Hanbit eBook

Realtime 34

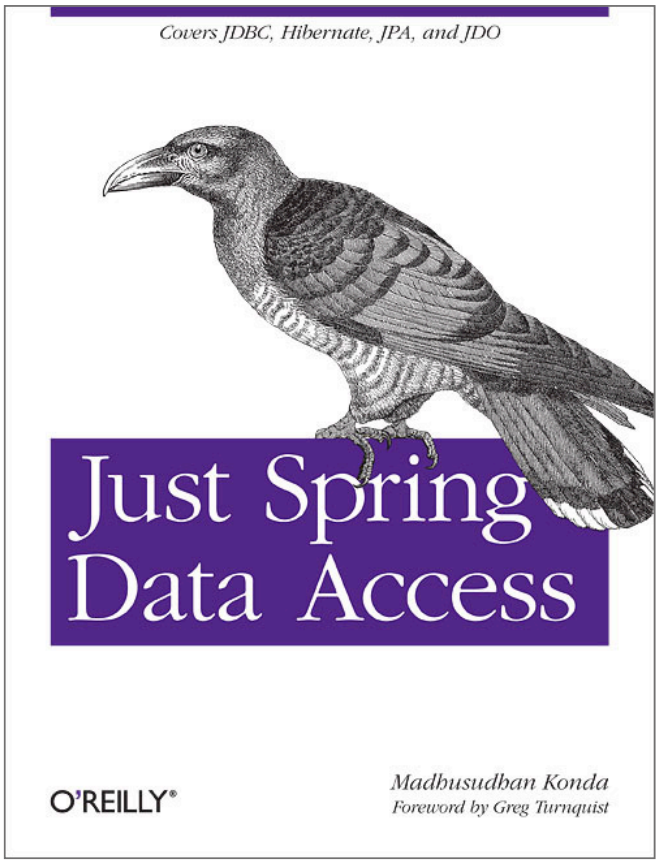
# 스프링 데이터 핵심 노트

## 자바 데이터베이스 API를 위한

## Just Spring Data Access

마드후수단 콘다 지음 / 송지연 옮김

O'REILLY® 한빛미디어 Hanbit Media, Inc.




이 도서는 O'REILLY의
Just Spring Data Access의
번역서입니다.

자바 데이터베이스 API를 위한

# 스프링 데이터
# 핵심 노트

자바 데이터베이스 API를 위한 **스프링 데이터 핵심 노트**

**초판발행** 2013년 7월 22일

**지은이** 마드후수단 콘다 / **옮긴이** 송지연 / **펴낸이** 김태헌
**펴낸곳** 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
**전화** 02-325-5544 / **팩스** 02-336-7124
**등록** 1999년 6월 24일 제10-1779호
**ISBN** 978-89-6848-637-1 15000 / **정가** 9,900원

**책임편집** 배용석 / **기획** 이중민 / **편집** 김연숙
**디자인** 표지 여동일, 내지 스튜디오 [밈], 조판 박진희
**마케팅** 박상용, 박주훈, 정민하

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
**한빛미디어 홈페이지** www.hanb.co.kr / **이메일** ask@hanb.co.kr

Published by HANBIT Media, Inc. Printed in Korea

지은이_ **마드후수단 콘다**

마드후수단 콘다Madhusudhan Konda는 자바 전문 컨설턴트로, 주로 투자 은행과 금융 기관에서 근무했다. 엔터프라이즈와 코어core 자바 분야에서 약 12년간 일했으며 분산, 멀티스레드, N-티어 스케일러블N-Tier Scalable과 같은 확장 아키텍처가 관심 분야다. 또한 금융 관련 고빈도high-frequency와 저지연low-latency 애플리케이션 아키텍처를 설계한 경험도 있다. 집필에 남다른 애착을 갖고 있으며 멘토링에도 관심이 많다.

옮긴이_ **송지연**

지엔텔, 노키아 지멘스 네트웍스에서 근무한 경험이 있는 WCDMA, LTE 분야의 통신 기술 엔지니어. 취미로 팀을 이루어 아이폰 애플리케이션 개발에 한동안 푹 빠져 있기도 했다. 현재는 주전공인 SW 개발 분야로 돌아와 오라클 자바 개발 팀에서 근무 중이다.

# 추천사

"페이스북이 30페타바이트의 하둡 클러스터를 새로운 데이터 센터로 이주시켰다." 와 같은 기사를 읽으면 현재 우리가 직면한 가장 큰 문제가 바로 빅데이터의 관리라는 것을 직감할 수 있다. 또한 데이터 중심의 애플리케이션, 복잡한 데이터 구조를 가진 모바일 프런트 엔드 개발, 하루 동안 수십억 트랜잭션을 다루면서 수백만 클라이언트를 데이터 세트에 연결할 수 있게 해야 하는 경우 등의 사례는 애플리케이션 개발에서 데이터 관리와 유지를 간단하고 쉽게 하는 것이 얼마나 어려운 일인지를 알 수 있게 해준다.

감사하게도 스프링 데이터 같은 도구와 유틸리티는 각 팀의 능력과 요구사항에 맞는 표준으로, 모든 데이터 세트에 쉽게 접속할 수 있도록 도와준다. 특히 자바가 유연하면서도 일관된 JDBC 표준을 제공하여 전통적인 SQL 패러다임에서 발생했던 지루하고 반복적인 작업을 과감히 없앤 일은 스프링 프레임워크의 큰 장점이 되었다. 즉, 개발자는 비즈니스 로직, 스케일링 요구사항, 모바일 플랫폼 지원과 같은 여러 가지 요구사항 구현에만 집중할 수 있게 하고, 스프링 프레임워크는 여러 가지 데이터 관리 기술과 상호작용하며 애플리케이션과 데이터의 연결을 관리하게 한다는 뜻이다. 이는 마치 C에서 수동으로 했던 메모리 관리를 (매일 개발자가 다루어야 했던 버그들을 모두 없애버린) 자바의 정교한 가비지 컬렉션으로 변경한 것과 비슷할 정도로 획기적이다. 또한 빅데이터에 접근하는 데 필요했던 전체 코드의 양도 줄어들었기 때문에 프로젝트를 시작할 때 발생할 수 있는 잠재적인 버그의 양 또한 줄일 수 있게 되었다.

