인딩한다. 이 프록시는 호출 하부시스템에 의해 생성된다. 프록시는 SOAP 메시지 송신을 통해 SOAP 엔드포인트에서 배포되는 WSDL 오퍼레이션을 호출하게 된다. 그래서 웹 서비스 프록시의 newPurchase 메소드 구현은 반드시 com.soabook.sales.Customer와 com.soabook.purchasing.PurchaseOrder의 인스턴스를 취하는 어떤 장치를 호출해야만 하고, SOAP 메시지의 바디body 안에 삽입될 수 있는 wrapper:customerPurchase의 인스턴스를 생성해야만 한다.

이러한 장치가 바로, 웹 서비스 플랫폼 아키텍처WSPA의 직렬화 하부시스템이다. 직렬화 하부시스템은 호출 과정에서 다음 단계들에 대한 책임을 진다:

호출 과정에서 직렬화 시스템의 책임

1. 웹 서비스 프록시로부터 파라미터를 전달받는다.




```
package com.soabook;

import com.soabook.sales.Customer;
import
com.soabook.purchasing.PurchaseOrder;

public interface PurchaseTransactions {

  public void newPurchase
    (Customer cust,
     PurchaseOrder po);
  ...
}
```

```
<definitions ...
  targetNamespace="http://soabook.com"
  xmlns:soa="http://soabook.com"
  xmlns:wrapper="http://soabook.com/wrapper" ...>
  <types>
    <schema elementFormDefault="qualified"
     targetNamespace="http://soabook.com/wrapper"
     ...>
      <element name="customerPurchase">
        <complexType>
          <sequence>
              <element ref="imported:customer"/>
              <element ref="imported:po"/>
          </sequence>
        </complexType>
      </element>
      ...
    </schema>
    ...
  </types>
  <message name="onCustomerPurchase">
    <part element="wrapper:customerPurchase"
      name="purchase"/>
  </message>
  ...
  <portType name="CustomerPurchase">
    <operation name="processCustomerPurchase">
      <input message="soa:onCustomerPurchase"
       ... />
      ...
    </operation>
  </portType>
  ...
</definitions>
```

그림 1.2 직렬화는 SOAP을 통해 웹 서비스에 전달하도록 자바 인스턴스를 XML 문서로 변환한다.

2. cust (com.soabook.sales.Customer의 인스턴스) 파라미터를 imported:customer의 인스턴스로 직렬화한다.
3. 파라미터 po(com.soabook.purchasing.PurchaseOrder의 인스턴스)를 imported:po의 인스턴스로 직렬화한다.
4. 이 두 요소를 wrapper:customerPurchase의 인스턴스로 합친다.
5. wrapper:customerPurchase의 인스턴스가 SOAP 메시지에 포함되도록 웹 서비스 프록시에게 전달하고, SOAP 엔드포인트로 보낸다.

이 간단한 예제가 보여주듯이, 직렬화 하부시스템은 자바 인터페이스를 통한 웹 서비스 호출 과정에 있어서 중심역할을 한다. 직렬화 하부시스템은 각각의 자바 클래스들의 인스턴스에서 추출한 파라미터들을 타겟 XML 스키마의 인스턴스로 변환(인터페이스 프록시에 전달)하게 된다—이런 경우에는 wrapper:customerPurchase가 타겟이 된다. 이러한 자바 클래스에서 타겟 스키마 컴포넌트로의 매핑을 가리켜 타입 매핑Type Mapping이라고 부른다. 이러한 변환을 위해서 직렬화 엔진은 어떻게 타입 매핑을 구현할 것인지를 알려주는, 일련의 매핑 전략(그림 1.3에서 묘사한 바 있는)—달리 말해, 어떻게 자바 클래스의 인스턴스를 XML 스키마 컴포넌트들의 인스턴스로 직렬화 할 것인지—을 필요로 한다

매핑 전략은 자바 클래스, 클래스의 타겟 XML 스키마 타입, 그리고 클래스의 인스턴스들을 스키마 타입(또는 기타 등등의) 인스턴스로 변환해 줄 수 있는 직렬변환기 (또는 역직렬변환기)와 연관되어 있다. 직렬화 컨텍스트serialization context는, 특정한 웹 서비스를 배포할 때 사용되는 타입 매핑을 구현하기 위해 직렬화 하부시스템이 사용할 수 있는 일련의 매핑 전략을 말한다.

