# Spring in Action 제4판

Spring 4.1 업데이트 FOURTH EDITION 스프링 인 액션

**Spring In Action 4th edition**

by Craig Walls

Original English language edition published by Manning Publications, USA.
Copyright © 2013 by Manning Publications Co.
Korean edition copyright © 2015 by J-Pub.
All rights reserved.

이 책의 한국어판 저작권은 대니홍 에이전시를 통한 저작권사와의 독점 계약으로 제이펍 출판사에 있습니다.
저작권법에 의해 한국 내에서 보호를 받는 저작물이므로 무단전재와 무단복제를 금합니다.

Spring in Action 제4판
Spring 4.1 FOURTH EDITION 스프링 인 액션

**초판 1쇄 발행** 2015년 12월 29일

**지은이** 크레이그 월즈
**옮긴이** 테크 트랜스 그룹 T4
**펴낸이** 장성두
**펴낸곳** 제이펍

**출판신고** 2009년 11월 10일 제406-2009-000087호
**주소** 경기도 파주시 문발로 141 뮤즈빌딩 403호
**전화** 070-8201-9010 / **팩스** 02-6280-0405
**홈페이지** www.jpub.kr / **이메일** jeipub@gmail.com

**편집부** 이민숙, 이 슬, 이주원 / **소통·기획팀** 민지환, 현지환
**용지** 신승지류유통 / **인쇄** 한승인쇄사 / **제본** 광우제책사

**ISBN** 979-11-85890-38-8 (93000)
**값** 35,000원

※ 이 책은 저작권법에 따라 보호를 받는 저작물이므로 무단 전재와 무단 복제를 금지하며,
이 책 내용의 전부 또는 일부를 이용하려면 반드시 저작권자와 제이펍의 서면동의를 받아야 합니다.
※ 잘못된 책은 구입하신 서점에서 바꾸어 드립니다.

**제이펍**은 독자 여러분의 아이디어와 원고 투고를 기다리고 있습니다. 책으로 펴내고자 하는 아이디어나 원고가 있으신 분께서는 책의 간단한 개요와 차례, 구성과 저(역)자 약력 등을 메일로 보내주세요. **jeipub@gmail.com**

# Spring in Action 제4판

Spring 4.1 업데이트 FOURTH EDITION

# 스프링 인 액션

크레이그 월즈 지음 / 테크 트랜스 그룹 T4 옮김

제이펍

**※ 드리는 말씀**

- 이 책에 기재된 내용을 기반으로 한 운용 결과에 대해 저/역자, 소프트웨어 개발자 및 제공자, 제이펍 출판사는 일체의 책임을 지지 않으므로 양해 바랍니다.
- 이 책에 기재한 회사명 및 제품명은 각 회사의 상표 및 등록명입니다.
- 이 책에서는 ©, ®, ™ 등의 기호를 생략하고 있습니다.
- 이 책에서 사용하고 있는 실제 제품 버전은 독자의 학습 시점에 따라 책의 버전과 다를 수 있습니다.
- 이 책의 예제 코드는 다음 사이트에서 다운로드할 수 있습니다.

  – https://github.com/Jpub/SpringInAction
- 책의 내용과 관련된 문의사항은 옮긴이나 출판사로 연락주시기 바랍니다.

  – 옮긴이: asinayo73@hotmail.com
  – 출판사: jeipub@gmail.com

# 차례


옮긴이 머리말 ····· xiv
머리말 ····· xvi
감사의 글 ····· xviii
이 책에 대하여 ····· xix
베타리더 후기 ····· xxiv

PART 1 코어 스프링

CHAPTER 1 스프링 속으로 3

1.1 자바 개발 간소화 ····· 4
1.1.1 POJO의 힘 5
1.1.2 종속객체 주입 6
1.1.3 애스펙트 적용 13
1.1.4 템플릿을 이용한 상투적인 코드 제거 18
1.2 빈을 담는 그릇, 컨테이너 ····· 21
1.2.1 또 하나의 컨테이너, 애플리케이션 컨텍스트 22
1.2.2 빈의 일생 24
1.3 스프링 현황 ····· 25
1.3.1 스프링 모듈 26
1.3.2 스프링 포트폴리오 29
1.4 스프링의 새로운 기능 ····· 33
1.4.1 스프링 3.1에서 새로워진 기능 33
1.4.2 스프링 3.2에서 새로워진 기능 34
1.4.3 스프링 4.0에서 새로워진 기능 36
1.5 요약 ····· 37

CHAPTER 2 빈 와이어링(묶기) 39

2.1 스프링 설정 옵션 알아보기 ····· 40
2.2 자동으로 빈 와이어링하기 ····· 41
2.2.1 발견 가능한 빈 만들기 42
2.2.2 컴포넌트 스캔된 빈 명명하기 46
2.2.3 컴포넌트 스캐닝을 위한 베이스 패키지 세팅 47
2.2.4 오토와이어링되는 빈의 애너테이션 48
2.2.5 자동 설정 검증하기 51
2.3 자바로 빈 와이어링하기 ····· 52
2.3.1 설정 클래스 만들기 53
2.3.2 간단한 빈 선언하기 53
2.3.3 JavaConfig 주입하기 55

**2.4 빈을 XML로 와이어링하기** 57
2.4.1 XML 설정 스펙 만들기 57
2.4.2 간단한 빈 선언 58
2.4.3 생성자 주입을 사용하여 빈 초기화하기 60
2.4.4 프로퍼티 세팅 67
**2.5 설정 가져오기와 믹싱하기** 73
2.5.1 JavaConfig로 XML 설정 참조하기 73
2.5.2 XML 설정에서 JavaConfig 참조하기 75
**2.6 요약** 77

CHAPTER 3 고급 와이어링 79

**3.1 환경과 프로파일** 79
3.1.1 빈 프로파일 설정하기 81
3.1.2 프로파일 활성화하기 86
**3.2 조건부 빈** 88
**3.3 오토와이어링의 모호성** 92
3.3.1 기본 빈 지정 94
3.3.2 오토와이어링 빈의 자격 95
**3.4 빈 범위** 100
3.4.1 요청과 세션 범위 작업하기 101
3.4.2 XML로 범위 프록시 선언하기 103
**3.5 런타임 값 주입** 104
3.5.1 외부 값 주입 105
3.5.2 스프링 표현식 와이어링 110
**3.6 요약** 118

CHAPTER 4 애스펙트 지향 스프링 121

**4.1 AOP란 무엇인가?** 123
4.1.1 AOP 용어 정의 124
4.1.2 스프링의 AOP 지원 127
**4.2 포인트커트를 이용한 조인 포인트 선택** 129
4.2.1 포인트커트 작성 130
4.2.2 포인트커트에서 빈 선택하기 132
**4.3 애스펙트 애너테이션 만들기** 132
4.3.1 애스펙트 정의하기 133
4.3.2 around 어드바이스 만들기 138
4.3.3 어드바이스에서 파라미터 처리하기 139
4.3.4 인트로덕션 애너테이션 143
**4.4 XML에서 애스펙트 선언하기** 145
4.4.1 before 어드바이스와 after 어드바이스 선언하기 147
4.4.2 around 어드바이스 선언 150
4.4.3 어드바이스에 파라미터 전달 151
4.4.4 애스펙트를 이용한 새로운 기능 도입 153

**4.5 AspectJ 애스펙트 주입** 155
**4.6 요약** 158

PART 2 웹에서의 스프링

CHAPTER 5 스프링 웹 애플리케이션 만들기 161

**5.1 스프링 MVC 시작하기** 162
5.1.1 스프링 MVC를 이용한 요청 추적 162
5.1.2 스프링 MVC 설정하기 164
5.1.3 Spittr 애플리케이션 소개 170
**5.2 간단한 컨트롤러 작성하기** 170
5.2.1 컨트롤러 테스팅 173
5.2.2 클래스 레벨 요청 처리 정의하기 174
5.2.3 뷰에 모델 데이터 전달하기 176
**5.3 요청 입력받기** 182
5.3.1 쿼리 파라미터 입력받기 183
5.3.2 패스 파라미터를 통한 입력받기 185
**5.4 폼 처리하기** 188
5.4.1 폼 처리 컨트롤러 작성 191
5.4.2 폼 검증하기 194
**5.5 요약** 198

CHAPTER 6 웹 뷰 렌더링 199

**6.1 뷰 리졸루션 이해하기** 199
**6.2 JSP 뷰 만들기** 202
6.2.1 JSP-ready 뷰 리졸버 설정하기 203
6.2.2 스프링의 JSP 라이브러리 사용하기 205
**6.3 아파치 타일즈 뷰로 레이아웃 정의하기** 221
6.3.1 타일 뷰 리졸버 설정하기 222
**6.4 Thymeleaf로 작업하기** 228
6.4.1 Thymeleaf 뷰 리졸버 설정하기 228
6.4.2 Thymeleaf 템플릿 정의하기 230
**6.5 요약** 235

CHAPTER 7 고급 스프링 MVC 237

**7.1 다양한 스프링 MVC 설정** 238
7.1.1 DispatcherServlet 설정 사용자 정의하기 238
7.1.2 서블릿과 필터 추가하기 239
7.1.3 web.xml에서 DispatcherServlet 선언하기 241

**7.2 멀티파트 폼 데이터 처리하기** 244
7.2.1 멀티파트 리졸버 설정하기 246
7.2.2 멀티파트 요청 처리하기 250
**7.3 예외 처리하기** 255
7.3.1 예외를 HTTP 상태 코드에 매핑하기 255
7.3.2 예외-처리 메소드 작성하기 257
**7.4 어드바이징 컨트롤러** 259
**7.5 리다이렉션되는 요청 간의 데이터 전달하기** 261
7.5.1 URL 템플릿과 함께 리다이렉션하기 262
7.5.2 플래시 애트리뷰트로 작업하기 263
**7.6 요약** 265

CHAPTER 8 스프링 웹 플로로 작업하기 267

**8.1 스프링에 웹 플로 설정하기** 268
8.1.1 플로 실행기 연결하기 269
8.1.2 플로 설정하기 269
8.1.3 플로 요청 처리하기 270
**8.2 플로의 컴포넌트** 271
8.2.1 상태 272
8.2.2 전이 275
8.2.3 플로 데이터 277
**8.3 모든 것을 합쳐 보기: 피자 플로** 279
8.3.1 기본 플로 정의하기 279
8.3.2 고객 정보 수집하기 283
8.3.3 주문 만들기 290
8.3.4 결제 처리하기 293
**8.4 웹 플로 보안** 295
**8.5 요약** 295

CHAPTER 9 웹 애플리케이션 보안 297

**9.1 스프링 시큐리티 시작하기** 298
9.1.1 스프링 시큐리티 모듈의 이해 299
9.1.2 웹 요청 필터하기 300
9.1.3 간단한 보안 설정 작성하기 301
**9.2 사용자 상세 서비스 선택** 304
9.2.1 인메모리 사용자 저장소로 작업하기 305
9.2.2 데이터베이스 테이블로 인증하기 307
9.2.3 LDAP 기반 인증 적용하기 310
9.2.4 사용자 정의 사용자 서비스 설정하기 314
**9.3 요청 가로채기** 316
9.3.1 스프링 표현식 보안 319
9.3.2 채널 보안 적용하기 321
9.3.3 사이트 간 요청 위조 방지하기 322

**9.4 사용자 인증하기** 324
9.4.1 사용자 정의 로그인 페이지 추가하기 325
9.4.2 HTTP 기본 인증 활성화하기 326
9.4.3 기억하기 기능 활성화 327
9.4.4 로그아웃하기 329
**9.5 뷰 보안하기** 330
9.5.1 스프링 시큐리티 JSP 태그 라이브러리 사용하기 330
9.5.2 Thymeleaf의 스프링 시큐리티 언어로 작업하기 334
**9.6 요약** 336

PART 3 백엔드에서의 스프링

CHAPTER 10 스프링과 JDBC를 사용하여 데이터베이스 사용하기 339

**10.1 스프링의 데이터 액세스 철학** 340
10.1.1 스프링의 데이터 액세스 예외 계층 구조 341
10.1.2 데이터 액세스 템플릿화 346
**10.2 데이터 소스 설정** 348
10.2.1 JNDI 데이터 소스 이용 349
10.2.2 풀링 기능이 있는 데이터 소스 사용하기 350
10.2.3 JDBC 드라이버 기반 데이터 소스 351
10.2.4 임베디드 데이터 소스 사용하기 353
10.2.5 데이터 소스 선택을 위한 프로파일링하기 354
**10.3 스프링과 JDBC** 357
10.3.1 지저분한 JDBC 코드 해결 357
10.3.2 JDBC 템플릿과 놀아 보자 361
**10.4 요약** 367