이처럼 데이터 관리가 중요시되는 시점에서 『스프링 데이터 핵심 노트』와 같은 얇고 정리된 책은 예전처럼 두껍게 만든 책을 읽지 않고도 간단하고 명료하게 스프링 데이터에 접근할 수 있도록 도와주므로 추천하는 바다. 스프링 데이터를 사용하는

팀에 갓 들어온 신입이 이 책을 주말 동안 읽어본다면 빠르게 스프링 데이터에 대해 숙지한 후 월요일에 당장 업무에 임할 수 있을 정도다. 또한 이 책은 새로운 시스템에 사용할 데이터 처리 표준을 정하려는 설계자가 매우 빠르게 읽을 수 있는 책이며, 여러 사람과 간단히 의견을 교환하는 것보다 더 구체적인 사항을 평가할 수 있도록 도와줄 것이다. 그리고 숙련된 개발자라면 데이터 관리와 다른 자바 표준에서 제공하는 옵션들을 좀 더 숙지하고 다시 한 번 검토할 수 있도록 도와주는 참고서가 될 것이다. 끝으로 누구도 모든 것을 다 알 수는 없기 때문에 현재 당면한 문제점을 해결하고 해당 기술을 익히려면 한 분야의 전문서적을 자주 찾아보기 바란다.

**그렉 턴퀴스트** Greg Turnquist

스프링 파이썬 1.1의 저자이자

VMWare SpringSource팀의 소프트웨어 개발자

# 저자 서문

여기 두 개의 다른 세계가 있습니다. 하나는 객체 이외에는 아무것도 알 수 없는 세계고, 다른 하나는 전통적인 행-열 방식으로 데이터가 존재하는 세계입니다. 이렇게 서로 다른 두 세계를 연결하는 일은 항상 다루기 어려운 작업이고 문제가 발생할 소지도 많습니다. 하지만 두 세계는 반드시 함께 동작해야만 하며 다른 대체 방법이란 없습니다.

물론 이를 어느 정도 해결하기 위해 JDBC를 이용할 수 있지만 자바 객체를 관계형 데이터베이스에 저장할 때의 복잡성과 어려움은 여전히 문제로 남습니다. 하지만 다행히 객체 관계 매핑Object Relational Mapping, ORM 프레임워크가 이러한 어려움을 해결하여 개발자의 불편함을 줄였고, 그중 하이버네이트는 가장 인기 있는 오픈 소스 프레임워크입니다. 그리고 스프링 프레임워크는 여기서 한 발 더 나아가 이를 사용하는 방법을 더욱 간단하게 해주었습니다.

이 책은 개발자가 스프링 프레임워크에 좀 더 가까이 다가갈 수 있도록 세심한 노력을 기울였습니다. 그래서 간단하면서도 단순한 용어와 이해하기 쉬운 예제를 통해 자바로 웹 애플리케이션을 개발할 때 필요한 데이터 액세스 방법을 다룹니다. 또한 JDBC, 하이버네이트, JPA, JDO에 대해서도 살펴보고 스프링 프레임워크가 이 기술을 어떻게 활용하는지도 설명합니다.

간단하고 직관적이며 명료한 설명을 담은 예제 중심의 매력적인 책을 집필하는 것이 제 목적이었습니다. 그렇기에 지금 이 책을 읽는 여러분이라면 반드시 하루나 이틀 만에 이 책을 다 읽을 수 있을 것입니다! 그리고 이 책을 통해 충분한 지식과 정보를 얻게 될 것임을 진심으로 믿습니다.

이 책을 읽으려면 먼저 자바와 스프링 프레임워크의 기본을 이해한 상태여야 합니다. 이 책을 읽으며 즐길 수 있길 바라며 내용에 대해 만족스럽지 않은 부분이 있다면 필자에게 알려주시기 바랍니다. 런던에 거주하는 분이라면 직접 만나서 이야기하는 것도 좋습니다(커피 한 잔 사주시는 것도 환영합니다). 또한 이메일(madhusudhan@madhusudhan.com)이나 트위터(@mkonda007)로 질문을 주시거나 만족스럽지 않은 부분에 대해 알려주시는 것도 환영합니다.

끝으로 편집자 마이크 루키디스Mike Loukides, 매건 블랜새트Meghan Blanchette, 모든 오라일리 편집부, 특히 이 책을 구성하는 데 도움을 준 아이리스 페브레스Iris Febres에게 진심으로 감사드립니다. 또한 프로젝트를 진행하는 동안 필자를 지도해준 그렉 턴퀴스트Greg Turnquist에게도 깊은 감사의 뜻을 표합니다.

이 책을 집필하는 동안 묵묵히 인내해주고 협조해준 사랑하는 아내 제네티Jeannette와 디즈니랜드 여행에 대한 답례로 아빠와 보낼 시간을 책 쓰는 시간으로 할애할 수 있게 허락해준 멋진 5살 아들 조슈아Joshua에게도 진심으로 고맙다는 말을 전합니다. 마지막으로 많은 협조와 사랑을 보내준 인도의 가족에게도 매우 감사하다는 말을 전하고 싶습니다.

사랑하는 아버지를 기리며! 우리는 모두 아버지를 그리워합니다.