여러 다른 웹 서비스 플랫폼들이 직렬화 컨텍스트를 만드는 매핑 전략을 제각기 다른 방식으로 제공한다. 대부분 그 과정에는 하나가 아닌 복수의 메소드가 사용되는 편이다. 다음은 그 방법들 중 일부를 나열해본 것이다.

타입 매핑을 구현하는 방법

- 표준 바인딩Standard Binding. 매핑은 자바 클래스들을 XML 스키마로 변환하는 표준 바인딩에 의해 미리 정의된다. 각 자바 클래스는 XML 스키마로서, 고유의 표현을 가지게 된다. JWS는 기본적으로 이 방식을 사용하며, 다른 커스터마이징도 지원한다. 표준 바인딩은 JAXB와 JAX-WS 명세에 기술되어 있다.
- 소스 코드 어노테이션Source Code Annotation. JWS는 표준 바인딩에 기반한 커스터마이징을 지원하는데 이 방식을 사용한다. 타겟 자바 클래스의 소스에 있는 어노테이션들은 어떻게 클래스가 XML 스키마 컴포넌트에 매핑될지, 그리고 웹 서비스의 WSDL이 어떤 형태를 갖게 될지를 명시하기 위해 표준 바인딩을 수정하게 된다.
- 알고리즘을 사용하는 방법Algorithmic. 직렬화 하부시스템이 실행하는 알고리즘에 매핑 방법이 내장된다. JAX-RPC 1.1과 Axis 1.x가 이 방법을 사용한다.
- 규칙 기반Rule Based. 매핑은 직렬화 하부시스템과 독립적으로 생성, 편집될 수 있는 별개의 규칙으로써 명시된다. 이 규칙들은 직렬화 하부시스템에 의해 해석된다. SOA-J(1.5절에서 소개한)는 매핑에 규칙 기반의 방식을 사용한다. Castor [CASTOR] 직렬화 프레임워크도 매핑 파일을 사용해서 이런 접근법을 지원한다.

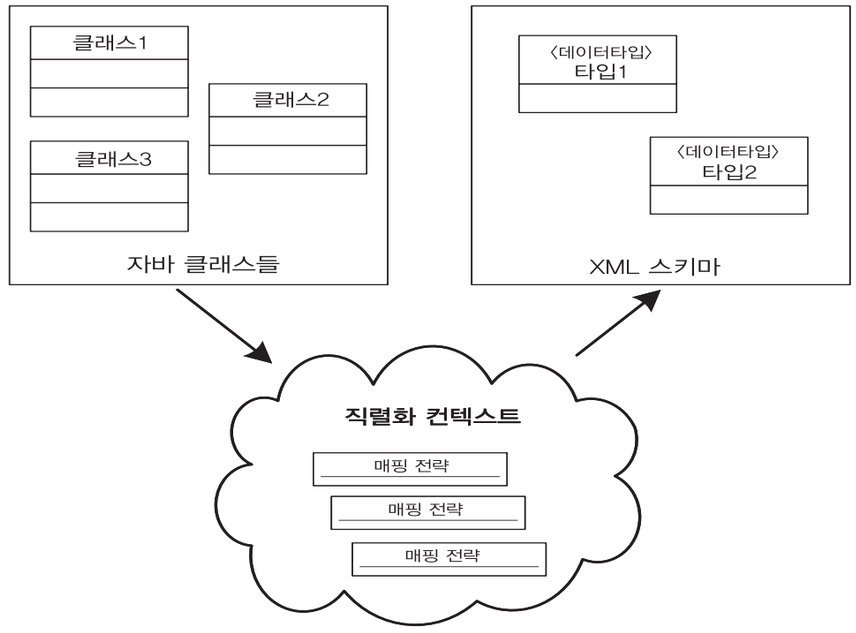


그림 1.3 직렬화 컨텍스트는 직렬화를 수행할 직렬화 하부시스템이 사용할 매핑 전략을 포함하게 된다.

각각의 방식들은 제각기 장단점을 지닌다. JWS는 자바 프로그래머들이 자바 타겟이 어떻게 WSDL 오퍼레이션으로 표현되어야 하는지를 쉽게 명시하기 위해 소스 코드 어노테이션을 도입했다. 나는 규칙 기반 방식을 선호하는데, 최종 사용자가 이미 존재하는 자바 클래스를 기존의 스키마 타입으로 매핑할 수 있어야 하기 때문이다—여러분이 많은 레거시 클래스들과 스키마들을 다루어야 하는 엔터프라이즈 애플리케이션들을 SOA 스타일로 느슨하게 결합시

키기 위해 웹 서비스를 사용하는 경우라면 스키마 타입은 매우 유용한 방법이 될 것이다.