CHAPTER 11 객체 관계형 매핑을 통한 데이터 퍼시스팅 369

**11.1 스프링과 하이버네이트 통합** 371
11.1.1 하이버네이트 세션 팩토리 선언 371
11.1.2 스프링으로부터 해방된 하이버네이트 구성 374
**11.2 스프링과 자바 퍼시스턴스 API** 376
11.2.1 엔티티 관리자 팩토리 설정 376
11.2.2 JPA 기반 저장소 작성 383
**11.3 스프링 데이터를 사용한 자동 JPA 저장소** 386
11.3.1 쿼리 메소드 정의하기 388
11.3.2 맞춤형 쿼리 선언하기 392
11.3.3 맞춤형 기능 혼합 394
**11.4 요약** 396

CHAPTER 12 NoSQL 데이터베이스 사용하기 397

12.1 MongoDB의 유지성 도큐먼트 398
12.1.1 MongoDB 활성화 399
12.1.2 MongoDB 퍼시스턴트를 위한 애너테이션 모델 타입 403
12.1.3 MongoTemplate을 사용하여 MongoDB 액세스하기 407
12.1.4 MongoDB 저장소 작성하기 408
12.2 Neo4j로 그래프 데이터 사용하기 414
12.2.1 스프링 데이터 Neo4j 설정하기 415
12.2.2 그래프 엔티티 애너테이션하기 418
12.2.3 Neo4jTemplate 사용하기 422
12.2.4 자동 Neo4j 저장소 만들기 424
12.3 Redis에서 키-값 데이터 사용하기 429
12.3.1 Redis에 연결하기 430
12.3.2 RedisTemplate을 가지고 작업하기 431
12.3.3 키와 값의 직렬 변환 설정 436
12.4 요약 437

CHAPTER 13 데이터 캐싱하기 439

13.1 캐시 지원하기 440
13.1.1 캐시 매니저 설정하기 442
13.2 캐싱을 위한 애너테이션 메소드 446
13.2.1 캐시 채우기 447
13.2.2 캐시 엔트리 삭제하기 452
13.3 XML에서 캐싱 선언하기 453
13.4 요약 457

CHAPTER 14 시큐리티 메소드 459

14.1 애너테이션을 사용한 시큐어 메소드 460
14.1.1 @Secure를 이용한 메소드 보안 460
14.1.2 JSR-250의 @RolesAllowed 사용 462
14.2 메소드 레벨 시큐리티를 위한 표현식 사용하기 463
14.2.1 메소드 액세스 규칙 표현하기 464
14.2.2 필터링 메소드의 입출력 467
14.3 요약 472

## PART 4 스프링 통합


CHAPTER 15 원격 서비스 사용하기 477

15.1 스프링 리모팅 개요 478
15.2 RMI 활용 481
15.2.1 RMI 서비스 익스포트 481
15.2.2 RMI 서비스 와이어링 484
15.3 Hessian과 Burlap을 이용한 리모트 서비스 노출 487
15.3.1 Hessian과 Burlap을 이용한 빈 기능 노출 488
15.3.2 Hessian/Burlap 서비스에 액세스하기 492
15.4 스프링의 HttpInvoker 사용하기 494
15.4.1 빈을 HTTP 서비스로 익스포트하기 494
15.4.2 HTTP를 거쳐 서비스에 액세스하기 496
15.5 웹 서비스의 발행과 소비 497
15.5.1 스프링을 사용할 수 있는 JAX-WS 엔드포인트 생성 498
15.5.2 클라이언트 측에서 JAX-WS 서비스 프록시하기 502
15.6 요약 504

CHAPTER 16 스프링 MVC로 REST API 사용하기 505

16.1 휴식(REST)을 취하다 506
16.1.1 REST의 기본 개념 506
16.1.2 스프링이 REST를 지원하는 방법 508
16.2 첫 번째 REST 엔드포인트 만들기 508
16.2.1 리소스 표현 협상 511
16.2.2 HTTP 메시지 변환기 사용 518
16.3 더 많은 리소스 사용하기 524
16.3.1 클라이언트와 에러 처리하기 524
16.3.2 응답 시의 헤더 설정하기 530
16.4 REST 리소스 사용하기 533
16.4.1 RestTemplate 동작 살펴보기 534
16.4.2 리소스 GET하기 535
16.4.3 리소스 조회 536
16.4.4 응답 메타데이터 추출 537
16.4.5 리소스 PUT하기 539
16.4.6 리소스 DELETE하기 540
16.4.7 리소스 데이터 POST하기 541
16.4.8 POST 요청으로부터 객체 응답 수신하기 542
16.4.9 POST 요청 후 리소스 위치 수신하기 543
16.4.10 리소스 교환 544
16.5 요약 547

CHAPTER 17 스프링 메시징 549

17.1 비동기 메시징에 대한 간단한 소개 550
17.1.1 메시지 보내기 551
17.1.2 비동기 메시징의 장점 554
17.2 JMS로 메시지 보내기 556
17.2.1 스프링에서 메시지 브로커 셋업하기 557
17.2.2 스프링의 JMS 템플릿 사용 559
17.2.3 메시지 드리븐 POJO 작성 569
17.2.4 메시지 기반의 RPC 활용 573
17.3 AMQP를 이용한 메시징 576
17.3.1 AMQP에 대한 간략한 소개 577
17.3.2 AMQP 메시징을 위한 스프링 설정하기 579
17.3.3 RabbitTemplate으로 메시지 전송하기 582
17.3.4 AMQP 메시지 수신하기 584
17.4 요약 588

CHAPTER 18 WebSocket과 STOMP를 사용하여 메시징하기 589

18.1 스프링의 하위 레벨 웹 소켓 API 사용하기 590
18.2 웹 소켓 지원 부족에 대해 대응하기 596
18.3 STOMP 메시징 작업하기 599
18.3.1 STOMP 메시징 사용하기 601
18.3.2 클라이언트로부터의 STOMP 메시지 처리 604
18.3.3 클라이언트로 메시지 보내기 608
18.4 사용자 타깃 메시지 사용하기 613
18.4.1 컨트롤러에서 사용자 메시지 사용하기 614
18.4.2 특정 사용자에게 메시지 보내기 616
18.5 예외 상황 메시지 처리하기 617
18.6 요약 619

CHAPTER 19 스프링을 사용하여 이메일 전송하기 621

19.1 이메일 전송을 위해 스프링 설정하기 622
19.1.1 메일 전송자 설정하기 622
19.1.2 메일 서버를 와이어링하고 사용하기 624
19.2 이메일 메시지를 풍부하게 꾸미기 625
19.2.1 첨부파일 추가하기 625
19.2.2 리치 콘텐츠를 이용한 이메일 전송 627
19.3 템플릿을 사용하여 이메일 생성하기 629
19.3.1 Velocity를 사용하여 이메일 메시지를 구성하기 629
19.3.2 이메일 메시지를 만들기 위해서 Thymeleaf 사용하기 632
19.4 요약 634

CHAPTER 20 JMX를 이용한 스프링 빈 관리 635

20.1 스프링 빈을 MBean으로 익스포트하기 636
20.1.1 이름으로 메소드 노출시키기 639
20.1.2 인터페이스를 이용한 MBean 오퍼레이션과 애트리뷰트 정의 642
20.1.3 애너테이션 주도의 MBean을 이용한 작업 643
20.1.4 MBean 충돌 처리 645
20.2 MBean 리모팅 646
20.2.1 원격 MBean 노출하기 647
20.2.2 원격 MBean에 액세스하기 648
20.2.3 MBean 프록시 만들기 650
20.3 통지 처리 651
20.3.1 통지 듣기 653
20.4 요약 654

CHAPTER 21 스프링 부트를 사용한 스프링 개발 간소화 655

21.1 스프링 부트 소개 656
21.1.1 스타터 의존 추가하기 657
21.1.2 자동 설정 660
21.1.3 스프링 부트 CLI 660
21.1.4 액추에이터 661
21.2 스프링 부트로 애플리케이션 만들기 662
21.2.1 요청 처리하기 665
21.2.2 뷰 만들기 668
21.2.3 정적 아티팩트 추가하기 669
21.2.4 데이터 보존하기 671
21.2.5 실행하기 673
21.3 스프링 부트 CLI로 그루비 사용하기 677
21.3.1 그루비 컨트롤러 작성하기 678
21.3.2 그루비 저장소로 데이터 보존하기 680
21.3.3 스프링 부트 CLI 실행 682
21.4 액추에이터로 애플리케이션 파악하기 684
21.5 요약 688

찾아보기 691

# 옮긴이 머리말

스프링 프레임워크는 자바 플랫폼을 위한 오픈 소스 애플리케이션 프레임워크로서 보통은 스프링(Spring)이라 부르며, 동적인 웹 사이트를 개발하기 위한 여러 서비스를 제공하고 있다. 스프링의 창시자인 로드 존슨(Rod Johnson)은 "항상 프레임워크 기반으로 접근하라."는 유명한 말을 했는데, 그의 말대로 스프링은 J2EE 애플리케이션 개발을 위한 책의 예제를 작성하면서 사용했던 프레임워크들을 발전시켜 지금에 이른 것이다.

스프링은 또한 자바 엔터프라이즈 개발을 편하게 해 주는 오픈 소스 경량급 애플리케이션 프레임워크이자 자바 개발을 위한 프레임워크로, 종속 객체를 생성해 주고 조립해 주는 도구의 역할을 담당한다.

이번 4판은 스프링의 새로운 릴리스인 4.0의 내용을 다루고 있는데, 매우 높은 수준의 메시지 기반 모델을 가지는 웹 소켓과 새로운 메시징 모듈도 포함한다. 또한, 자바 8의 특징인 람다식(lambdas)을 지원하며, 무엇보다도 읽기에 훨씬 깨끗하고 쉬운 콜백 인터페이스(JdbcTemplate과 함께 있는 RowMapper)로 작업할 수 있는 장점이 있다.

이 책에서는 스프링의 핵심부터 시작하여 애플리케이션 개발에 필요한 실전적인 내용까지 다루고 있다. 총 4개의 부(part)로 이루어져 있으며, 1부에서는 스프링의 기본 핵심 사항을 다루고, 2부에서는 웹 애플리케이션을 만들기 위한 내용을 다룬다. 3부에서는 백엔드 단을 다루며, JDBC, NoSQL 등의 내용도 같이 살펴본다. 4부에서는 스프링과 여러 관련 기술들의 통합 작업 방법을 사용자에게 제공하고 있다. 더불어 이 책에서 소개하는 종속객체 주입(DI)과 애스펙트 지향 프로그래밍(AOP)과 같은 기술을 활용하면 애플리케이션 개발에서의 어려움을 해결할 수 있을 것이다. 이러한 기술은 여러분에게 애플리케이션 개발에 유연함을 더해 주리라 생각한다.

스프링은 엔터프라이즈 애플리케이션 개발에 도움을 주기 위해 시작되었지만, 모든 자바 개발자들이 이를 활용할 수 있다. 독자들이 스프링을 사용하여 다양한 서비스를 구현할 때 이 책이 유용하게 사용되기를 바란다.

이 책을 만들면서 많은 도움을 주신 가족과 동료, 제이펍 대표님과 편집자께 깊은 감사를 드린다. 그동안 많은 날들을 함께 보내면서 지금까지의 우리를 있게 해 준 모든 관계자께도 지면으로나마 고마움을 표한다.

**테크 트랜스 그룹 T4**

# 머리말

'최고는 계속 나아지며 유지된다.'

스프링은 엔터프라이즈 자바 개발을 단순화하기 위한 야심찬 목표로 자바 개발 현장에 들어간 지 10여 년이 훌쩍 지났다. 이는 평범하고 오래된 자바 객체를 기반으로 헤비급 프로그래밍 모델을 가볍고 단순화된 프로그래밍 모델로 만들기 위한 도전이었다.

스프링은 몇 년간 꽤 많은 릴리스를 하면서 엔터프라이즈 애플리케이션 개발에 큰 영향을 끼쳤다. 그리고 자바 프로젝트를 위한 사실상의 표준 프레임워크가 되었다. 현재의 엔터프라이즈 자바 빈즈(EJB) 사양은 스프링이 도전하지 않은 이전 버전의 EJB 스펙과 매우 다름을 알 수 있다. 뿐만 아니라 스프링은 여전히 진화하고 있어서 자바 개발자에게 혁신적인 기능을 꾸준히 제공하고 있다. 스프링은 현상 유지에 도전하기 위해 출발하였지만, 이제는 현상을 뛰어넘으면서 자바 애플리케이션 개발을 위한 새로운 길을 닦고 있다.