마드후수단 콘다Madhusudhan Konda

www.madhusudhan.com

On Twitter @mkonda007

# 역자 서문

스프링은 세상에 선을 보인 이후 자바에 기반을 둔 웹 개발 프레임워크로 큰 인기를 얻고 있습니다. 하지만 새로운 프레임워크를 공부하기란 쉽지 않은 일입니다. 이 책은 스프링의 개념을 상당히 간단하면서도 쉽게 설명하고 있으며, 분량 또한 많지 않습니다. 스프링 프레임워크에 대한 근본적이면서도 반드시 필요한 부분을 명확하게 짚어주고 있으므로 잠시 시간을 내어 읽어본다면 많은 도움이 될 것이라 확신합니다.

자바 엔터프라이즈 개발을 고려하거나 개발이 필요하다면 스프링 프레임워크는 이제 반드시 익혀야 할 기술이라고 생각합니다. 하지만 막연히 새로운 기술이라는 이유로 거부감이 든다면 이 책을 통해 일단 스프링 프레임워크의 기본을 습득해보는 일도 좋을 것이라 생각합니다. 이 책은 이제 개발에 발을 막 내디딘 초보 개발자들도 이해하기 쉬울 정도로 많은 예제를 통해 친절하게 설명했을 뿐만 아니라 스프링 프레임워크의 전체 내용보다는 기본 개념을 설명하므로 모든 독자가 큰 어려움 없이 읽을 수 있을 것으로 생각합니다.

스프링 프레임워크는 그 자체로도 훌륭한 기술이지만 다른 표준 기술, 예를 들어 하이버네이트나 JDO, JPA 등도 지원한다는 점 때문에 앞으로의 확장 가능성도 더 엿볼 수 있습니다. 따라서 스프링 프레임워크에 대해 좀 더 자세한 내용을 알고 싶다면 이미 출간되어 있는 『스프링 핵심 노트』 (한빛미디어, 2013)와 『스프링 인티그레이션 핵심 노트』 (한빛미디어, 2013)에도 관심을 가져주시기를 바랍니다.

개인적으로도 이 책을 번역하면서 스프링, 하이버네이트 등에 대한 자료와 API를 살펴볼 기회를 갖게 되어 많은 도움이 되었습니다. 이 때문에 매너리즘에 빠지려는

순간에 새로운 자극제를 얻을 수 있었으며 예전에 프로그래밍을 처음 공부하면서 가졌던 열정을 다시금 불태울 수 있는 계기가 되기도 했습니다.

책이 완성되어 세상에 나올 때까지 많은 분의 도움을 받았습니다. 일단 이 책을 번역할 기회를 만들어준 오빠에게 감사하며, 함께 작업하면서 많은 도움을 준 한빛미디어 관계자분께도 감사드립니다. 또한 역자의 부족한 지식 때문에 여러 번 귀찮게 했던 삼성 SDS 정혜미 선임에게도 깊은 감사를 드리며, 항상 제 곁에서 버팀목이 되어주신 부모님과 친구들에게도 감사를 드립니다.

번역을 마치며

역자 **송지연**

# 대상 독자 및 시리즈 도서 구성

초급 **초중급** 중급 중고급 고급

이 책은 자바 웹 애플리케이션 개발 프레임워크의 대세로 자리를 잡아 가는 스프링의 핵심 개념을 소개하는 책입니다. 자바와 XML의 기본을 알고 있으며 스프링 프레임워크에 관심이 있는 분이라면 누구나 읽을 수 있습니다. 또한 세 권의 시리즈로 기획되었으며 다음 내용을 다룹니다.

**『스프링 핵심 노트』**

스프링 프레임워크의 기본과 핵심 개념을 다룹니다. 스프링 프레임워크를 간단히 소개하고 스프링의 근간이 되는 빈과 컨테이너를 살펴봅니다. 마지막에는 스프링 JMS와 JDBC, 하이버네이트 등 데이터 처리와 관련한 내용을 소개합니다.

**『스프링 인티그레이션 핵심 노트』**

『스프링 핵심 노트』 혹은 스프링 프레임워크 입문서를 읽어본 독자들이 기업용 애플리케이션 통합Enterprise Application Integraion 과정을 살펴볼 수 있도록 엮어진 책입니다. 또한 개발 과정에서 비즈니스 로직을 수립하는 방법도 배울 수 있습니다.

**『스프링 데이터 핵심 노트』**

JDBC, 하이버네이트, JPA, JDO 등 스프링 프레임워크와 연결해서 사용하는 데이터 처리 방법을 소개합니다. 스프링 프레임워크와 데이터베이스의 관계를 심도 깊게 이해함으로써 자바 기반의 웹 애플리케이션을 자유자재로 만들 수 있는 기반을 다질 수 있습니다.

# 예제 파일

- 스프링 프레임워크 공식 사이트

  : http://www.springsource.org/spring-framework

- 스프링 데이터 공식 사이트

  : http://www.springsource.org/spring-data

- 예제 파일 다운로드 사이트

  : http://examples.oreilly.com/0636920025405/

# 한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook 입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. **eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.**

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내시는 선배, 전문가, 고수분에게는 보다 쉽게 집필하실 기회가 되리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. **무료로 업데이트되는, 전자책 전용 서비스입니다.**

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정한 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나, 저자(역자)와 독자가 소통하면서 보완되고 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

### 3. 독자의 편의를 위하여, DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT기기에서 자유롭게 활용하실 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해, 독자 여러분이 언제 어디서 어떤 기기를 사용하시더라도 편리하게 전자책을 보실 수 있도록 하기 위함입니다.