직렬화는 매력적이고 자세히 살펴볼 만한 가치가 충분한 주제다. 여러 가지 다양한 종류의 작업에 대해 제각기 적합한 방식들이 존재한다(예를 들어 레거시 시스템과의 통합과 신규 시스템 개발의 경우를 생각해보자). 바로 이것이야말로 항상 시장에서 수없이 다양한 직렬화 방식간에 경쟁이 계속되고 있는 이유라고 할 수 있을 것이다. 5장에서는 JAXB 2.0의 직렬화를 좀 더 깊숙이 다뤄보겠다.

1.2.3 배포

배포 하부시스템은 자바 타겟을 설정하는 데 필요한 도구를 제공하여, SOAP 기반의 웹 서비스를 실행한다. 좀 더 상위 관점에서 바라보면, 배포 하부시스템은 다음과 같은 것들을 처리할 필요가 있다.

배포 하부시스템의 책임

- 자바 타겟을 배포하기. 이 작업은 호출이 이루어지는 자바 컨테이너에 크게 의존하게 된다. EJB 컨테이너에서는 무상태 세션 빈Stateless Session Bean을 배포하는 것을 의미할 수 있다. 하지만 다른 환경에서는 단순히 각각의 자바 타겟들의 클래스 정의를 호출 하부시스템이 사용하는 클래스 로더에서 사용할 수 있게 하는 것을 의미하기도 한다.
- WSDL 오퍼레이션을 자바 타겟에 매핑하기. 이 작업은 웹 서비스 플랫폼을 설정하는 작업을 포함하는데, 이를 통해 호출 하부시스템이 수신하는 SOAP 메시지를 자바 타겟에 정확히 연관시킬 수 있게 된다. 이 연관관계(또는 바인딩)는 호출 하부시스템이 배포 하부시스템으로부터 얻어올 수 있도록 메타데이터로 저장되며, 반드시 호출되어야 하는 자바 타겟을 결정하는 데 사용된다. WSDL 오퍼레이션을 자바 메소드에 연결시키는 일 외에도, 배포 하부시스템은 호출 시스템이 수신되는 메시지에 대해 SOAP 바인딩을 정확히 해석하도록 도와야 한다(예: RPC 와 포장된wrapped상태의 document style 또는 포장되지 않은 document style).
- 직렬화 컨텍스트를 정의하기. 배포 시스템은 직렬화 컨텍스트 (그림 1.3 참조)를 사용해 직렬화 하부시스템을 설정하게 된다. 이때 시스템은 WSDL과 파라미터에서 얻어진 XML 스키마 타입과 자바 타겟으로부터 얻은 반환 클래스를 바인딩하게 된다.
- WSDL을 발행하기Publishing WSDL. 배포 하부시스템은 자바 타겟을 바인딩될 WSDL 오퍼레이션을 담고 있는 WSDL 문서와 연관시킨다. 이 WSDL 문서는 URL 또는 다른 형식(예: UDDI 레지스트리 내부에서)으로 웹 서비스 클라이언트에게 제공된다.
- SOAP 핸들러SOAP Handler 설정하기. 배포 하부시스템은 자바 타겟의 호출 이전 또는 이후 시점에 QoSQuality of Service를 제공하기 위해, 필요한 SOAP 핸들러를 설정한다. 이 핸들러들은 인증authentication, 신뢰성reliability 그리고 암호화encryption 같은 서비스들을 제공하게 된다. 호출 하부시스템이 핸들러를 호출하긴 하지만 실제로는 배포 하부시스템이 핸들러들을 웹 서비스에 설정하고 연계시킨다.
- 엔드포인트 리스너Endpoint Listener를 설정하기. WSDL 포트에 의해 명시되는 URI에 SOAP 메시지 전송 리스너가 위치하기 때문에 배포 하부시스템은 컨테이너에 별도의 설정을 하게 된다. 몇몇 웹 서비스 플랫폼에서는 WSDL이 엔드포인트가 정의되지 않은 채 제공되고, 이 경우에 엔드포인트는 배포 기술자에 있는 배포 하부시스템에 의해 '채워지게' 된다.