그래서 이번 4판은 스프링의 최신 상황을 반영하려 애썼으며, 3판 출간 이후 지난 몇 년간 발생한 많은 내용이 추가되었다. 사실, 이 모든 변화를 책 한 권에 다루기는 불가능하다. 그럼에도 할 수 있는 만큼 준비하여 4판 개정 작업을 하였다. 이번 판에서 추가된 흥미롭고 새로운 몇 가지는 다음과 같다.

- 스프링 개발의 거의 모든 분야에 사용할 수 있는 자바 설정 옵션을 바탕으로 자바 기반의 스프링 설정에 중점을 둠
- 어떤 스프링 설정이 사용되고 무시되는지에 대해 실시간으로 결정할 수 있는 조건 설정과 프로파일
- REST 서비스 생성과 관련된 스프링 MVC에 대한 여러 개선 사항
- JSP의 대안인 Thymeleaf를 스프링 웹 애플리케이션에서 사용

- 자바 기반의 설정을 사용하여 스프링 보안 활성화
- JPA, MongoDB 및 Neo4j에 대한 실시간 저장소 구현을 자동 생성하기 위한 스프링 데이터 사용
- 스프링의 새 선언적 캐싱 지원
- WebSocket과 STOMP를 사용한 비동기 웹 메시징
- 스프링 부트, 스프링을 사용할 때의 새로운 접근 방법

스프링에 경험 많은 개발자라면 이러한 새로운 무기를 어렵지 않게 장착할 수 있을 것이다. 반면, 스프링 초보자에게는 스프링 자체를 배울 수 있는 좋은 시간이 될 것으로 생각한다.

실제로 스프링을 사용하면 흥미진진한 시간이 만들어진다. 스프링으로 개발하면서 많은 일이 일어났으며, 지난 십수 년간 수많은 내용이 작성되었다. 스프링이 다음엔 무엇을 제공할지 더는 기다릴 수 없다!

**크레이그 월즈(Craig Walls)**

# 감사의 글

이 책이 독자의 앞에 놓이기까지 전체 출판 과정을 관리하는 수많은 분의 손을 거쳤다. 이 모든 분의 손길이 없었다면 이 책의 탄생도 불가능했을 것이다.

우선, 매닝 식구들의 노고에 감사의 말을 전한다. 탈고하기까지 많은 압박을 받았지만, 이로 인해 이 책이 최고의 책으로 거듭날 수 있었다. Marjan Bace, Michael Stephens, Cynthia Kane, Andy Carroll, Benjamin Berg, Alyson Brener, Dottie Marisco, Mary Piergies, Janet Vail 외 많은 분께 감사의 말을 전한다.

피드백을 받는 것은 소프트웨어를 개발할 때와 마찬가지로 책을 쓸 때도 중요하다. 이 책에 아직도 거친 부분이 있지만, 여러 훌륭한 리뷰어 덕택에 최종 결과물이 한층 부드러워졌다. 다음의 사람들에게 감사를 드리고 싶다. Bob Casazza, Chaoho Hsieh, Christophe Martini, Gregor Zurowski, James Wright, Jeelani Basha, Jens Richter, Jonathan Thoms, Josh Hart, Karen Christenson, Mario Arias, Michael Roberts, Paul Balogh, Ricardo da Silva Lima.

John Ryan의 기술적인 사전 검토 진행에 특히 감사의 말을 전한다. 그리고 글 쓰는 내내 응원해 준 아내에게도 감사의 말을 전하며, 누구보다도 더 사랑한다는 말을 전하고 싶다.

이 책에 대한 다양한 관점의 인사이트를 제공한 Maisy와 Madi에게도 다시 감사의 말을 전한다. 스프링 팀의 동료에게도 감사의 말을 전한다.

마지막으로, 여러 나라를 강연하면서 만났던 스프링 사용자 그룹과 콘퍼런스 참여자에게도 감사의 말을 드린다.

# 이 책에 대하여

스프링 프레임워크는 "JEE 애플리케이션 개발을 더 쉽게 만들겠다."는 대단히 명확한 목적으로 만들어졌다. 이번 4판은 스프링을 더 쉽게 사용하는 법을 배울 수 있도록 집필되었다. 이 책은 스프링의 API 하나하나를 세세하게 이야기하지는 않는다. 대신, JEE 개발자에게 실감 나게 다가갈 수 있도록 실무 경험을 바탕으로 한 실용적인 코드 예제를 제공하는 방식으로 스프링 프레임워크를 보여 줄 것이다. 스프링은 잘 모듈화된 프레임워크라서 이 책 또한 같은 방식으로 집필되었다.

그렇더라도 필자는 모든 개발자가 필요로 하는 내용이 모두 같지 않다는 사실을 잘 안다. 스프링 프레임워크를 기본부터 차근차근 배우려는 사람도 있고, 어떤 주제를 골라서 자신이 원하는 부분만 읽으려는 사람도 있다. 따라서 스프링을 처음 배우는 사람에게는 잘 연마된 도구로, 특정 주제에 대해 더 깊이 파고들기를 원하는 사람에게는 지침서나 참고서로 활용될 수 있도록 작성했다.

이 책은 모든 자바 개발자를 대상으로 하지만, 엔터프라이즈 개발자에게 특히 유용하다. 각 장을 거치면서 코드 예제의 복잡도를 서서히 늘려가겠지만, 스프링의 진정한 힘은 엔터프라이즈 애플리케이션 개발을 더 쉽게 하는 데 있다. 따라서 엔터프라이즈 개발자들은 이 책에 있는 예제들의 진가를 알 수 있을 것이다. 스프링은 대체로 엔터프라이즈 서비스를 제공하는 데 기여하도록 되어 있기 때문에 스프링과 EJB 사이에 겹치는 부분이 많다.

## 로드맵

이번 4판은 4개의 부(part)로 구성되었다. 1부는 스프링 프레임워크의 필수적인 내용을 살펴본다. 2부는 스프링으로 웹 애플리케이션을 어떻게 만드는지를 보여 준다. 3부는 프런트엔드와 애플리케이션의 백엔드 부분을 보여 준다. 마지막 4부는 스프링이 다른 애플리케이션과

서비스에 어떻게 통합되는지를 보여 준다.

1부에서는 스프링 프레임워크의 필수인 스프링 컨테이너, 종속객체 주입(DI, Dependency Injection)과 애스펙트 지향 프로그래밍(AOP, Aspect-Oriented Programming)에 관해 알아본다. 그리고 이 책의 나머지 부분에 대한 기초를 다진다.

- 1장에서는 스프링 개요를 제공하고, DI와 AOP의 기본 예제를 포함한다. 스프링 에코시스템의 개요를 살펴볼 수 있다.
- 2장에서는 DI에 대해서 더 자세히 살펴보고, 애플리케이션(빈)의 컴포넌트들이 어떻게 서로 와이어링되는지에 대한 다양한 내용을 살펴본다. 이는 XML, 자바를 사용한 와이어링과 오토와이어링에 관한 내용을 포함한다.
- 빈 와이어링에 관한 기본적 내용과 더불어 3장에서는 여러 가지 고급 와이어링 기법을 제공한다. 자주 사용하지는 않겠지만, 스프링 컨테이너에서 강력한 파워를 얻을 수 있는지를 보여 준다.
- 4장에서는 스프링 AOP를 이용해 서비스하는 객체에서 횡단관심사(cross-cutting concerns)를 서로 분리하는 방법을 살펴본다. 또한, 4장 이후에서는 스프링 AOP를 이용해 트랜잭션이나 보안, 캐시 등의 서비스를 선언적 방식으로 개발하는 방법을 제공한다.

2부에서는 웹 애플리케이션을 만들기 위해 스프링을 어떻게 사용하는지를 설명한다.

- 5장에서는 스프링 MVC에 대한 기본 내용을 보여 주고, 스프링의 기능적 웹 프레임워크를 설명한다. 웹 요청을 다루기 위해 컨트롤러를 어떻게 만들어야 하는지와 모델 데이터 응답 방법을 볼 수 있다.
- 일단, 컨트롤러가 작업을 완료하고 모델 데이터는 뷰를 사용하여 렌더링된다. 6장은 JSP, 아파치 타일, Thymeleaf를 포함하여 스프링을 사용하는 다양한 뷰 기술을 설명한다.
- 7장에서는 스프링 MVC를 깊이 있게 파고든다. 구체적으로는 스프링 MVC 설정을 커스터마이즈(customize)하거나 파일 분할 업로드를 처리하는 방법, 컨트롤러에서 일어날 수 있는 예외를 처리하고 플래시 속성을 가진 요청들 사이에서 데이터를 전달하는 방법 등을 다룰 것이다.
- 8장에서는 스프링 웹 플로를 살펴본다. 스프링 웹 플로는 스프링 MVC의 확장판으로, 대화형 웹 애플리케이션의 개발을 가능하게 한다. 그리고 특정 흐름을 통해 사용자를 안내하는 웹 애플리케이션을 어떻게 구성하는지 알아본다.

- 9장에서는 스프링 시큐리티(Spring Security)를 이용해 애플리케이션의 웹 계층에 보안을 적용하는 방법을 알아본다.

3부는 애플리케이션의 프런트엔드에서 데이터가 어떻게 처리되고 유지되는지를 살펴본다.

- 10장에서는 관계형 데이터베이스에 저장된 데이터를 사용하기 위해 JDBC를 통한 스프링 추상화를 사용하는 데이터 영속성에 관한 내용을 다룬다.
- 11장에서는 관계형 데이터베이스에 데이터를 저장하기 위한 JPA(Java Persistence API)를 사용한 다른 각도의 데이터 영속성을 살펴본다.
- 12장에서는 MongoDB와 Neo4j 등 비관계형 데이터베이스가 스프링에 어떻게 사용되는지 살펴본다. 캐싱은 데이터의 저장 위치와 상관없이 데이터베이스에 필요 이상으로 접근하지 않음으로써 성능 향상에 도움이 될 수 있다.
- 13장에서는 스프링의 선언적 캐싱을 소개한다.
- 14장에서는 메소드 레벨에서 시큐리티를 적용하기 위해 AOP 사용 방법을 제공하는 스프링 시큐리티를 다시 설명한다.

마지막 4부에서는 다른 시스템과 스프링 애플리케이션의 통합 방법을 설명한다.

- 15장은 원격 서비스를 어떻게 만들고 사용하는지를 나타내며, RMI, Hessian과 Burlap, SOAP 기반의 서비스를 포함한다.
- 16장은 스프링 MVC로 다시 돌아와, 5장에서 이전에 나타낸 동일한 프로그래밍 모델을 사용하여 RESTful 서비스 제작 방법을 보여 준다.
- 17장은 스프링의 비동기 메시징을 지원하는 방법을 나타낸다. 또한, JMS(Java Message Service)와 AMQP(Advanced Message Queuing Protocol)의 사용 방법을 포함한다.
- 18장에서는 비동기 메시징을 다른 방법으로 나타내며, 서버와 클라이언트 간의 비동기 통신을 위해 웹 소켓과 STOMP를 가지고 스프링과 어떻게 사용하는지를 나타낸다.
- 19장은 스프링에서 이메일을 어떻게 보내는지를 나타낸다.
- 20장은 JMX(Java Management Extensions)를 위한 스프링의 관리 지원 방법을 나타내며, 스프링 애플리케이션을 위한 런타임 세팅을 수정하고 모니터링한다.
- 마지막으로 21장에서는 기존의 판도를 뒤집을 만큼 혁신적인 방법인 스프링 부트(Spring Boot)를 소개한다. 스프링 부트는 스프링 애플리케이션의 불필요한 설정들을 제거하고 필요한 기능에 대해서만 집중할 수 있도록 한다.

## 소스 코드에 적용된 규칙과 예제 다운로드

이 책에는 많은 코드 예제가 수록되어 있다. 코드 예제는 항상 너비가 일정한 코드용 폰트로 표기했다.

스프링의 클래스와 패키지 이름이 대체로 길다(하지만 표현력은 좋다). 이로 인해 코드의 한 줄을 한 줄로 표시하기 어려운 경우에는 ➥ 기호를 사용해 줄이 계속된다는 사실을 나타내었다.

이 책에 있는 코드 예제가 전체 코드가 아닐 수도 있다. 종종 특정한 주제에 집중하기 위해서 클래스 중 한두 개의 메소드만 나타내기도 했다. 이 책에서 다루는 애플리케이션의 전체 소스 코드는 출판사 웹사이트 www.manning.com/SpringinActionFourthEdition에서 다운로드할 수 있다.