### 4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

# 차례

# 1 | 기초 개념

데이터를 퍼시스턴스Persistence [01]하는 것은 중간에 문제가 발생할 수 있는 요소가 많으므로 매우 어려운 작업이다. 이런 이유로 자바 애플리케이션에서는 다루기 어려운 작업이었던 데이터베이스 액세스를 대신 처리해주는 JDBC의 등장은 개발자들에게 희소식일 수밖에 없었다. 하지만 JDBC는 상용 구문boilerplate code을 여러 번 써야만 한다든가, SQLExcetion 스택트레이스stacktrace로부터 예외 관련 내용을 알아내야 한다든가, 리소스 관리에 문제점이 있다든가 하는 등의 몇몇 단점도 가지고 있다.

스프링 프레임워크는 간단하고 명확한 구조를 제공함으로써 데이터 액세스를 단순화시킬 수 있도록 한 발 더 진화했다. 1장에서는 스프링 프레임워크가 JDBC로부터 가져온 기능과 어떻게 JDBC 프로그래밍 모델을 단순화시켰는지에 대해 알아본다. 먼저 간단히 설명하자면 스프링에서는 의존성 주입이나 템플릿, 그 외 패턴들과 같은 간단하지만 강력한 메커니즘을 사용하여 이를 가능하게 했다.

## 1.1 기존 JDBC의 사용

JDBC가 등장하면서 자바 애플리케이션에서 데이터에 액세스하는 일은 비교적 쉬워진 편이다. 데이터베이스 벤더에 종속되어야만 했던 상황에서 벗어났을 뿐만 아니라 여러 가지 데이터베이스에 액세스하기 위한 표준 API까지 가질 수 있게 되었기 때문이다. 하지만 JDBC를 사용하는 절차는 항상 똑같다. 애플리케이션과 데이터베이스를 연결한 후 Statement 객체를 생성하며 쿼리문을 실행하고 이를 ResultSet 객체로 실행하고 리소스를 해제하는 것이다.

---

01 역자주_객체의 속성을 어딘가에 저장하고 나중에 객체를 다시 생성했을 때 객체가 파괴되기 전의 상태로 되돌아갈 수 있는 속성을 말한다. 객체의 측면에서 보면 영원히 존재하는 것이라 볼 수 있다.

다음은 JDBC를 사용해서 TRADES라는 테이블의 데이터를 선택하는 예제다.

```
public class JdbcPlainTest {
  private String DB_URL="jdbc:mysql://localhost:3306/JSDATA";
  private final String USER_NAME = "XXXX";
  private final String PASSWORD = "XXXX";

  private Connection createConnection() {
    Connection conn = null;
    try {
      Class.forName("com.mysql.jdbc.Driver");
      conn = DriverManager.getConnection(DB_URL, USER_NAME, PASSWORD);
    } catch (ClassNotFoundException e) {
      e.printStackTrace();
    } catch (SQLException e) {
      e.printStackTrace();
    }
    return conn;
  }

  private void query() {
    ResultSet rs = null;
    Statement stmt = null;
    Connection conn = createConnection();

    try {
      stmt = conn.createStatement();
      rs = stmt.executeQuery("SELECT * FROM TRADES");

      while (rs.next()) {
        System.out.println(rs.getString(1));
      } catch (SQLException e) {
```

```
            e.printStackTrace();
          } finally {
            try {
              rs.close();
              stmt.close();
              conn.close();
            } catch (SQLException ex) {
              e.printStackTrace();
            }
          }
        }
      }

      public static void main(String args[]) {
        JdbcPlainTest t = new JdbcPlainTest();
        t.query();
      }
    }
```

---

이 간단한 작업을 위해서 이렇게나 많은 코드가 필요하다니! 위 예제로부터 예외 처리를 위한 코드를 몇 가지를 확인해보자.

- 리소스 관리(데이터베이스에 연결하는 것과 SQL 구문을 생성하고 종료시키는 것)는 반복되는 프로세스다.
- SQLException 클래스는 프로세스를 생성하고 종료할 때 모두 실행되어야 한다.
- 실제 비즈니스 로직은 몇 줄의 코드로 작성할 수 있는 것이 아니므로 이를 구현하면 많은 수의 JDBC API 구문과 호출로 인해 전체 코드가 지저분해질 것이다.

이러한 문제를 해결하기 위한 콜백 메서드와 핸들러들로 직접 프레임워크를 만들 수도 있다. 하지만 이것이 가능하다고 할지라도 직접 프레임워크를 만드는 것은 몇 가지 문제를 일으킬 수 있다. 문제의 예를 들면 유지보수, 새로운 요구사항에 대응하기 위한 확장, 광범위한 테스트 등이다. 그러므로 위에서 언급한 문제점들을 해결할 수 있는 프레임워크가 이미 존재한다면 굳이 새로 만들 필요는 없을 것이다.

스프링 데이터 액세스 프레임워크는 이러한 문제를 해결하기 위해 등장한 프레임워크로, 의존성 주입 원칙과 여러 가지 기능을 가진 매우 좋은 프레임워크라고 이야기하고 싶다.

## 1.2 스프링 데이터 액세스 프레임워크

스프링 데이터 액세스 프레임워크는 개발자가 해야 하는 작업을 단순화시켜주었다. 즉, 액세스 메커니즘으로부터 애플리케이션을 분리하여 데이터베이스 혹은 데이터베이스에 접근할 수 있는 프레임워크를 만들었다는 뜻이다. 항상 그랬듯이 스프링 프레임워크는 의존성 주입 패턴을 주로 사용하므로 애플리케이션을 분리하는 것이 실제로 가능해진 것이다.

그리고 스프링 프레임워크에서 제공하는 API를 사용하는 컴포넌트를 테스트해보기도 매우 쉽다. 게다가 API를 사용할 때 고려해야 할 예외 상황도 전혀 없다! 또한 데이터 액세스 로직은 템플릿 패턴과 지원 클래스를 중점적으로 다룬다. 이는 모든 상용 구문을 숨기고 개발자로 하여금 오직 비즈니스 로직에만 집중할 수 있도록 도와준다는 뜻이다.