이 설명에서 볼 수 있듯이 배포 하부시스템은 온갖 종류의 매력 없는 작업들을 잔뜩 수행해야만 한다. 다양한 종류의 상황들을 처리하기 위해 (예: QoS 요구사항, 커스텀 자바/XML 바인딩, 설정 가능한 엔드포인트 URL 등) 배포 기술자(배포 하부시스템에 의해 사용되는 XML 파일들)는 GUI 도구 없이는 관리하기 어렵고 복잡할 정도로 빠른 속도로 덩치가 커진다. [7] 그림 1.4는 웹 서비스 플랫폼에서 사용될 수 있는 배포 기술자들의 종류와, 그 기반의 컨테이너와 기술자 간의 관계를 보여주고 있다.

그림 1.4에서 볼 수 있듯이, 웹 서비스 플랫폼은 복수의 컨테이너에 걸쳐 존재할 수 있다. 여기서는, 애플리케이션 서버 컨테이너 (예: Java EE)와 웹 서비스 디렉토리 컨테이너 (예: UDDI)를 보여주고 있다.

애플리케이션 서버 컨테이너는 디렉토리를 포함하고 있을 수도 있다. 화살표는 의존관계를 보여준다. 그림에서 볼 수 있는 것처럼 각각의 객체들은 각 특정 컨테이너의 배포 기술자에 따라 컨테이너에 배포된다. 엔드포인트 리스너, SOAP 핸들러, 자바 타겟은 WSDL/자바 매핑 기술자에서도 사용될 수 있다. 참조reference가 여러 개 있다는 것은 객체도 다수의 역할을 가진다는 것을 의미한다. 예를 들어, 자바 타겟은 컨테이너의 객체와 웹 서비스 플랫폼의 객체 둘 모두로써 배포될 수 있다.

---

7) 돌이켜보면, EJB 2.1에서 작업하는 이들에겐, 기하급수적으로 늘어나는 배포 기술자야말로, 공통적인 불만사항이었던 것 같다. J2SE의 웹 서비스를 통해, 배포 기술자는 더더욱 복잡해져만 갔다. 다행히도 Java EE 5에서, 배포 기술자의 필요성은 급격히 감소했다.

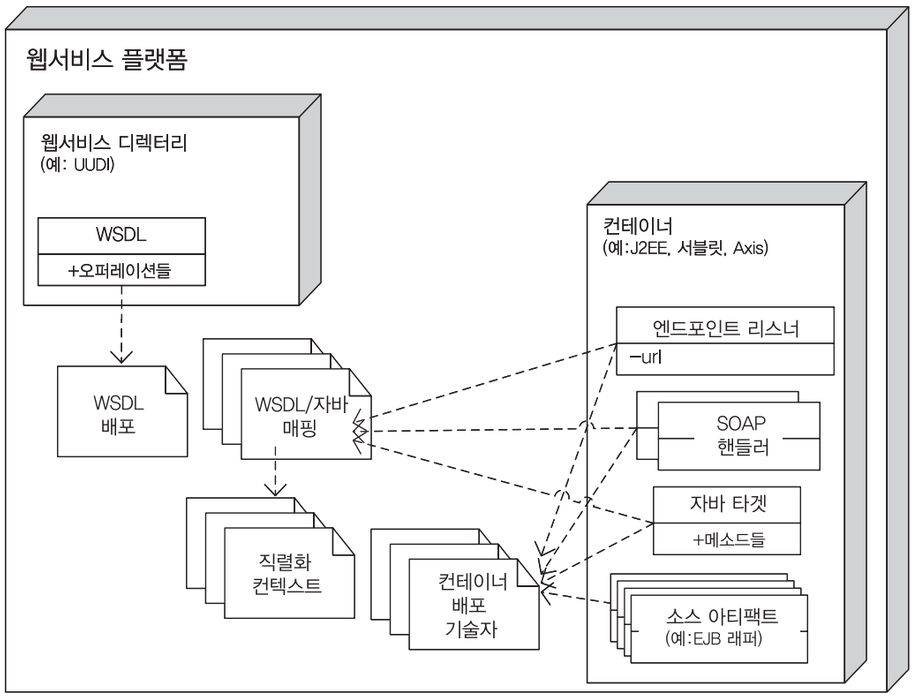


그림 1.4 배포 하부시스템은 매우 많은 배포 기술자를 사용할 수도 있다.