## 저자와의 온라인 교류

이 책을 구매하면 매닝 출판사가 운영하는 사설 웹 포럼에 자유롭게 접근할 수 있다. 이 포럼에서는 책에 대한 의견을 공유할 수도 있고, 기술적인 질문을 올릴 수도 있으며, 저자 및 다른 사용자들로부터 도움을 얻을 수도 있다. 포럼을 이용하려면 www.manning.com/SpringinActionFourthEdition 링크를 방문해 보기 바란다. 저자와의 온라인 교류(Author Online) 페이지는 일단 회원 가입 후 포럼에 접근하는 방법과 이를 통해 받을 수 있는 지원, 그리고 포럼을 이용하기 위한 규칙 등을 설명하고 있다.

매닝 출판사가 이 책의 독자들에게 공약한 것은 독자들끼리, 그리고 독자와 저자가 의미 있는 대화를 나눌 수 있는 공간이다. 이 포럼은 저자들의 자발적인 (그리고 무보수) 참여로 운영되지만, 의무적으로 참여해야 하는 것은 아니다. 따라서 저자들의 관심을 끌 만한 도전적인 질문들을 올려 주기를 권한다.

이 포럼과 이 포럼에 쌓여 온 갖가지 논의들은 이 책이 계속 인쇄되는 동안 출판사의 웹사이트를 통해 지속적으로 접할 수 있다.

## 저자 소개

크레이그 월즈(Craig Walls)는 스프링 소셜과 스프링 싱크 프로젝트를 이끄는 피보탈(Pivotal)의 수석 엔지니어다. 스프링 프레임워크의 열렬한 지지자이자 프로모터로서 지역 사용자 그룹과 콘퍼런스에서 강연하고, 블로그에 스프링 관련 글을 쓰면서 바쁜 나날을 보내고 있다. 코딩을 하지 않을 때는 아내와 두 딸, 두 마리의 새, 두 마리의 강아지와 함께 지낸다.

## 책 표지에 대하여

《스프링 인 액션(제4판)》의 표지에 등장하는 인물은 요르단 남서부에 있는 케락(Karak) 지방의 원주민인 '르 카라코(Le Caraco)'다. 이 지방의 주도인 알카라크(Al-Karak)는 사해와 주위를 둘러싼 평원의 멋진 풍경을 감상할 수 있는, 언덕 위에 자리 잡은 옛 성을 자랑하는 도시다. 이 삽화는 1796년에 출판된 사브어(J. G. St. Saveur)의 프랑스 여행 서적 《여행 백과사전(Encyclopédie des Voyages)》에서 발췌한 것이다. 여행을 즐기는 것이 새로운 현상이었던 당시에는 이 책처럼 프랑스의 여러 지방과 외국의 원주민을 소개하는 여행 안내서가 유행했었다.

이 《여행 백과사전》에 있는 다양한 삽화들은 200여 년 전 세계 각지의 독특함과 개성을 분명하게 나타내고 있다. 이 시대는 수십 킬로미터 떨어진 두 지역에 사는 사람들의 복식이 명확히 구분되는 시대였다. 이 여행 안내 책자는 교류가 활발하지 않았던 과거 시대와 당시 사람들이 느꼈을 고립과 차이의 느낌을 생생하게 전해 준다. 그 시대 이후로는 복식이 변하면서 당시에는 매우 풍부했던 지역적 다양성이 차차 사라져 갔다. 우리가 살고 있는 시대는 복식만으로는 서로 다른 대륙에 살고 있는 사람들을 구분하기 어렵다. 긍정적으로 보자면, 복식의 다양성이 사라진 대신에 문화적, 시각적 다양성이 개인의 지적, 기술적인 생활에 더 많은 다양성과 흥미를 부여하는 시대다. 매닝 출판사는 컴퓨터 비즈니스의 독창성과 즐거움을 찬양하기 위해 이 여행 안내 책자의 그림들이 전해 주는 2세기 전의 풍부한 지역적 다양성을 책 표지에 싣고 있다.

# 베타리더 후기

**고승광(플랜티넷)**

스프링이 버전 업을 하면서 크기가 커지듯이 이 책도 점점 두꺼워지고 있는 것 같습니다. 'In Action' 책이 그렇듯이 스프링으로 프로젝트를 급하게 진행할 분들에게는 추천하기 힘든 책입니다. 다만, 스프링을 공부하실 분들에게는 강추합니다!

**김경범(네이버)**

스프링 개발을 막 시작했을 때를 되돌아보면 애너테이션과 설정 방법을 몰라 쩔쩔맸던 기억이 생생합니다. 이 책은 XML 설정법과 자바 설정법이 함께 소개되어 있고, 웹 이외에도 JPA, NoSQL, 스프링 보안과 최신 기술인 스프링 부트까지 포괄적으로 다루고 있습니다. 스프링 기술에 관해 공부하고자 한다면 이 책이 더없이 좋은 선택이 될 것입니다.

**김태경(카카오)**

국내에서 스프링은 웹 영역에서 큰 비중을 차지합니다. 하지만 웹은 스프링의 아주 일부분이며, 다른 많은 영역에서도 스프링은 존재하고 있습니다. 다양한 주제가 압축되어 있어 깊게 이해할 수는 없지만, 스프링의 다양한 모습과 사용에 관해 이해할 기회가 될 것입니다. 아마존 서평에서 말하는 만큼이나 좋은 책이었습니다. 일반적으로 생각하는 스프링뿐만 아니라 전체적인 포트폴리오를 소개함으로써 우물 안 개구리가 될 수 있는 스프링 프레임워크 유저들에게 '머스트 해브 아이템'이 되리라 봅니다.

### 장회수(프리랜서)

저에게 스프링을 이해하기 위한 단 한 권의 책을 추천해 달라고 하면 언제나 《Spring In Action》이었습니다. 이론과 배경에 충실하고 예제가 훌륭한 책이 스프링 4.1 버전에 맞춰 다시 나왔으니 스프링 입문자나 스프링 3 사용자에게 최고의 책이 될 것 같습니다. 특히, 도입 부분과 빈(Bean) 생명주기에 대한 이론 설명은 정독을 권합니다.

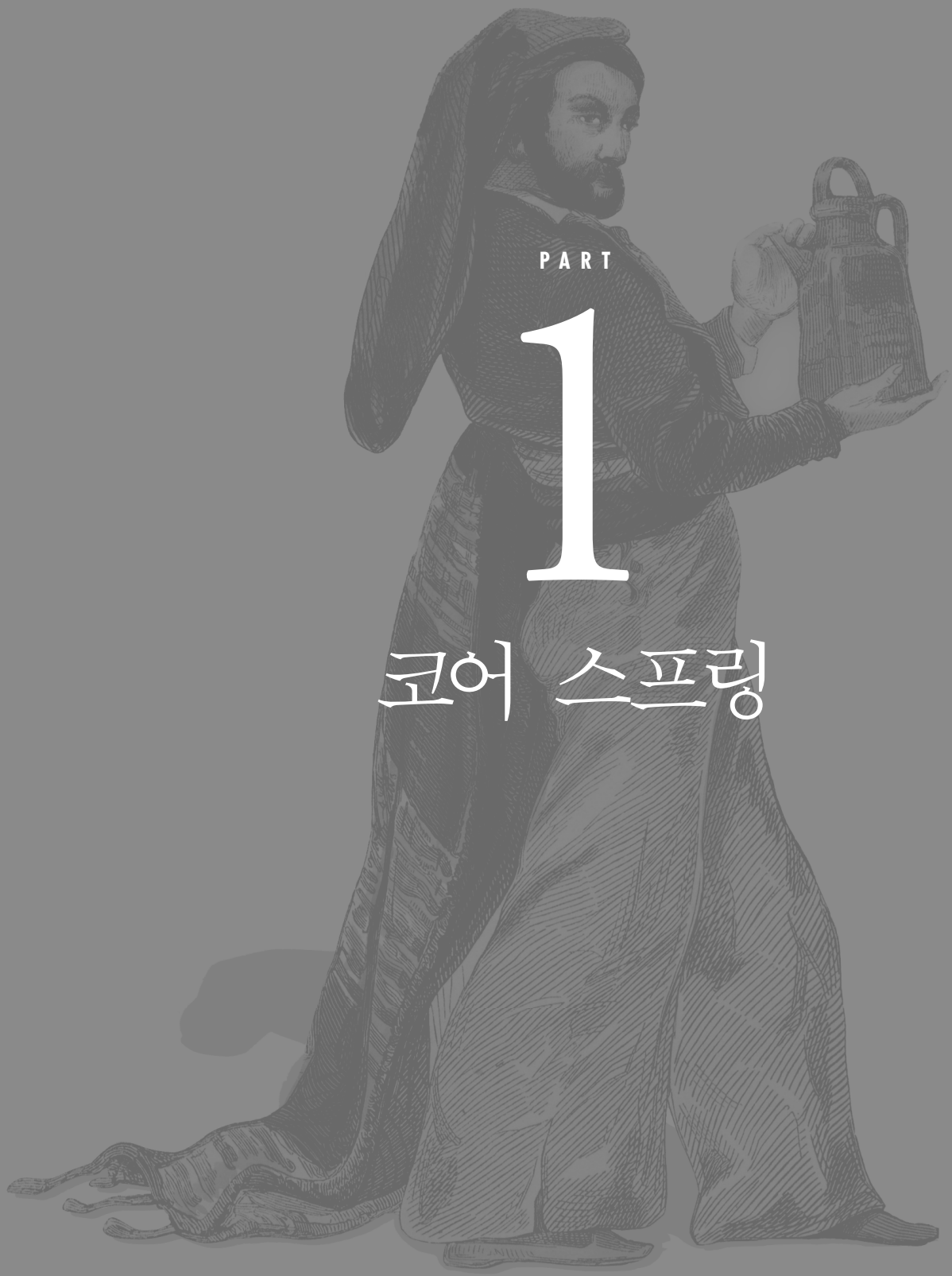
### 정원화(스프링코리아)

원서는 이미 전 세계적으로 인정받고 있는 책이라 내용 면에서 지적할 부분은 거의 없었습니다만, 4판 번역서를 베타리딩하면서 아쉬움 점은 번역에 대한 품질이었습니다. 출간 전에 베타리더의 지적을 반영하여 전체를 다시 읽어 보고 수정한다고 하니 다소 안심은 됩니다.

### 황인서(카카오)

실무에서 최대한 활용될 수 있는 예제들로 잘 구성된 《스프링 인 액션(제4판)》은 자바 진영의 가장 큰 에코시스템의 하나인 스프링 프레임워크를 기초부터 활용까지 잘 소개하고 있습니다. 입문자에게는 스프링 프레임워크의 다양한 컴포넌트에 관한 이해를 돕고, 경험 있는 중급자에게는 실무에서 조금 더 다양한 시각으로 스프링 프레임워크를 사용할 수 있도록 도와주는 훌륭한 입문서라 생각합니다. 아직 시중에는 스프링 4.x 기반의 책들이 많지 않은데, 예제와 실용성 있는 스프링 컴포넌트들을 기반으로 잘 소개한 책인 것 같습니다. 특히 JMS, AMQP 부분은 재미나게 보았습니다. WebSocket과 같은 부분도 최대한 실무에서 활용할 수 있도록 잘 소개한 것 같습니다. 스프링에 입문하는 초급 개발자 그리고 조금 더 깊이 있는 내용까지 알아야 할 중급 개발자에게 아주 적합한 입문서가 되리라 생각합니다.

제이펍은 책에 대한 애정과 기술에 대한 열정이 뜨거운 베타리더들로 하여금
출간되는 모든 서적에 사전 검증을 시행하고 있습니다.



PART

# 1

# 코어 스프링

스프링이 제공하는 기능은 매우 다양하지만, 핵심을 파고들면 스프링의 주요 기능은 종속객체 주입(DI, Dependency Injection, 의존성 주입이라고도 불림)과 애스펙트 지향 프로그래밍(AOP, Aspect-Oriented Programming, 관점지향 프로그래밍이라고도 불림)으로 귀결된다.

1장 "스프링 속으로"에서는 DI와 AOP의 기본적인 개념을 살펴보고, 두 기능이 애플리케이션 객체 간의 결합도를 줄이는 데 어떻게 기여하는지 알아본다.

2장 "빈 와이어링"에서는 애플리케이션 컴포넌트들에 대해서 더 깊이 살펴본다. 스프링에서 제공되는 자동 설정, 자바 기반 설정, XML 설정 옵션에 대해 알아본다.

3장 "고급 와이어링"에서는 오토와이어링(autowiring), 스코핑(scoping) 및 스프링 표현 언어(Spring Expression Language)를 취급하고, 조건적 설정 방법을 포함하여 스프링의 최대 효과를 이끌어 낼 수 있는 몇 가지 트릭과 테크닉을 소개한다.