## 1.3 템플릿

이전 예제에서 실제 기능과 관련 없는 수많은 코드를 살펴보았다. 이렇게 중요하지 않은 코드는 모두 별도 클래스로 묶어 실제 비즈니스 코드와 분리하는 편이 좋다. 스프링 프레임워크의 JdbcTemplate 클래스가 바로 이 역할을 한다.

JdbcTemplate 클래스는 모든 액세스 로직을 포함하므로 개발자는 애플리케이션의 핵심 부분 개발에만 집중할 수 있게 된다. 만약 JdbcTemplate 클래스가 어떻게 동작하는지 이해할 수 있다면 스프링 프레임워크의 데이터 액세스가 어떻게 이루어지는지에 대한 대부분을 파악하고 있다고 말할 수 있다.

또한 표준으로 사용하는 JdbcTemplate 클래스 외에도 템플릿 클래스의 변형인 SimpleJdbcTemplate 클래스와 NamedParameterJdbcTemplate 클래스가 있다. 이들은 JdbcTemplate 클래스와 함께 특별한 경우에 사용하는 래퍼wrapper 클래스다. 이 클래스에 대해서는 나중에 이야기하도록 하고 실제 예제를 살펴보기 전에 먼저 데이터베이스 스키마를 생성하고 미리 테스트할 데이터를 입력해놓는 사전 작업을 수행하자.

혹 이미 테스트할 데이터베이스가 있다면 이 부분은 건너뛰어도 좋다.

### 1.3.1 MySQL 데이터베이스 스크립트

이 책에 나오는 예제는 모두 MySQL 데이터베이스를 사용했으며, MySQL의 설치 매뉴얼만 제대로 따른다면 데이터베이스를 설치하기는 쉽다.

일단 데이터베이스를 제대로 설치하면 이 책의 소스 코드에서 제공하는 SQL 스크립트를 꼭 실행해보자. 해당 스크립트는 JSDATA라는 데이터베이스를 생성하고 ACCOUNTS, TRADES, PRICES 등과 같은 필수 테이블을 생성한다. 이미 다른 데이터베이스를 사용한다면 해당 스크립트는 아마 문제없이 실행될 것이다(하지만 개인적으로 이 스크립트를 테스트해보지는 못했다).

다음으로 중요한 사항은 DataSource 인터페이스의 빈을 생성하는 것이다. DataSource 빈은 데이터베이스 제품 정보를 캡슐화하며 데이터베이스와 통신할 수 있도록 연결하는 연결 팩토리처럼 동작한다. 이는 다음 예제처럼 URL이나 사용자 이름, 비밀번호와 같은 드라이버 정보를 포함해 생성되어야 한다. 만약 MySQL이 아니고 다른 데이터베이스를 사용한다면 DataSource 빈을 생성하는 데 필요한 제품(드라이버) 정보를 제공하는 것을 잊지 말자.

다음 datasource-beans.xml은 MySQL 데이터베이스에 DataSource 인터페이스의 빈을 생성하는 코드다.

```
<bean id="mySqlDataSource"
  class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="com.mysql.jdbc.Driver" />
  <property name="url" value="jdbc:mysql://localhost:3306/JSDATA" />
  ....
</bean>
```

class 속성은 DataSource 인터페이스의 구현 클래스(여기서는 아파치 공통 라이브러리 DBCP 프로젝트의 BasicDataSource 클래스)를 설정하며, <property /> 요소의 driverClassName 속성값은 데이터베이스에 사용할 클래스를 가리킨다. 전체 코드에 대해서는 잠시 후에 설명하기로 하자.

이 책에서는 DBCP 데이터소스를 사용할 것이며, 이는 http://commons.apache.org/dbcp에서 다운로드할 수 있다. 이미 메이븐Maven을 사용한다면 DBCP와 MySQL connector jar를 포함하기 위한 다음 코드를 pom.xml에 추가하도록 하자(전체 pom.xml 코드는 이 책의 예제 파일에서 확인할 수 있다).

```
<!-- pom.xml -->
<dependency>
  <groupId>commons-dbcp</groupId>
  <artifactId>commons-dbcp</artifactId>
  <version>1.4</version>
</dependency>
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.18</version>
</dependency>
```

### 1.3.2 JdbcTemplate 클래스 사용

JdbcTemplate는 데이터 조회, 삽입, 삭제와 같은 데이터 액세스 작업에 특화된 스프링 프레임워크의 클래스다. 스프링 데이터 액세스 프레임워크의 기반이 되는 클래스이기도 하므로 지금부터 자세히 알아보도록 하자.

JdbcTemplate는 스레드 안전 클래스[02]이므로 스레드 간 쉽게 공유될 수 있는 특징이 있다. 또한 JdbcTemplate 클래스의 큰 장점 중 하나는 리소스를 해제할 수 있는 능력이다. 개발자 대부분은 JDBC 연결이나 그 외 관련 리소스를 종료시키는 것을 종종 잊어버리는 경우가 많은데, 이는 많은 문제를 초래할 수 있다. 하지만 JdbcTemplate 클래스는 이와 같은 번거로운 해제 작업을 대신 처리하여 개발자를 구원해준다.

JdbcTemplate 클래스를 본격적으로 사용하기 전에는 DataSource 인터페이스의 객체를 구성해야 한다. JdbcTemplate 클래스에 DataSource 객체가 반드시

02 역자주_여러 개의 스레드가 접근하더라도 작업 실행의 안정성을 보장하는 클래스를 말한다.