요약하자면, 웹 서비스 플랫폼 아키텍처WSPA는 세 가지 하부시스템을 정의한다: 호출, 직렬화 그리고 배포. 책의 나머지 부분을 통해 아직 다루지 않은 자바 웹 서비스의 내용들을 알아볼 것이다. 그 과정에서 Java EE와 Java SE에 정의된 JWS의 다양한 컴포넌트들이 WSPA 전반에서 수행하는 역할에 대해 논하게 될텐데, 그때 하부시스템들과 그 세부적인 사항들을 다시 살펴보게 될 것이다.

1.3 자바 웹 서비스 : 2장 ~ 8장

앞에서 언급한 대로, 이 책의 주된 목적은, SOA 애플리케이션 개발에 자바 웹 서비스를 사용하는 데 있어서 자세한 기술적 이해를 제공하는 데 있다. 직접 코딩을 해봐야지만 기술적인 실제의 이해를 얻을 수 있기 때문에, 이 책은 그런 기술들을 사용하는 많은 소프트웨어 예제들을 제공한다. 자세한 기술적 이해를 깊게 하기 위한 그 첫 단계는 JWS API의 깊숙한 곳을 탐험해가면서 그 장점과 한계를 살펴보는 것이다. 2장은 JWS API에 대한 고수준의 개요를 제공한다. 3~8장은 이 API들로 어떻게 웹 서비스를 작성하고 배포할지에 대한 자세한 예제들을 제공한다. 이 예제들은 소프트웨어 벤더들이 제공하는 전형적인 "Hello World" 정도의 튜토리얼의 수준을 넘어선다. 대신에, 보다 자세하고 실세계에서 사용할 수 있는 구현들을 제공한다. 게다가 어떻게 하면 이 API들로 프로그래밍할 수 있는지 만을 간단히 보여주는데 그치지 않고 1.2절에서 다룬 웹 서비스 플랫폼 아키텍처와 그 API들을 연관시킨 예제들을 사용한다. 다음은 각 장의 내용에 대한 간단한 요약 설명이다.

2장 : 자바 웹 서비스 개요

주요 자바 웹 서비스 API들의 기능과 강점 그리고 약점에 대해 전반적인 개요들을 다루고 있으며, JAX-WS 2.0 [JSR 224], JAXB 2.0 [JSR 222], WS-Metadata 2.0 [JSR 181], 그리고 Web Services for Java EE 1.2 [JSR 109] 등을 다루고 있다. 그리고 특별히 이 API들이 실제 웹 서비스 애플리케이션 프레임워크에서 어떻게 적용되는지를 살펴본다. 여기서 다루진 않았지만 SOAP with Attachments API for Java (SAAJ) [JSR 67]가 6장과 7장에서 SOAP 처리에 대해 JAX-WS를 논하는 과정 중에 언급된다.

3장 : REST를 사용한 기본적인 SOA

웹 서비스를 가장 단순한 방식으로 바라보는 방식에 대해 다루고, 본격적인 기술적 예제가 나오기 시작한다: Representational State Transfer REST가 기존의 HTTP와 JWS를 사용해 RESTful 서비스를 어떻게 구현하는지를 보여준다. 또한, SOA 통합을 위한 방법으로 REST가 가진 한계들에 대해서도 논의할 것이다. 이러한 한계들을 이해함으로써 다음 장에서 소개하는 SOAP과 WSDL이 왜 필요한지를 알 수 있게 될 것이다.

4장 : SOA에서 WSDL , SOAP의 역할 그리고 자바/XML 매핑

이 장은 SOA에서 왜 WSDL과 SOAP가 필요한지에 대한 논의로 시작된다. 그러고 나서, 실세계의 SOA 통합 시나리오에서, SOAP과 WSDL이 어떻게 사용되는지에 대해 자세히 설명하는 부분으로 넘어간다. 어떻게 WSDL의 구조에 기반해서 SOAP 요청이 디스패치 되는지를 보여주기 위해, SOAP과 WSDL을 1.2절에서 소개한 웹 서비스 플랫폼 아키텍처를 바탕으로 설명한다. 마지막으로, 이 장은 SOAP 요청에 의해 운반된 XML을 웹 서비스를 구현하는 자바 클래스들로 매핑하는 방법에 대해 논한다. 그러한 매핑을 구현하는 도구로써 JAXB 2.0을 소개하고, 그 제약과 해결방법에 대해서도 다룬다.