4장 "애스펙트 지향 스프링"에서는 스프링의 AOP 기능을 이용해 시스템 전반에 걸친 서비스(보안이나 감사 등)와 그 서비스를 받는 객체 간의 결합도를 줄이는 방법을 살펴본다. 그리고 4장에서는 9장, 13장, 14장에서 다루는 서술적 보안과 캐싱(caching)을 위한 스프링 AOP 레버리지(leverage) 방법의 기초를 다룬다.

PART 1

# *Core Spring*

CHAPTER

# 1

# 스프링 속으로

**이 장에서 다룰 내용**

- 스프링의 빈 컨테이너
- 스프링의 코어 모듈 살펴보기
- 더 훌륭해진 스프링 에코시스템
- 스프링의 새로워진 점

자바 개발자가 되기에 좋은 시절이다.

거의 20년이라는 역사 속에 자바는 좋은 때도 있었고 나쁜 때도 있었다. 한무리의 거친 모래 폭풍과 같았던 애플릿, 엔터프라이즈 자바빈(EJB, Enterprise JavaBean), 자바 데이터 객체(JDO, Java Data Object), 무수한 로깅 프레임워크(countless longging framework)가 있었지만, 자바는 엔터프라이즈 소프트웨어를 구축하는 플랫폼으로서 풍부하고 다양한 역사를 즐겨 왔으며, 스프링은 이 스토리에서 주인공 역할을 해왔다.

초창기에 스프링은 EJB 같은 무거운 엔터프라이즈 자바 기술에 대한 대안으로 만들어졌다. 스프링이 제안한 것은 EJB에 비해 더 가벼운 프로그래밍 모델이었다. 그전에는 EJB나 다른 엔터프라이즈 자바 명세를 이용해야만 가능했던 일인데, 스프링 덕분에 일반 자바 객체(POJO, Plain Old Java Object)로도 가능해졌다.

시간이 지나면서 EJB와 자바 2 엔터프라이즈 에디션(J2EE, Java 2 Enterprise Edition)은 진화했고, EJB는 간단한 POJO 지향 프로그래밍 모델이라는, 자신만의 모델을 제공하기 시작했다. EJB는 종속객체 주입(DI)과 애스펙트 지향 프로그래밍(AOP)를 적용한다. 이는 스프링의 성공을 보며 최고의 영감을 받았던 부분이다.

비록 J2EE(지금은 JEE로 알려진)가 스프링을 따라잡을 수 있었다고는 해도, 스프링은 발전을 멈추지 않았다. 스프링은 어딘가에서 지금도 발전을 계속하고 있고, JEE는 이제 막 시작하거나 아직 시작조차 못한 부분도 있다. 모바일 개발, 소셜 API 통합, NoSQL 데이터베이스, 클라우드 컴퓨팅, 빅 데이터는 스프링이 발전되고 있는 몇몇 영역이다. 스프링의 미래는 밝다.

앞서 말했듯이 지금은 자바 개발자가 되기에 좋은 시절이다.

그리고 이제 스프링에 대해 많은 내용을 살펴볼 것이다. 이번 장에서는 높은 수준의 스프링을 경험하고, 스프링이 무엇에 관한 것인지에 대해 맛보기를 제공할 것이다. 이번 장은 스프링이 해결하는 문제 타입에 대한 좋은 아이디어를 줄 것이며, 이 책의 남은 부분을 위한 무대를 세팅할 것이다.

## 1.1 자바 개발 간소화

스프링(Spring)은 로드 존슨(Rod Johnson)의 책 《Expert One-on-One: J2EE Design and Development》(Wrox, 2002, http://amzn.com/0764543857)(한글판 《Expert One-on-One J2EE 설계와 개발》, 정보문화사)를 통해 소개한 오픈 소스 프레임워크로서 엔터프라이즈 애플리케이션 개발의 복잡함을 해소하기 위해 만들어졌다. 스프링은 EJB로만 할 수 있었던 작업을 평범한 자바빈을 사용해 가능케 한다. 하지만 스프링이 서버 측 개발에만 유용한 것은 아니다. 스프링을 사용하는 모든 자바 애플리케이션은 간소함, 테스트 용이성, 낮은 결합도라는 이득을 얻었다.

스프링이 애플리케이션 컴포넌트를 참조하면서 빈(bean)과 자바빈(JavaBean)을 사용하지만, 반드시 자바빈의 스펙을 따라야 한다는 것을 의미하지는 않는다. 스프링 컴포넌트는 POJO 타입을 가진다. 여기서는 자바빈을 느슨하게 정의하며, POJO와 동일하다고 간주한다.

이 책을 통해 스프링의 다양한 내용을 살펴보겠지만, 스프링이 제공하는 거의 모든 기본 사상은 몇 가지 기초적인 개념으로 귀결되며, 이는 스프링의 기본 임무인 '자바 개발 간소화'에 모든 초점을 맞춘다.

이것이 핵심 키워드다! 특정 기능을 간소화하는 프레임워크는 많이 존재한다. 하지만 스프링의 목적은 자바 개발을 폭넓게 간소화하는 데 있다. 이 내용에 대해서는 더 많은 설명이 필요하다. 그러면 어떻게 자바 개발을 간소화할 수 있을까?

자바 복잡도 간소화를 지원하기 위해 스프링은 네 가지 주요 전략을 사용한다.

- POJO를 이용한 가볍고(lightweight) 비침투적(non-invasive)인 개발[1]
- DI와 인터페이스 지향(interface orientation)을 통한 느슨한 결합도(loose coupling)
- 애스펙트와 공통 규약을 통한 선언적(declarative) 프로그래밍
- 애스펙트와 템플릿(template)을 통한 반복적인 코드 제거

스프링이 수행하는 거의 모든 작업은 이 네 가지 전략 중 하나 이상에 속한다. 앞으로 스프링이 자바 개발을 간소화하는 방법에 대한 구체적인 예제를 살펴보면서 이와 같은 사상을 확장해 나갈 것이다. 먼저 스프링이 POJO 지향 개발을 통해 어떻게 침투적 개발[2]을 최소화할 수 있는지부터 알아보자.

### 1.1.1 POJO의 힘

장기간의 자바 개발 경험이 있다면 인터페이스의 구현이나 클래스의 확장을 강요하는 프레임워크를 본 적이 있고, 심지어 이런 프레임워크로 작업한 경험도 있으리라 생각된다. 이런 침략적인 프로그래밍 모델의 쉬운 예는 EJB 2 시대의 무상태 세션 빈(stateless session bean)이다. 초창기 EJBs가 침투적 프로그래밍[3]의 쉬운 예라고는 하지만, 침투적 프로그래밍은 스트러츠(Struts), 웹워크(WebWork), 태피스트리(Tapestry) 및 수많은 다른 자바 명세와 프레임워크의 초기 버전에서 쉽게 발견됐다.

스프링은 API를 이용하여 애플리케이션 코드의 분산을 가능한 한 막는다. 스프링은 스프링에 특화된 인터페이스 구현이나 스프링 자체에 의존성이 높은 클래스 확장을 거의 요구하지 않는다. 스프링 기반 애플케이션의 클래스에는 스프링이 사용한다는 표시도 거의 없다. 최악의 경우, 클래스에 스프링의 애너테이션(annotation)이 붙지만 그렇지 않은 경우에는 POJO다.

---

1 역주 비침투적 개발이란, 바탕이 되는 기술을 사용하는 클래스, 인터페이스, API 등을 코드에 직접 나타내지 않는 방법이다. 복잡함을 분리할 수 있다.

2 역주 침투적 개발이란, 바탕이 되는 기술을 사용하는 클래스, 인터페이스, API 등을 코드에 나타내며 종속객체 주입을 명시적으로 고려하는 방법이다. 침투적이면 복잡함을 가중시킨다.

3 역주 침투적 프로그래밍은 침투적 개발 방법을 이용한 프로그래밍이다.

이해를 위해 아래의 HelloWorldBean 클래스를 살펴보자.

**코드 1.1 HelloWorldBean에 불필요한 요구(EJB를 사용할 때 반드시 구현해야 했던 요구)를 하지 않는 스프링**

```
package com.habuma.spring;
public class HelloWorldBean {
  public String sayHello() {   ← 이것이 필요한 전부
    return "Hello World";
  }
}
```

보다시피 이것은 간단하고 흔한 자바 클래스, 즉 POJO다. 스프링 컴포넌트라고 가리키는 것 외에 마땅한 특징은 없다. 스프링의 비침투적 프로그래밍[4] 모델에서, 이 클래스는 스프링 애플리케이션 외에서도 잘 동작한다.

간단한 형태에도 불구하고 POJO는 매우 강력하다. 스프링이 POJO에 힘을 불어넣는 방법 중 하나는 DI를 활용한 조립이다. 그럼, DI를 통해 애플리케이션 객체 상호 간의 결합도를 낮추는 방법을 알아보자.

## 1.1.2 종속객체 주입

"종속객체 주입(DI, Dependency Injection)"이라는 문구가 다소 위협적으로 들려서 복잡한 프로그래밍 기술의 개념이나 디자인 패턴이 떠오른다. 하지만 DI는 생각만큼 복잡하지 않다. 사실 프로젝트에 DI를 적용해 보면 코드가 훨씬 더 간단해지고, 이해하기 쉬우며, 테스트하기도 쉬워진다.

### DI 동작 방법

HelloWorld 예제보다 복잡한 실제 애플리케이션에서는 두 개 이상의 클래스가 서로 협력하여 비즈니스 로직(logic)을 수행한다. 이때 각 객체는 협력하는 객체에 대한 **레퍼런스**(reference, 즉 종속객체)를 얻을 책임이 있다. 그 결과, 결합도가 높아지고 테스트하기 힘든 코드가 만들어지기 쉽다.

예를 들어, 코드 1.2에 있는 Knight 클래스를 생각해 보자.

4 역주 비침투적 프로그래밍은 비침투적 개발 방법을 이용한 프로그래밍이다.

코드 1.2 **소녀를 구하는 기사(DamselRescuingKnight)는 소녀 구출 임무를 받은 자만 떠날 수 있다.**

```
package com.springinaction.knights;

public class DamselRescuingKnight implements Knight {
  private RescueDamselQuest quest;
  public DamselRescuingKnight() {
    this.quest = new RescueDamselQuest();  ← RescueDamselQuest에 강하게 결합
  }
  public void embarkOnQuest() {
    quest.embark();
  }
}
```

보다시피 DamselRescuingKnight는 생성자 안에 RescueDamselQuest를 생성한다. 이것은 DamselRescuingKnight가 RescueDamselQuest에 강하게 결합되도록 하며, 기사(knight)의 원정 출정 목록을 심각하게 제한한다. 도움이 필요한 여성이 있는 곳에 기사도 있다. 하지만 용을 물리쳐야 하거나 원탁이 필요하다면, 또한 순회하려면 그냥 앉아만 있어야 한다.

게다가 DamselRescuingKnight에 대한 단위 테스트를 작성하기도 몹시 어렵다. 단위 테스트에서는 기사의 embarkOnQuest() 메소드가 호출될 때 quest의 embark() 메소드 호출을 확인하고 싶다. 하지만 여기서는 이를 수행하는 명확한 방법이 없다. 안타깝게도 DamselRescuingKnight는 테스트하지 못한 채로 남아 있다.

결합도가 높은 코드는 한편으로 테스트와 재활용이 어렵고 이해하기도 어려우며, 오류를 하나 수정하면 다른 오류가 발생하는 경향이 있다. 반면에 전혀 결합이 없는 코드는 아무것도 할 수 없다. 무언가 쓸 만한 일을 하려면 클래스들끼리 어떻게든 서로 알고 있어야 한다. 결합은 필요하지만 주의해서 관리해야 한다.

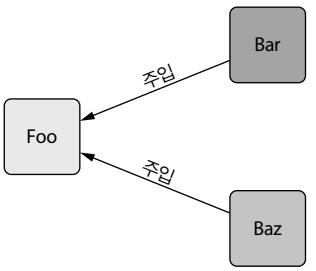



그림 1.1 **종속객체 주입은 객체가 스스로 종속객체를 획득하는 것과는 반대로 객체에 종속객체가 부여되는 것을 의미한다.**

DI를 이용하면 객체는 시스템에서 각 객체를 조율하는 제3자에 의해 생성 시점에 종속객체(dependency)가 부여된다. 그림 1.1에서 설명하듯이, 객체는 종속객체를 생성하거나 얻지 않는다. 즉, 종속객체는 종속객체가 필요한 객체에 주입된다.