구성되어야 (개발자가 볼 수 없는 곳에서) 템플릿 클래스가 데이터 소스에 연결하는 작업과 구문statement을 생성할 수 있다.

### DataSource 빈 구성

알다시피 javax.sql.DataSource는 특정 데이터베이스 제품에 필요한 연결 세부사항을 결정하는 인터페이스다. 각 제품에는 자체적으로 구현한 클래스를 가지고 있으며, 이 클래스는 보통 jar 파일로 제공된다. 예를 들어 MySQL 드라이버 클래스는 com.mysql.jdbc.Driver 클래스로 정의한다.

다음 코드는 MySQL에 데이터 소스를 지정하는 방법의 한 예다.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">
  <!-- mySqlDataSource 빈 -->
  <bean id="mySqlDataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/JSDATA" />
    <property name="username" value="jsuser" />
    <property name="password" value="jsuser" />
  </bean>
</beans>
```

위 코드는 jdbc:mysql://localhost:3306/JSDATA라는 로컬 호스트에서 실행되는 MySQL 데이터베이스를 가리키는 mySqlDataSource 빈을 생성한다. 만약

MySQL이 아닌 다른 데이터베이스 제품을 사용한다면 value 속성값으로 각 데이터베이스 제품에 대응하는 적절한 값을 설정해 다른 빈을 생성해야만 한다.

### JdbcTemplate 클래스 구성

데이터 소스를 구성했다면 다음으로 JdbcTemplate 클래스를 생성하고 이 클래스로 작업을 수행해야 한다.

JdbcTemplate 클래스는 두 가지 방법으로 생성할 수 있다. 하나는 클래스에 인스턴스를 생성하여 이미 구성된 데이터 소스를 제공하는 것이며, 다른 하나는 설정 파일에 빈을 생성하고 인스턴스화하여 이를 데이터 액세스 객체Data Access Object, DAO 클래스에 주입하는 것이다. DAO 클래스란 데이터에 접근하기 위해 데이터베이스와 통신하는 클래스를 말한다.

그럼 앞서 만든 DataSource 빈과 함께 JdbcTemplate 클래스를 인스턴스화하는 예제를 살펴보자.

```
public class JdbcTemplateTest {
  private ApplicationContext ctx = null;
  private JdbcTemplate template = null;
  private DataSource datasource = null;

  public JdbcTemplateTest() {
    // datasources-beans.xml 파일로 빈을 형성하는 컨테이너를 생성함
    ctx = new ClassPathXmlApplicationContext("datasources-beans.xml");

    // 컨테이너로부터 mySqlDataSource 빈을 가져옴
    datasource = ctx.getBean("mySqlDataSource", DataSource.class);

    // dataSource 객체를 인자로 취한 JdbcTemplate 클래스를 인스턴스화함
    template = new JdbcTemplate(datasource);
```

```
  }

  public static void main(String[] args) {
    JdbcTemplateTest t = new JdbcTemplateTest();

    // 데이터에 접근하는 메서드를 여기에서 실행함
    ....
  }
}
```

절차는 매우 간단하다.

- 데이터 소스를 가진 설정 파일에서 애플리케이션 컨텍스트를 로드하여 가져온다(이 예제에서는 datasouces-beans.xml이 설정 파일이다).
- new 연산자를 사용해서 생성자에게 mySqlDataSource 빈을 제공하는 JdbcTemplate 클래스를 생성한다.

위 절차에 따라 JdbcTemplate 클래스 구성이 완료되고 기능을 모두 구현했다면 이제 이 클래스를 사용해 데이터베이스 테이블에 접근할 준비가 된 것이다. 이 외에도 JdbcTemplate 클래스는 좀 더 자세히 살펴봐야 하는 많은 기능이 있다.

#### JdbcTemplate 클래스로 작업

JdbcTemplate 클래스는 데이터 세트에 다양한 방법으로 접근할 수 있는 100개 이상의 메서드를 가지고 있다. 예를 들어 테이블을 생성하거나 데이터를 삽입하는 것과 같은, 즉시 실행할 수 있는 쿼리문을 사용하고 싶다면 JdbcTemplate 클래스의 execute 메서드를 사용하면 된다.

마찬가지로 하나 이상의 데이터 행을 조회하고 싶다면 queryForXXX 메서드를 사용해야 한다. 이 외에도 여러 가지 메서드가 있으며 그중에는 이름 자체로 기능을 이해할 수 있기도 하고 몇몇 메서드는 JavaDoc을 이용했을 때 어떤 메서드인지 쉽게 이해할 수 있는 것들도 있다.

지금부터 이 중 가장 중요한 메서드를 설명하도록 하겠다.

**하나 이상의 행 조회하기** TRADES라는 테이블에 몇 개의 행이 있는지 알아보는 것이 애플리케이션의 요구사항이라고 생각해보자. 다음 코드는 JdbcTemplate 클래스를 가장 간단하게 사용할 수 있는 형태로, TRADES 테이블 안 모든 행 수가 몇 개인지 보여준다.

```
public int getTradesCount() {
  int numOfTrades = template.queryForInt("select count(*) from TRADES");
  return numOfTrades;
}
```

queryForInt 메서드는 테이블에서 count(*)에 해당하는 결과를 반환한다. 그러므로 반환 타입은 분명히 int일 것이다. 이러한 queryForXXX 형태의 메서드로는 queryForString, queryForLong, queryForMap, queryForObject가 있으며, 이는 열값을 적절한 자료형으로 변형해주는 기본 퍼실리티facility 메서드다.

물론 queryForObject와 같은 조금 더 일반적인 메서드를 사용하여 위 예제를 다시 작성할 수도 있다. 하지만 이 메서드는 다른 메서드와 다르게 반환 값의 타입을 결정하는 두 번째 매개변수(아래 예제에서는 Integer.class)를 가지고 있다. 예제에서는 count(*)가 int 타입을 반환하므로 메서드 호출 시 Integer 클래스를 두 번째 인자로 넘겨주어야 한다.

다음 예제는 위 설명이 실제 어떤 방식으로 이루어지는지 보여준다.