5장 : JAXB 2.0 데이터 바인딩

JAXB 2.0을 자세히 다루면서, XML을 자바로 매핑하는 다른 방법들과 비교하게 된다. JAXB 2.0의 표준 자바/XML 바인딩, 스키마 컴파일러, 그리고 스키마 컴파일러에 의해 생성되는 어노테이션 등의 많은 기술적인 예제들을 자세히 다룬다. 어떻게 JAXB 런타임이 어노테이션에 기반

해 직렬화와 역직렬화를 수행하는지에 대해서도, 또한 JAXB 직렬화/역직렬화 과정의 동작을 어떻게 여러분 고유의 어노테이션을 통해서 커스터마이징 할 수 있는지도 소개할 것이다. 여기서 JAXB 2.0을 웹 서비스 플랫폼 아키텍처—이 책에서 사용되는 레퍼런스 아키텍처인—의 직렬화 하부시스템으로써 적용하고 있다. 그 다음으로 JAXB의 한계에 대해 자세히 다루는데, 특별히 JAXB 2.0 기반의 직렬화 하부시스템에서 타입 매핑의 추상화가 가진 어려움들을 살펴보고, 이런 어려움들이 어떻게 관심사의 분리를 어렵게 하고 변화 관리에 부정적인 영향을 미치게 되는지에 대해 논한다. 이 장에서는 `XmlAdapter` 클래스와 같은 좀 더 진보적인 JAXB 2.0의 특징들을 어떻게 사용할지, 그리고 어떻게 여러분 고유의 JAXB 2.0 기반의 재귀적인 직렬화 하부시스템을 여러분 고유의 방식으로 구축할지에 대한 자세한 예제를 포함한다. 앞서 논한 제약들에 대한 해결책들도 제시된다. 여기 소개되는 커스텀 직렬화 계획은 SOA-J 직렬화 하부시스템의 프로토타입을 소개하면서 좀 더 상세히 설명할 것이다.

6장 : JAX-WS 2.0 클라이언트 측 개발

이 장에서는, JAX-WS 2.0의 클라이언트 측 API가 자세히 다룬다. 즉, 타겟 서비스의 자바 인터페이스를 제공하는 JAX-WS 2.0 프록시 클래스를 사용해서, 웹 서비스를 호출하는 방법에 대해 설명한다. JAX-WS WSDL 컴파일러(예: 글래스피시의 `wsimport` 명령[8])를 사용해서 컴파일 타임에 WSDL로부터 자바 인터페이스를 생성하는 방법을 다루고 나서, 이 인터페이스를 사용해 런타임에 웹 서비스를 호출할 수 있는 JAX-WS 프록시 클래스 인터페이스를 생성하는 법을 알아볼 것이다. 이런 기초적인 예제들 외에도, 6장에서는 JAX-WS 2.0의 WSDL에서 자바로의WSDL to Java 매핑에 대해 깊이 있게 다룬다. 이 매핑을 이해하는 것은, JAX-WS WSDL 컴파일러에 의해 생성된 클래스들을 사용하는 방법을 이해하는 데 있어서 매우 중요한 대목이다. 생성된 인터페이스 클래스에서 JAX-WS가 사용하게 되는 어노테이션들을 살펴보고 나서, 이 어노테이션들이 어떻게 JAX-WS 2.0 프록시에 의해서, 송수신 메시지를 구성하게 되는지를 보여줄 것이다. 또한 이 장에서는, JAX-WS에서 예외처리와, 자바 `Exception` 클래스 인스턴스, 그리고 SOAP 오류 메시지SOAP Fault Message 간의 매핑 방법에 대해서도 살펴본다. 기본적인 것을 모두 다룬 후엔 3장에서 소개한 REST 모델로 돌아가, JAX-WS를 사용한 XML 메시징(SOAP을 사용하지 않는)에 대한 좀 더 상세한 주제들을 살펴본다. 디폴트 바인딩을 다른 것으로 교체하는 방법과 JAXB 클래스와 Castor를 포함한 다양한 바인딩을 사용하고, 웹 서비스 호출 방법으로 JAX-WS의 사용(`Dispathc` API를 통해)에 대한 논의가 이어진다. JAX-WS의 비동기 호출과 SOAP 메시지 핸들러를 설명하는 데 있어서 입문자도 쉽게 이해할 수 있을 정도의 상세하고 기술적인 예제들이 제공된다.

8) 이 책의 서문에서 밝힌 대로, 모든 예제는 글래스피시 를 사용해서 개발하고 테스트했다. 하지만, 기반이 되는 모든 코드들은 어떤 JAX-WS 구현체에서든지 동작할 수 있다. IBM, JBoss, BEA, 오라클 등의 모든 JAX-WS 도구들이 동일한 WSDL 자바간 매핑을 구현하고 있고, 동일한 클래스들을 생성해낸다.