이 의미를 설명하기 위해 코드 1.3에 있는 BraveKnight를 살펴보자. 이 클래스는 용감할 뿐만 아니라 발생할 어떤 종류의 원정도 떠난다.

코드 1.3 **부여된 어떤 Quest도 충분히 감당할 만큼 유연한 BraveKnight**

```
package com.springinaction.knights;

public class BraveKnight implements Knight {

  private Quest quest;

  public BraveKnight(Quest quest) { ← Quest 주입
    this.quest = quest;
  }

  public void embarkOnQuest() {
    quest.embark();
  }

}
```

보다시피 DamselRescuingKnight와 달리 BraveKnight는 자신의 원정(Quest)을 생성하지 않는다. 그 대신 생성 시점에 생성자 인자에 원정이 부여된다. 이와 같은 종류의 종속객체 주입은 **생성자 주입(constructor injection)**이다.

게다가 부여된 원정은 모든 원정 구현 인터페이스인 Quest 타입으로 제공된다. 따라서 BraveKnight는 RescueDamselQuest, SlayDragonQuest, MakeRoundTableRounderQuest, 또는 부여된 다른 Quest를 구현한다.

요점은 BraveKnight가 Quest의 특정 구현체에 결합되지 않는다는 사실이다. Quest 인터페이스를 구현하기만 하면 기사에게 어떤 종류의 원정을 떠나도록 요청하든 문제가 되지 않는다. 이것이 바로 DI의 주요 이점인 "느슨한 결합도(loose coupling)"이다. 어떤 객체가 자신이 필요로 하는 종속객체를 인터페이스를 통해서만 알고 있다면(구현 클래스나 인스턴스화 방법이 아니라), 사용하는 객체 쪽에서 아무런 변경 없이 종속객체를 다른 구현체로 바꾼다.

실제 종속객체를 바꾸는 가장 일반적인 방법 중 하나는 테스트하는 동안의 모의(mock) 구현체를 이용한 것이다. 강한 결합도(tight coupling)로 인해 DamselRescuingKnight를 적절히 테스트할 수 없었지만, 코드 1.4와 같이 Quest의 모의 구현체를 제공하여 BraveKnight를 쉽게 테스트한다.

코드 1.4 **BraveKnight 테스트를 위하여 모의 Quest 주입하기**

```
package com.springinaction.knights;
import static org.mockito.Mockito.*;
import org.junit.Test;

public class BraveKnightTest {
```

```
  @Test
  public void knightShouldEmbarkOnQuest() {
    Quest mockQuest = mock(Quest.class);  ← 모의 Quest 생성
    BraveKnight knight = new BraveKnight(mockQuest);  ← 모의 Quest 주입
    knight.embarkOnQuest();
    verify(mockQuest, times(1)).embark();
  }

}
```

여기서는 Quest 인터페이스의 모의 구현체를 만들기 위해 Mockito로 알려진 모의 객체 프레임워크를 사용했다. 모의 객체가 생겼으면 BraveKnight의 새로운 인스턴스(instance)를 생성하고, 생성자를 통해 모의 Quest를 주입한다. embarkOnQuest() 메소드를 호출한 후에는 Mockito에게 Quest의 embark() 메소드가 정확히 한 번 호출됐는지 확인한다.

### 기사에게 원정 임무 주입

이렇게 해서 BraveKnight 클래스는 모든 원정 임무를 부여받을 수 있게 됐다. 그러면 이제 BraveKnight 클래스에 어떤 Quest를 부여할지를 어떻게 지정할 수 있을까? 예를 들어, BraveKnight가 용을 물리치는 원정에 출정하길 원한다고 가정하자. 아마도 코드 1.5에 보이는 SlayDragonQuest가 적절할 것이다.

**코드 1.5 SlayDragonQuest는 BraveKnight에 주입될 Quest다.**

```
package com.springinaction.knights;

import java.io.PrintStream;

public class SlayDragonQuest implements Quest {

  private PrintStream stream;

  public SlayDragonQuest(PrintStream stream) {
    this.stream = stream;
  }

  public void embark() {
    stream.println("Embarking on quest to slay the dragon!");
  }

}
```

보다시피 SlayDragonQuest는 Quest 인터페이스를 구현하지만 BraveKnight에 적합하다. 또한, 대부분의 시작용 자바 샘플처럼 System.out.println( )에 의존하기보다는 SlayDragonQuest는 좀 더 일반적인 방법으로 생성자를 통해 PrintStream에 요청한다는 것을 눈치챘을지도 모르겠다. 여기서 중요한 질문은 바로 "어떻게 BraveKnight에게 SlayDragonQuest를 줄 수 있는가? 그리고 어떻게 SlayDragonQuest에게 PrintStream을 줄 수 있는가?" 하는 것이다.

애플리케이션 컴포넌트 간의 관계를 정하는 것을 **와이어링**(wiring)이라고 한다. 스프링에서 컴포넌트를 와이어링하는 방법은 여러 가지가 있지만 가장 일반적인 방법은 XML을 이용하는 방법이다. 코드 1.6은 BraveKnight, SlayDragonQuest, PrintStream을 함께 엮는 간단한 스프링 설정 파일인 knights.xml이다.

**코드 1.6 스프링의 XML 설정을 이용하여 BraveKnight에게 SlayDragonQuest 주입하기**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="knight" class="com.springinaction.knights.BraveKnight">
    <constructor-arg ref="quest" />  ← 원정 빈(bean) 주입
  </bean>

  <bean id="quest" class="com.springinaction.knights.SlayDragonQuest">  ↰
    <constructor-arg value="#{T(System).out}" />                        SlayDragon
  </bean>                                                               Quest 생성

</beans>
```

여기서 BraveKnight와 SlayDragonQuest가 스프링의 빈(bean)으로 선언되었다. BraveKnight 빈의 경우, 생성되면서 레퍼런스를 SlayDragonQuest 빈에게 넘겨주며 생성자의 인자가 된다. 반면, SlayDragonQuest 빈 선언은 스프링 표현 언어(Spring Expression Language)를 사용하여 System.out(이것은 PrintStream이다)을 SlayDragonQuesst의 생성자에게 넘긴다.

만약 XML 설정 방법이 마음에 들지 않으면, 스프링에서는 자바를 이용하여 설정할 수 있다. 그 예로 코드 1.7은 코드 1.6과 대등한 자바 기반 코드다.

코드 1.7 **XML 설정의 대안으로 자바 기반 설정을 제공하는 스프링**

```
package com.springinaction.knights.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import com.springinaction.knights.BraveKnight;
import com.springinaction.knights.Knight;
import com.springinaction.knights.Quest;
import com.springinaction.knights.SlayDragonQuest;

@Configuration
public class KnightConfig {

  @Bean
  public Knight knight() {
    return new BraveKnight(quest());
  }

  @Bean
  public Quest quest() {
    return new SlayDragonQuest(System.out);
  }

}
```

XML 기반의 설정이든 자바 기반의 설정이든 DI의 이점은 같다. 비록 BraveKnight가 Quest에 의존적이긴 하지만, 어떤 타입의 Quest가 주어질지 또는 그 Quest가 어디에서부터 올지는 모르는 것이다. 마찬가지로 SlayDragonQuest가 PrintStream에 의존적이긴 하지만, PrintStream이 어떻게 돌아가는지 알고서 코딩되는 것은 아니다. 오직 스프링만이 모든 조각이 어떻게 합쳐지는지 설정을 통해 아는 것이다. 이것을 통해, 종속된 클래스를 수정하지 않으면서도 종속성 수정이 가능하다.

이 예제는 스프링에서 빈을 와이어링하는 간단한 예를 보여 준다. 지금은 자세한 내용에 신경 쓰지 않아도 된다. 자세한 내용은 2장에서 스프링이 빈을 와이어링하는 또 다른 방법과 스프링이 자동으로 빈을 찾고 빈의 관계를 생성하는 방법을 살펴볼 예정이다.

BraveKnight와 Quest 사이의 관계를 정의했으니 XML 설정 파일을 로드하여 애플리케이션을 구동해 볼 차례다.

### 실행해 보기

스프링 애플리케이션에서 **애플리케이션 컨텍스트(application context)**는 빈에 관한 정의들을 바탕으로 빈들을 엮어 준다. 스프링 애플리케이션 컨텍스트는 애플리케이션을 구성하는 객체의 생성과 와이어링을 전적으로 책임진다. 스프링에는 애플리케이션의 여러 구현체가 존재하며, 각각의 주요 차이점은 오직 설정을 로드하는 방법에 있다.

knights.xml에서 빈들이 XML 파일에 선언되어 있을 때, 애플리케이션 컨텍스트로 ClassPathXmlApplicationContext()를 사용하면 좋다(자바 기반 설정에서 스프링은 AnnotationConfigApplicationContext을 제공한다). 이 스프링 컨텍스트 구현체는 애플리케이션의 클래스패스에 있는 하나 이상의 XML 파일에서 스프링 컨텍스트를 로드한다. 코드 1.8에 있는 main() 메소드는 ClassPathXmlApplicationContext를 사용해 knights.xml을 로드하고 Knight 객체에 대한 레퍼런스를 얻는다.

**코드 1.8 KnightMain.java는 기사를 포함하는 스프링 컨텍스트를 로드한다.**

```
package com.springinaction.knights;

import org.springframework.context.support.
                  ClassPathXmlApplicationContext;

public class KnightMain {

  public static void main(String[] args) throws Exception {
    ClassPathXmlApplicationContext context =  ← 스프링 컨텍스트 로드
        new ClassPathXmlApplicationContext(
            "META-INF/spring/knights.xml");
    Knight knight = context.getBean(Knight.class);  ← 기사 빈 얻기
    knight.embarkOnQuest();  ← 기사 사용
      context.close();
  }