```
public int getTradesCount() {
  int numOfTrades =
    template.queryForObject("select count(*) from TRADES", Integer.class);
  return numOfTrades;
}

// queryForObject 메서드를 사용하여 TRADES 테이블 안 행의 max id를 얻음
public int getTradeMaxId() {
  int maxId = template.queryForObject(
    "select max(id) from TRADES", Integer.class
  );
  return maxId;
}
```

위 예제는 queryForObject 메서드를 사용하여 TRADES 테이블 안의 최대 ID 값을 가진 행을 조회하는 코드다. queryForLong과 queryForString 메서드 또한 이와 같은 방식으로 각각 Long과 String 타입의 값을 반환한다.

queryForMap 메서드는 다음 예제처럼 Map<String, Object> 형식의 단일 행을 반환한다.

```
public Map<String, Object> getTradeAsMap() {

  // 여기에서 ID를 하드코딩한다는 점에 유의!
  Map<String, Object> tradeAsMap =
    template.queryForMap("select * from TRADES where id=1");
  System.out.println("Trades Map:" + tradeAsMap);
  return tradeAsMap;
```

```
}

// 콘솔의 출력값은 다음과 같다
Trades Map:{ ID=1, ACCOUNT=1234AAA, SECURITY=MDMD,
   QUANTITY=100000, STATUS=NEW, DIRECTION=BUY },
```

위 코드를 살펴보면 각 열의 이름은 String 타입으로 표기한 키값이며, 해당 값은 Map<String, Object>에 선언했듯이 Object 타입이다. 하지만 queryForList 메서드는 다른 메서드와는 조금 달리 Map<String, Object> 형식의 리스트 타입으로 여러 행을 반환할 수 있다.

그럼 실제로 어떻게 동작하는지 살펴보자. getAllTrades 메서드는 모든 trades 객체(TRADES 테이블 안의 모든 행)를 가져와 콘솔에 출력하는 메서드다.

```
public List<Map<String, Object>> getAllTrades() {
  List<Map<String, Object>> trades =
    template.queryForList("select * from TRADES");
  System.out.println("All Trades:" + trades);
  return trades;
}

// 콘솔에는 다음처럼 출력됨
All Trades:
[{ ID=1, ACCOUNT=1234AAA, ... STATUS=NEW, DIRECTION=BUY },
  ...,
{ ID=5, ACCOUNT=452SEVE, ... STATUS=NEW, DIRECTION=SELL }]
```

물론 위 예제에서 사용한 쿼리문은 매우 간단하지만 이와 같은 방식으로 복잡한 쿼리문도 작성할 수 있다. 이처럼 복잡한 쿼리문을 실행하려면 where 구절이나 그 외 SQL 구문을 자주 사용하는데 where 구절의 경우 입력 변수가 반드시 지정되어

있어야 한다. 그렇다면 이러한 바인드 변수들을 매개변수화할 수 있는 방법이 있을까? 지금부터 이를 살펴보자.

**바인드 변수** 바인드 변수는 동적인 SQL 쿼리를 생성하는 데 도움을 준다. 보통은 여러 가지 조건에 따라 레코드를 가져와야 할 경우 SQL 스크립트에 where 구절을 사용한다. 이때 바인드 변수는 인라인 방식으로 변수를 사용할 때와는 다르게 SQL 삽입 공격SQL injection attack으로부터 애플리케이션을 보호할 수 있으므로 개발자들이 선호하는 편이다.

예를 들어 id가 5인 TRADES 테이블 행의 STATUS 값을 얻고 싶다면 다음처럼 SQL 구문을 작성하면 된다.

```
public String getTradeStatus(int id) {
  String status = template.queryForObject(
    "select STATUS from TRADES where id= ?", new Object[]{id}, tring.class
  );
  return status;
}
```

위 쿼리에서 ? 연산자는 프레임워크에게 해당 메서드의 두 번째 매개변수에 지정한 값으로 대체될 수 있다는 것을 말해주며 위 예제에서는 id 변수 값이다. 이를 위해서는 전달받는 id 값을 가지고 Object 배열을 생성하면 된다. 세 번째 매개변수는 메서드 안의 쿼리가 반환할 것이라고 예상하는 값의 타입이다. 이 예제의 경우 STATUS 값이 String 타입이므로 세 번째 매개변수를 String.class로 표기했다.

또한 하나 이상의 바인드 변수를 제공하는 것도 가능하며 개수에는 아무 제한이 없다. 다음 예제 코드에서 오버로딩한 getTradeStatus 메서드는 쿼리의 where 구절에 두 개의 조건식이 있으며, queryForObject 메서드의 두 번째 매개변수인 Object 배열을 통해 두 번째 바인드 변수값 또한 제공하는 것을 볼 수 있다.

```
public String getTradeStatus(int id, String security) {
  String status = template.queryForObject(
    "select STATUS from TRADES where id = ? and security=?",
    new Object[]{id, security}, String.class
  );
  return status;
}
```

**행을 도메인 객체로 매핑하기** 지금이라면 TRADES 테이블의 모든 행이 Trade 도메인 객체로 표기된다는 것을 알았을 것이다. 하지만 TRADE 테이블의 값을 Trade 객체를 통해 가져오는 것은 이전 예제에서 살펴보았지만 아직 어떻게 레코드의 각 행을 Trade 객체로 생성하는지는 살펴보지 못했다.

레코드의 각 행을 객체로 생성하기 위해서는 스프링 프레임워크에서 제공하는 RowMapper (콜백) 인터페이스를 사용해야만 한다. RowMapper 인터페이스는 전달받는 행을 도메인 객체로 매핑할 때 사용하는 mapRow 메서드만 가지고 있다. 또한 RowMapper는 익명 클래스로써 생성해도 되고 따로 RowMapper 인터페이스를 구현하는 클래스를 직접 만들어도 된다.

각각의 예를 지금부터 살펴보도록 하자. 먼저 RowMapper 인터페이스를 구현하고 mapRow 메서드를 선언하는 TradeMapper 클래스를 생성해보자.