7장 : JAX-WS 2.0 서버 측 개발

이 장은 JAX-WS 2.0에 대한 논의의 후반부로써, 서버 측 API에 초점을 맞추고 있다. 논의는 JAX-WS의 서버 측 아키텍처에 대한 설명과 이 아키텍처가 1.2 절에서 소개한 웹 서비스 플랫폼 아키텍처WSPA의 참조 설계에 어떻게 대응되는지에 대한 내용으로 시작한다. SOAP 프로토콜 바인딩, 오류 처리, 메시지 핸들러를 포함해서 JAX-WS의 다양한 부분들이 어떻게 조화를 이루는지에 대해 자세한 이해를 돕기 위한 내용들을 다룬다. JAXB 바인딩 인터페이스, WSDL 생성 그리고 디스패치 서비스를 설명하고 난 다음, `@WebService`와 `@WebServiceProvider` 어노테이션 두 가지 모두를 사용해서, 웹 서비스 배포 방법에 대한 일련의 예제들을 소개한다. SOAP 요청과 함께 전달되는 HTTP 요청 헤더로 접근할 수 있게 하는 방법으로써 `WebServiceContext`의 리소스 주입Resource Injection을 다룬다. JAXB 2.0의 또 다른 바인딩 방식으로써, Castor를 사용해서 웹 서비스를 배포하는 방법에 대한 상세한 예제도 다룬다. 그리고 마지막으로 유효성 검증과 오류 처리에 대한 논의와 서버 측 핸들러를 구현하는 방법, Java SE의 웹 서비스 배포 도구인 `javax.xml.ws.Endpoint` 클래스를 사용해서 컨테이너 없이 배포하는 방법 등에 대한 내용들을 다룬다.

8장 : SOA 컴포넌트 패키징하고 배포하기

웹 서비스를 패키징하고 배포하는 방법에 초점을 맞추면서, 이 장을 끝으로 JWS 표준에 대한 상세한 논의를 마감하게 된다. WS-Metadata [JSR 181] 어노테이션과, 이를 사용해 EJB와 서블릿 엔드포인트을 배포하는 방법에 대한 예제를 다룬다. 배포 기술자를 전혀 사용하지 않고서도 웹 서비스를 배포할 수 있는 방법(그렇다, Java EE 5에서는 가능하다!)과 디폴트 배포 방식과 어노테이션 기반의 배포 방식을 재정의하는 방법을 소개하고, 언제 어느 부분에서 배포 기술자를 사용하는 것이 적합한지를 배울 것이다.

이 장은 서블릿 엔드포인트의 배포에 필요한 WAR 구조에 대한 설명과, EJB 엔드포인트에 사용되는 EJB JAR/EAR 구조에 대한 설명도 포함하고 있다. 또 그 밖에도, Java EE 5 컨테이너 구현체가 배포 처리를 어떻게 수행하는지에 대한 상세한 개요도 다룬다(예: 컨테이너가 어노테이션이 달린 패키지 컴포넌트에서 배포된 웹 서비스로 이동하는 방법).

다른 JWS API에 대한 설명처럼, 이 논의도 Java EE 5의 웹 서비스 배포 하부시스템에 대한 장/단점을 비평하기 위해, 1.2절의 웹 서비스 플랫폼 아키텍처의 레퍼런스 설계를 적용하고 있다. 패키징과 배포에는 수많은 방법이 존재하기 때문에, 이 장은 이런 방법들을 적용한 각각의 다양한 형태들에 대한 10 종류의 패키징 예제를 포함하고 있다. 마지막으로, 심화 주제로써 Java EE 5에서 제공되는 OASIS XML Catalog [XML Catalog 1.1] 지원에 대해 다루게 된다.

1.4 SOAShopper 사례 연구 : 9장 , 10장

9장과 10장은 2장부터 8장까지 다뤘던 기술과 기법들을 통합해서 살펴보고, JWS를 사용해서 SOA 통합 애플리케이션을 만드는 방법을 보여주고 있다. SOAShopper 애플리케이션은 이베이, 아마존, 그리고 야후! 쇼핑을 통합하는 온라인 쇼핑 시스템이다. 이 시스템은 각 쇼핑 사이트들의 클라이언트로써, 웹 서비스 소비자consumer 역할을 하게 된다 또한, SOAShopper는 웹 서비스 공급자provider이기도 한데, 이는 이 세 쇼핑 사이트들을 아우르는 검색 서비스를 제공하는 REST와 SOAP 엔드포인트를 배포하기 때문이다.