}
```

여기서 main() 메소드는 knights.xml 파일을 기반으로 스프링 애플리케이션 컨텍스트를 생성한다. 그런 다음, Knight.class의 구현체를 조회하기 위해 팩토리로 애플리케이션 컨텍스트를 사용한다. Knight 객체에 대한 레퍼런스를 얻은 후에 간단히 embarkOnQuest() 메소드를 호출해 기사를 주어진 원정의 길로 떠나보낸다. 이 클래스는 기사가 어떤 유형의 Quest를 떠나는지에 대해서는 아무것도 알지 못한다. BraveKnight를 다룬다는 사실도 알지 못한다. knights.xml 파일만이 어떤 구현체인지 확실히 알고 있다.

지금까지 DI를 간단히 살펴보았다. 앞으로도 이 책을 통해 DI에 대한 더 많은 내용을 알아보겠다. 하지만 DI를 더 자세히 알아보고 싶다면 DI를 상세히 다루는 단지 R. 프라사나(Dhanji R. Prasanna)의 《Dependency Injection》(Manning, 2009, www.manning.com/prasanna/)을 읽어 보기 바란다.

이제는 또 다른 스프링의 자바 간소화 전략인 "애스펙트(Aspect)를 통한 선언적 프로그래밍"을 알아보자.

### 1.1.3 애스펙트 적용

DI가 소프트웨어 컴포넌트의 결합도를 낮춰 준다면, 애스펙트 지향 프로그래밍은 애플리케이션 전체에 걸쳐 사용되는 기능을 재사용할 수 있는 컴포넌트에 담는다.

애스펙트 지향 프로그래밍은 소프트웨어 시스템 내부의 관심사들을 서로 분리(separation of concerns)하는 기술이라고 설명한다. 시스템은 보통 특정한 기능을 책임지는 여러 개의 컴포넌트로 구성된다. 그러나 각 컴포넌트는 대체로 본연의 특정한 기능 외에 로깅(logging)이나 트랜잭션 관리, 보안 등의 시스템 서비스도 수행해야 하는 경우가 많다. 이러한 시스템 서비스는 시스템의 여러 컴포넌트에 관련되는 경향이 있으므로 **횡단 관심사(cross-cutting concerns)**라고 한다.

이러한 관심사가 여러 컴포넌트에 퍼지게 되면 코드는 다음 두 가지 차원에서 복잡해진다.

- 시스템 전반에 걸친 관심사를 구현하는 코드가 여러 컴포넌트에서 중복되어 나타난다. 이때 문제는 이 관심사의 구현을 변경해야 하는 경우 여러 컴포넌트를 모두 변경해야 한다는 점이다. 이 관심사를 별도의 모듈로 추상화해서 각 컴포넌트에서 하나의 메소드만 호출할 수 있도록 만든다고 하더라도, 여전히 이 메소드가 여러 컴포넌트에서 중복되어 나타나는 문제는 동일하다.
- 컴포넌트의 코드가 본연의 기능과 관련 없는 코드로 인해 지저분해진다. 주소록에 주소를 등록하는 메소드는 보안 상태가 유지됐는지 아닌지에는 신경 쓸 필요 없이 주소를 등록하는 방법에만 관여하는 것이 좋다.

그림 1.2는 위에 설명한 복잡성을 그림으로 나타낸 것이다. 왼쪽에 있는 비즈니스 객체들은 시스템 서비스와 매우 긴밀하게 관련을 맺고 있다. 각 객체는 본연의 책임을 수행할 뿐만 아니라 로깅과 보안, 트랜잭션 컨텍스트에 대해서도 파악해야 한다.

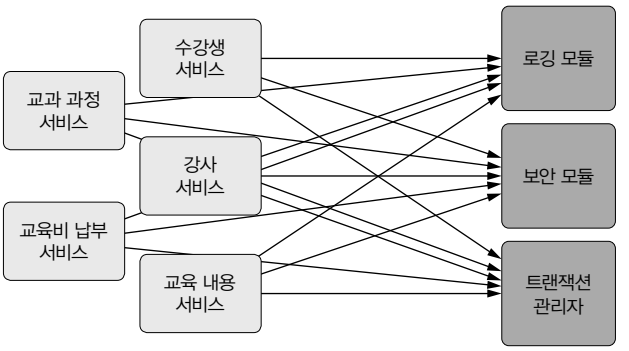



**그림 1.2 모듈의 주 관심사에 대한 호출보다는 로깅이나 보안 등 시스템 전반에 걸친 관심사에 대한 호출이 빈번하다.**

AOP는 시스템 서비스를 모듈화해서 컴포넌트에 선언적으로 적용한다. AOP를 이용하면 시스템 서비스에 대해서는 전혀 알지 못하지만, 응집도가 높고 본연의 관심사에 집중하는 컴포넌트를 만든다. 다시 말해, 애스펙트는 확실히 POJO를 단순화한다.

그림 1.3에서 묘사하고 있는 것처럼 애스펙트를 애플리케이션의 여러 컴포넌트를 덮는 담요처럼 생각하면 개념을 잡는 데 도움된다. 이 그림에서 애플리케이션의 핵심은 비즈니스 기능을 구현하는 모듈들로 구성되어 있고, AOP를 이용해서 이 핵심 애플리케이션 위에 추가적인 기능을 여러 겹으로 덮고 있다. 이때 핵심 기능을 구현하는 모듈에는 아무런 변화도 가하지 않고 추가적인 기능을 선언적으로 적용한다. 이 개념은 매우 강력한 개념으로서 보안, 트랜잭션, 로깅 등을 처리하고, 애플리케이션의 핵심 비즈니스 로직이 지저분해지는 것을 막는다.

기사 예제로 다시 돌아가, 기본적인 스프링 애스펙트를 추가해 보면서 스프링에서 애스펙트를 적용하는 방법을 구체적으로 살펴보자.

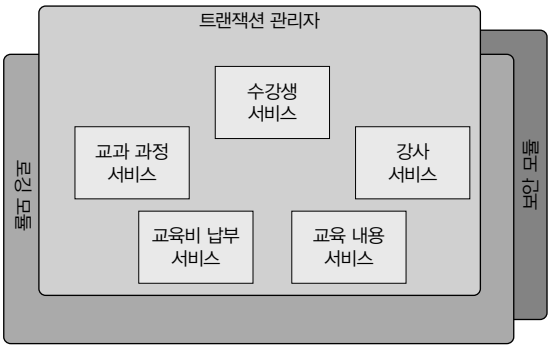



**그림 1.3 AOP를 이용하면 시스템 전반에 걸친 관심사가 관련 컴포넌트를 감싼다. 이렇게 함으로써 애플리케이션의 컴포넌트들이 본연의 비즈니스 기능에 집중한다.**

### AOP 실습

음유시인으로 알려지고 음악을 좋아하는 이야기꾼에 의해 기사의 업적이 노래로 기록되므로 음유시인의 서비스를 이용하여 BraveKnight의 출정과 복귀를 기록하고 싶다고 가정해 보자. 사용할 Minstrel 클래스는 코드 1.9와 같다.

**코드 1.9 Minstrel은 중세 시대의 음악을 좋아하는 로깅 시스템**

```
package com.springinaction.knights;

import java.io.PrintStream;

public class Minstrel {

  private PrintStream stream;

  public Minstrel(PrintStream stream) {
    this.stream = stream;
  }
  public void singBeforeQuest() {  ← 원정 전에 호출됨
    stream.println("Fa la la, the knight is so brave!");
  }

  public void singAfterQuest() {  ← 원정 후에 호출됨
    stream.println("Tee hee hee, the brave knight " +
        "did embark on a quest!");
  }

}
```

보다시피 Minstrel은 두 개의 메소드가 있는 간단한 클래스다. singBeforeQuest() 메소드는 기사가 원정을 떠나기 전에 호출되며, singAfterQuest() 메소드는 기사가 원정을 마친 후에 호출된다. 두 가지 경우 모두에서 Minstrel은 생성자를 통해 주입된 PrintStream을 거쳐 기사의 행동을 노래한다.

코드에서 작업하기 위해서는 간단해야 한다. BraveKnight에다 주입만 하면 된다. 안 그런가? Minstrel을 사용하기 위해서 BraveKnight를 적절히 수정하자. 코드 1.10은 첫 번째 시도로, BraveKnight와 Minstrel을 합치는 것을 보여 준다.

코드 1.10 **Minstrel 메소드를 호출해야 하는 BraveKnight**

```
package com.springinaction.knights;

public class BraveKnight implements Knight {

  private Quest quest;
  private Minstrel minstrel;

  public BraveKnight(Quest quest, Minstrel minstrel) {
    this.quest = quest;
    this.minstrel = minstrel;
  }

  public void embarkOnQuest() throws QuestException {
    minstrel.singBeforeQuest();  ← 기사가 자신의 음유시인을 관리해야 할까?
    quest.embark();
    minstrel.singAfterQuest();
  }

}
```

기교를 부려 봤다. 이제 필요한 것은 스프링 설정으로 돌아가서 Minstrel 빈을 선언하고 그것을 BraveKnight 빈의 생성자에 주입하는 것이다.

그렇지만 일부는 적절해 보이지 않는다. 정말로 기사의 관심 범위 내에서 음유시인을 관리해야 할까? 음유시인은 요청 없이도 그의 일을 수행한다. 궁극적으로 음유시인의 일은 기사의 업적에 대해 노래하는 것이다. 왜 기사가 음유시인을 상기시켜야 하는가?

게다가 기사는 음유시인에 대해 알아야 하므로 강제로 Minstrel을 BraveKnight에 주입(inject)한다. 이는 BraveKnight의 코드를 복잡하게 만들 뿐만 아니라 음유시인이 없는 기사를 원하는 경우 당황스러워진다. 만일 Minstrel이 null이라면? 이와 같은 상황을 다루기 위해 몇 가지 null 체크 로직을 도입해야 할까?

간단한 BraveKnight 클래스가 점차 복잡해지기 시작하고 nullMinstrel 시나리오까지 처리해야 한다면 그 복잡함은 더해진다. 하지만 AOP를 사용하면 기사의 원정에 대해 노래할 음유시인을 선언할 수 있으며, 기사는 Minstrel 메소드를 직접 처리하는 일에서 해방된다.

Minstrel을 애스펙트로 바꾸려면 스프링 설정 파일의 하나로 선언하기만 하면 된다. 여기서는 knights.xml 파일을 수정하여 Minstrel을 애스펙트로 선언했다.

코드 1.11 **Minstrel을 애스펙트로 선언하기**

```
<?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:aop="http://www.springframework.org/schema/aop"
  xsi:schemaLocation="http://www.springframework.org/schema/aop
      http://www.springframework.org/schema/aop/spring-aop-3.2.xsd
    http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="knight" class="com.springinaction.knights.BraveKnight">
    <constructor-arg ref="quest" />
  </bean>

  <bean id="quest" class="com.springinaction.knights.SlayDragonQuest">
    <constructor-arg value="#{T(System).out}" />
  </bean>

  <bean id="minstrel" class="com.springinaction.knights.Minstrel">   ← Minstrel 빈 선언
    <constructor-arg value="#{T(System).out}" />
  </bean>

  <aop:config>
    <aop:aspect ref="minstrel">
      <aop:pointcut id="embark"
          expression="execution(* *.embarkOnQuest(..))"/>   ← 포인트컷 정의

      <aop:before pointcut-ref="embark"   ← 비포 어드바이스 선언
          method="singBeforeQuest"/>

      <aop:after pointcut-ref="embark"   ← 애프터 어드바이스 선언
          method="singAfterQuest"/>
    </aop:aspect>
  </aop:config>

</beans>
```

여기서 스프링의 aop 설정 네임스페이스(namespace)를 사용하여 Minstrel 빈이 애스펙트라고 선언한다. 먼저, Minstrel을 빈으로 선언한다. 그런 다음, <aop:aspect> 요소에서 빈을 참조한다. 애스펙트를 더 정의해 보자면, embarkOnQuest() 메소드가 실행되기 전에 Minstrel의 singBeforeQuest() 메소드가 호출되어야 한다고 선언한다(<aop:before>를 이용). 이것을 **비포 어드바이스(before advice)**라 부른다. 그리고 embarkOnQuest() 메소드가 실행된 후에 singAfterQuest() 메소드가 호출되어야 한다고 선언한다(<aop:after>를 이용). 이것을 **애프터 어드바이스(after advice)**라 부른다.

양쪽 경우 모두 pointcut-ref 애트리뷰트는 embark라는 이름의 포인트커트(pointcut)를 참조한다. 이 포인트커트는 앞에 있는 <pointcut> 요소에 어드바이스가 적용될 위치를 선택하는 expression 애트리뷰트와 함께 정의되어 있다. expression 구문은 AspectJ의 포인트커트 표현식 언어다.

AspectJ나 AspectJ 포인트커트 표현식의 작성 방법을 자세히 모른다고 해도 걱정할 필요는 없다. 4장에서 스프링의 AOP에 대해 더 자세히 논의한다. 지금은 스프링이 BraveKnight가 원정을 떠나기 전과 후에 Minstrel의 singBeforeQuest( )와 singAfterQuest( ) 메소드를 호출한다는 정도만 알아도 충분하다.

이렇게 해서 몇 줄 안 되는 XML 설정으로 Minstrel을 스프링 애스펙트로 바꿨다. 지금 완전히 이해가 되지 않더라도 걱정할 필요는 없다. 4장에서 다양한 스프링 AOP 예제를 통해 충분히 이해할 수 있는 시간을 갖게 된다. 지금은 이 예제에서 다음에 설명하는 두 가지만 분명하게 알고 넘어가면 된다.

첫 번째는 Minstrel이 여전히 POJO라는 점이다. Minstrel이 애스펙트로 사용될 것임을 나타내는 내용은 Minstrel에 전혀 포함되어 있지 않다. 스프링 컨텍스트에서 선언적으로 애스펙트가 된다.

두 번째이면서 가장 중요한 점은 BraveKnight가 Minstrel을 명시적으로 호출하지 않아도 Minstrel이 BraveKnight에 적용된다는 사실이다. 실제로 BraveKnight는 Minstrel의 존재를 전혀 인식하지 못한다.

또한 중요한 것은 Minstrel을 애스펙트로 바꾸기 위해 몇 가지 스프링 마법을 사용했지만, 먼저 <bean>으로 선언되어야 한다는 사실이다. 여기서의 핵심은 종속객체 주입 같은 다른 스프링 빈과 함께 수행할 수 있는 스프링 애스펙트를 이용하면 무엇이든 가능하다는 점이다.

기사에 대해 노래하기 위해 애스펙트를 사용하는 일도 즐거울 수 있지만, 스프링의 AOP는 더 실용적인 목적으로 사용된다. 나중에 보겠지만 스프링 AOP는 선언적 트랜잭션과 보안 등의 서비스를 제공하기 위해 도입한다(9장과 14장).

하지만 지금은 스프링이 자바 개발을 간소화시켜 주는 또 하나의 방법을 알아보자.

### 1.1.4 템플릿을 이용한 상투적인 코드 제거

코드를 작성하다가 이전에 이미 작성했던 코드 같다고 느껴 본 적이 있는가? 이것은 데자뷰(déjà vu)가 아니다. 이것이 바로 **상투적인 코드**(boilerplate code)다. 즉, 공통 작업이나 간단한 작업을 위해 반복적으로 작성해야 하는 코드다.

안타깝게도 자바 API에는 상투적인 코드가 많이 포함되어 있다. 상투적 코드의 대표적인 예는 데이터베이스에서 데이터를 조회하는 JDBC(Java Database Connectivity) 작업이다. 이전에 JDBC로 작업한 적이 있다면 다음에 또 유사한 코드를 작성해야 하는 경우가 있다.

**코드 1.12 JDBC 등 상당수의 자바 API에는 상투적인 코드 작성이 포함되어 있다.**