```
private static final class TradeMapper implements RowMapper<Trade> {
  @Override
  public Trade mapRow(ResultSet rs, int rowNum) throws SQLException {
    Trade t = new Trade();
    // ResultSet 클래스의 getXXX 메서드를 사용해서 값을 지정함
    t.setId(rs.getInt("ID"));
    ....
```

```
    return t;
  }
}
```

mapRow 메서드에서 현재 행의 ResultSet 인터페이스의 인스턴스는 콜백 클래스를 통해서 제공된다. 그러므로 여기서 해야 할 일은 ResultSet 객체인 rs로부터 열 데이터를 추출해내고 새롭게 인스턴스화한 Trade 클래스의 도메인 객체인 t에 이 값을 넣어 주는 것이다. 그러면 해당 메서드는 완전히 초기화된 Trade 객체를 반환할 것이다.

지금까지 RowMapper 인터페이스의 구현 클래스를 만들었으니 다음에는 오버로딩한 queryForObject 메서드로 TRADES 테이블에서 생성한 trade 객체(TRADES 테이블의 모든 값) 모두를 검색해보자.

```
public Trade getMappedTrade(int id) {
  Trade trade = template.queryForObject(
    "select * from TRADES where id = ?", new Object[]{id}, new TradeMapper()
  );
  return trade;
}
```

위 코드의 queryForObject 메서드에 세 번째 인자가 있음을 확인했는가? 이 메서드는 ResultSet 인스턴스에서 추출한 열값을 가지고 Trade 객체를 생성하는 TradeMapper 클래스를 세 번째 인자로 받는다(세 번째 매개변수는 메서드의 쿼리가 반환할 것이라고 예상하는 값의 타입이라고 설명했던 것을 상기하자). 이 콜백 클래스의 장점은 열 데이터를 Trade 객체로 변경할 때 필요한 RowMapper 인터페이스를 어떤 메서드에서든 사용할 수 있다는 것이다.

이번에는 RowMapper 인터페이스를 사용하는 또 다른 방법을 살펴보자. 위처럼 개별의 인스턴스를 생성하지 않고 RowMapper 인스턴스를 생성하는 익명 클래스를 사용하는 방법이다. 다음은 이 방법을 이용하는 코드다.

```
public Trade getTrade(int id) {
  Trade trade = template.queryForObject(
    "select * from TRADES where id= ?",
    new Object[]{id},
    new RowMapper<Trade>() {
      @Override
      public Trade mapRow(ResultSet rs, int row) throws SQLException {
        Trade t = new Trade();
        t.setId(rs.getInt("ID"));
        t.setAccount(rs.getString("ACCOUNT"));
        ....
        t.setDirection(rs.getString("DIRECTION"));
        return t;
      }
    }
  );
  return trade;
}
```

익명의 클래스로서 인라인 방식으로 인스턴스화된 RowMapper 인스턴스는 이전 예제와 정확히 같은 작업을 실행한다. 또한 mapRow 메서드의 두 번째 인자는 콜백 클래스의 객체에 제공했던 레코드의 행 숫자에 해당하는 값이란 걸 기억하자. 또한 콜백 클래스의 객체에 주어진 ResultSet 인스턴스는 단 하나의 레코드만을 가지고 있다는 점에 주의하자. 즉, ResultSet.next 메서드를 사용하면 SQLException 클래스에서 오류가 발생했음을 알릴 것이다.

그리고 RowMapper 클래스를 익명으로 생성하면 애플리케이션 안 다른 위치에서는 사용할 수 없다는 제한이 따른다는 걸 기억하자. 그러므로 익명 클래스를 사용해야 하는 특정한 이유가 없다면 TradeMapper와 같은 개별 클래스를 생성하는 방법을 이용해 클래스를 재사용하는 것이 좋다. 이는 큰 장점이다!

JdbcTemplate과 RowMapper 클래스는 모두 스레드 안전 클래스라는 점을 기억하자. 그러므로 상태 손상의 우려 없이 여러 스레드 사이에 이 클래스를 공유하여 사용할 수 있다.

**Trade 리스트 가져오기** 지금까지는 하나의 레코드를 가져와 도메인 객체에 매핑하는 방법을 알아보았으니 도메인 객체에 매핑된 모든 행의 리스트를 어떻게 가져오는지 확인해보자. 사실 RowMapper 클래스가 이미 설계되었기에 이 절차는 매우 쉬울 것이다.

다음 코드는 리스트를 trades라는 객체로 가져오기 위해 사용하는데 이전 예제와의 차이점은 queryForXXX 메서드 대신에 query 메서드를 사용했다는 것이다.

```
public List<Trade> getAllMappedTrades() {
  List<Trade> trades =
    template.query("select * from TRADES", new TradeMapper());
  return trades;
}
```

여기서 받은 각 행의 값은 TradeMapper 클래스를 통해 Trade 클래스의 객체인 trades로 각각 만들어진 후에 리스트에 추가될 것이다. 매우 간단하지 않은가?

이처럼 데이터에 접근하는 방법은 다양하다. 지금까지 다양한 조회 메커니즘을 살펴보았으니 다음부터는 수정과 삭제 작업에 대해서도 다뤄보도록 하자.