9장 : SOAShopper ,이베이, 아마존, 야후 쇼핑 통합하기

SOAShopper는 데모 애플리케이션이긴 하지만 여기서 보여주는 기법들은 무척 강력한 것들이다. 이 장에서는 실제 SOA 통합 시스템을 생성하기 위해 이전 장에서 다룬 모든 도구들을 사용하게 될 것이다. 이 장은 RESTful 서비스들을 배포하고 소비하는 코드, WSDL/SOAP 서비스의 배포와 개발, JAXB로 타입 매핑 구현하기, WSDL 중심적인 서비스 통합, 그리고 Ajax 클라이언트에 대한 지원 등에 대한 논의와 예제들을 포함하고 있다.

10장 : Ajax와 자바 웹 서비스

10장은 9장에서 개발했던 SOAShopper 애플리케이션을 사용해서 RESTful 서비스들을 소비하는 Ajax 프론트 엔드를 만들어본다. 이 장에서는 JWS가 Ajax 클라이언트들을 지원하는 데 있어 실제로 어떻게 사용되는지에 대해 초점을 맞추게 된다.

1.5 SOA-J와 WSDL 중심 개발 : 11장

자바 웹 서비스JWS만을 배우고 싶은 독자라면 11장을 건너뛰어도 좋다. 대신, 웹 서비스 플랫폼 아키텍처WSPA가 어떻게 WSDL 중심적인 방식으로 구현되었는지, JWS에 의해 제공되는 도구들을 어떻게 사용했는지가 궁금하다면, 11장을 반드시 읽어야 한다. 이 최종 장에서는, SOA-J의 세세한 부분들을 짚어가며 설명하고 있다.

WSDL 중심적이라는 용어는 WSDL을 통해 웹 서비스를 생성하고, 그 WSDL 문서에 구현하는 자바 요소에 대한 참조를 어노테이션으로 다는 방식을 말한다. 이러한 WSDL 중심적인 접근방식은, 기업의 표준 또는 e-비즈니스 프레임워크(예: 표준 스키마와 메시지 기술이 존재하는 경우)에 통합된 웹 서비스를 생성하는 경우에 적합하다고 볼 수 있다. [9]

9) 4장에서 그런 상황들에 대해 설명하고 있다.

SOA-J는 프로토타입 애플리케이션 프레임워크다. 나는 JWS로 WSDL 중심적인 SOA 개발을 할 수 있다는 가능성을 살펴보고자, 개념을 증명하기 위한 목적으로 이 프레임워크를 개발했다. SOA-J의 소스코드(오픈소스)는 이 책과 함께 제공되는 코드 예제로 제공된다. [10] 최신 버전의 SOA-J 또한 http://soa-j.org에서 찾아볼 수 있다.

11장 : SOA-J와 함께하는 WSDL중심 자바 웹 서비스

이 장에서는 JWS 기반의 애플리케이션 프레임워크인 SOA-J가 어떻게 설계되었는지를 살펴보게 된다. SOA-J가 WSDL로부터 SOA 컴포넌트를 조합하는 단순한 방식을 어떻게 제공하는지를 설명할 것이다. 이 장에는 제법 많은 UML 다이어그램이 나온다. SOA 아키텍처를 좀 더 자세히 다루고 싶었기 때문이다. 이 논의는 웹 서비스 엔진(SOAP 서버라고 부르기도 하는)이 어떻게 동작하는지를 궁금해 왔던 모든 사람들에게 유용할 것이다. 여기에서 사용된 원칙들은 WSPA [11]에 요약되어 있으며, Axis와 XFire와 같은 제품들에도 녹아 들어 있다.

이 장에 세부 구현을 포함한 이유는, 예전에 내가 아파치 Axis를 접했을 때의 경험에서 비롯된 것이다. 처음 Axis 소스 코드를 접했을 때, 나는 완전히 길을 잃어버린 기분이었다. 다행스럽게도, SOA-J에 대한 설명을 읽고 나면, 여러분이 다른 웹 서비스 엔진의 소스 코드를 보게 되더라도, 길을 완전히 헤매진 않을 수 있을 것이다.

10) 책의 예제들을 다운로드하고 설치하는 방법에 대해서는 부록 B를 참조하라.

11) 1.2 절을 참조하라