```
public Employee getEmployeeById(long id) {
  Connection conn = null;
  PreparedStatement stmt = null;
  ResultSet rs = null;
  try {
    conn = dataSource.getConnection();
    stmt = conn.prepareStatement(
          "select id, firstname, lastname, salary from " +
          "employee where id=?");  ← 직원 조회
    stmt.setLong(1, id);
    rs = stmt.executeQuery();
    Employee employee = null;
    if (rs.next()) {
      employee = new Employee();  ← 데이터로부터 객체 생성
      employee.setId(rs.getLong("id"));
      employee.setFirstName(rs.getString("firstname"));
      employee.setLastName(rs.getString("lastname"));
      employee.setSalary(rs.getBigDecimal("salary"));
    }
    return employee;
  } catch (SQLException e) {  ← 여기서 무엇을 수행해야 하나?
  } finally {
      if(rs != null) {  ← 정리 작업
        try {
          rs.close();
        } catch(SQLException e) {}
      }
      if(stmt != null) {
        try {
        stmt.close();
        } catch(SQLException e) {}
      }
      if(conn != null) {
        try {
          conn.close();
        } catch(SQLException e) {}
      }
  }
  return null;
}
```

보다시피 JDBC 코드는 직원의 이름과 급여를 데이터베이스에서 조회한다. 하지만 조회하는 부분을 찾으려면 열심히 살펴봐야 한다. 몇 줄 안 되는 직원 조회 코드가 JDBC 형식의 더미에 묻혀 버렸기 때문이다. 먼저 커넥션(connection)을 생성한 후에 질의객체(statement)를 생성하며, 최종적으로 결과를 조회한다. 그리고 예외를 던졌을(throw) 때 수행할 수 있는 작업이 많지 않아도 JDBC의 분노를 달래기 위해 검사형 예외(checked exception)인 SQLException이 발생한다.

마지막으로, 모든 작업을 완료한 후에는 커넥션과 질의객체 그리고 결과 집합을 닫는 정리 작업을 수행해야 한다. 이 또한 분노를 일으킨다. 따라서 여기서도 별도로 SQLException을 잡아야 한다.

코드 1.11에서 가장 주목할 부분은 거의 모든 JDBC 작업을 위해 작성했던 코드와 완벽히 동일하다는 사실이다. 직원 조회를 위해 수행하는 작업은 극히 일부분이다. 대부분은 JDBC의 상투적인 코드다.

상투적인 코드 작업에 JDBC만 있는 것은 아니다. 많은 기능은 종종 유사한 상투적인 코드를 요구한다. JMS, JNDI 그리고 REST 서비스의 소비에는 수많은 공통적인 반복 코드가 포함되어 있다.

스프링은 템플릿에 상투적인 코드를 캡슐화하여 반복적인 코드를 제거하는 방법을 찾는다. 스프링의 JdbcTemplate은 전통적인 JDBC에서 필요한 모든 형식 없이도 데이터베이스 작업을 수행할 수 있게 한다.

예를 들어, 스프링의 JdbcTemplate을 사용하여 JDBC API가 요구하는 작업이 아닌 직원 데이터를 조회하는 작업에 초점을 맞추도록 getEmployeeById( ) 메소드를 재작성한다. 업데이트된 getEmployeeById( ) 메소드는 코드 1.13과 같다.

**코드 1.13 템플릿은 코드가 작업에 바로 집중할 수 있게 한다.**

```
public Employee getEmployeeById(long id) {
  return jdbcTemplate.queryForObject(
          "select id, firstname, lastname, salary " +  ← SQL 쿼리
          "from employee where id=?",
          new RowMapper<Employee>() {
            public Employee mapRow(ResultSet rs,
                    int rowNum) throws SQLException {  ← 결과를 객체에 매핑
              Employee employee = new Employee();
              employee.setId(rs.getLong("id"));
              employee.setFirstName(rs.getString("firstname"));
              employee.setLastName(rs.getString("lastname"));
              employee.setSalary(rs.getBigDecimal("salary"));
```

```
            return employee;
          }
        },
        id);   ← 쿼리 파라미터 지정
}
```

보다시피 새롭게 수정된 getEmployeeById()는 매우 간단하면서도 실제로 데이터베이스에서 직원을 조회하는 작업에 초점을 맞춘다. 템플릿의 queryForObject() 메소드에는 SQL 쿼리(query), RowMapper(결과 집합 데이터를 도메인 객체에 매핑) 그리고 쿼리 파라미터가 부여된다. getEmployeeById()에서는 이전에 보였던 JDBC의 상투적인 코드가 보이지 않는다. 모든 것은 템플릿 내부에서 처리된다.

지금까지 POJO 지향 개발, DI, 애스펙트 그리고 템플릿을 이용하여 자바 개발의 복잡성을 공격하는 방법을 살펴보았다. 또한, XML 기반 설정 파일에서 빈과 애스펙트를 설정하는 방법도 살펴보았다. 그런데 이러한 파일을 어떻게 로드할 수 있을까? 이제부터 애플리케이션의 빈이 위치하는 스프링 컨테이너를 살펴보자.

## 1.2 빈을 담는 그릇, 컨테이너

스프링 기반 애플리케이션에서는 스프링 컨테이너(container) 안에서 객체가 태어나고, 자라고, 소멸한다. 그림 1.4와 같이 스프링 컨테이너는 객체를 생성하고, 서로 엮어 주고, 이들의 전체 생명주기(lifecycle: 생성에서 소멸까지)를 관리한다.

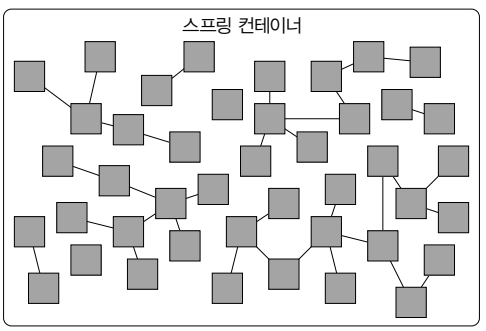



**그림 1.4 스프링 애플리케이션에서 객체는 스프링 컨테이너 내에서 태어나고 더불어 살아간다.**

스프링 컨테이너가 어떤 객체를 생성하고 구성해서 서로 엮어 줘야 하는지 알 수 있게 설정하는 방법은 다음 장에서 설명하겠다. 우선은 객체들의 삶의 터전인 스프링 컨테이너 자체부터

이해하는 것이 더 중요하다. 스프링 컨테이너에 대한 이해를 토대로 객체가 어떻게 관리되는지를 이해해야 한다.

스프링 컨테이너는 스프링 프레임워크의 핵심부에 위치한다. 스프링 컨테이너는 종속객체 주입을 이용해서 애플리케이션을 구성하는 컴포넌트를 관리하며, 협력 컴포넌트 간 연관관계의 형성도 여기에서 이뤄진다. 이러한 짐을 컨테이너에 덜어 버린 객체들은 더 명확하고, 이해하기 쉬우며, 재사용을 촉진하고, 단위 테스트가 용이해진다.

스프링 컨테이너는 여러 가지가 있다. 스프링에는 여러 컨테이너 구현체가 존재하며, 이들은 크게 두 가지로 분류된다. 첫 번째 부류는 **빈 팩토리**(org.springframework.beans.factory.BeanFactory 인터페이스에 의해 정의)로, 이는 DI에 대한 기본적인 지원을 제공하는 가장 단순한 컨테이너다. 두 번째 부류는 **애플리케이션 컨텍스트**(org.springframework.context.ApplicationContext 인터페이스에 의해 정의)로, 빈 팩토리를 확장해 프로퍼티(property) 파일에 텍스트 메시지를 읽고 해당 이벤트 리스너(listener)에 대한 애플리케이션 이벤트 발행 같은 애플리케이션 프레임워크 서비스를 제공하는 컨테이너다.

스프링으로 작업할 때 빈 팩토리나 애플리케이션 컨텍스트 중에 아무거나 사용해도 상관없지만, 빈 팩토리는 대부분의 애플리케이션에 대하여 지나치게 저수준의 기능을 제공한다. 따라서 빈 팩토리보다는 애플리케이션 컨텍스트를 더 선호한다. 따라서 앞으로는 애플리케이션 컨텍스트에 초점을 맞춰 작업할 예정이며, 빈 팩토리에 대해서는 많은 시간을 할애하지 않겠다.

### 1.2.1 또 하나의 컨테이너, 애플리케이션 컨텍스트

스프링에는 다양한 종류의 애플리케이션 컨텍스트가 있다. 그중에서 가장 많이 접하게 될 몇 가지는 다음과 같다.

- AnnotationConfigApplicationContext — 하나 이상의 자바 기반 설정 클래스에서 스프링 애플리케이션 컨텍스트를 로드한다.
- AnnotationConfigWebApplicationContext — 하나 이상의 자바 기반 설정 클래스에서 스프링 웹 애플리케이션 컨텍스트를 로드한다.
- ClassPathXmlApplicationContext — 클래스패스(classpath)에 위치한 XML 파일에서 컨텍스트 정의 내용을 로드한다.
- FileSystemXmlApplicationContext — 파일 시스템에서, 즉 파일 경로로 지정된 XML 파일에서 컨텍스트 정의 내용을 로드한다.

- XmlWebApplicationContext — 웹 애플리케이션에 포함된 XML 파일에서 컨텍스트 정의 내용을 로드한다.

AnnotationConfigWebApplicationContext와 XmlWebApplicationContext는 8장에서 웹 기반 스프링 애플리케이션을 설명할 때 살펴보기로 한다. 지금은 간단히 FileSystemXmlApplicationContext를 이용해서 파일 시스템에서 애플리케이션 컨텍스트를 로드하거나, ClassPathXmlApplicationContext를 이용해서 클래스패스에서 애플리케이션 컨텍스트를 로드하자.

파일 시스템이나 클래스패스에서 애플리케이션 컨텍스트를 로드하는 방법은 빈 팩토리 때와 유사하다. 다음은 FileSystemXmlApplicationContext를 로드하는 코드다.

```
ApplicationContext context = new
        FileSystemXmlApplicationContext("c:/knight.xml");
```

클래스패스에서 로드하는 ClassPathXmlApplicationContext도 다를 바 없다.

```
ApplicationContext context = new
        ClassPathXmlApplicationContext("knight.xml");
```

둘 사이에 차이가 있다면 FileSystemXmlApplicationContext의 경우 knight.xml을 파일 시스템에서 지정된 경로(c:/knight.xml)로 찾고, ClassPathXmlApplicationContext는 클래스패스에 포함된 모든 경로(JAR 파일 포함)에서 찾으려 한다는 점이다.

그렇지 않고 자바 설정에서 애플리케이션 컨텍스트를 로드하려고 한다면, AnnotationConfigApplicationContext를 사용한다.

```
ApplicationContext context = new AnnotationConfigApplicationContext(
    com.springinaction.knights.config.KnightConfig.class);
```

스프링 애플리케이션 컨텍스트를 로드하기 위한 XML 파일을 명시하는 대신, 빈을 로드하기 위한 설정 클래스를 AnnotationConfigApplicationContext가 받았다.

애플리케이션 컨텍스트를 얻은 다음, 컨텍스트의 getBean() 메소드를 호출하여 스프링 컨테이너에서 빈을 조회한다.

이제 스프링 컨테이너를 생성하는 기초 방법을 익혔으니 컨테이너 내부 빈의 생명주기를 좀 더 살펴보자.

### 1.2.2 빈의 일생

보통의 자바 애플리케이션에서 빈의 생명주기는 매우 단순하다. 자바의 new 키워드를 이용해 빈을 인스턴스화하고 이를 바로 사용한다. 빈이 더 이상 사용되지 않으면 가비지 컬렉션(garbage collection) 후보가 되어 언젠가는 메모리 덩어리가 됐다가 허공 속으로 사라질 것이다.

반면에 스프링 컨테이너 내에서 빈의 생명주기는 좀 더 정교하다. 빈이 생성될 때 스프링이 제공하는 커스터마이징(customizing) 기회를 이용하려면 스프링의 빈 생명주기를 이해해야 한다. 그림 1.5는 BeanFactory 컨테이너 내에서 빈이 갖는 구동(startup) 생명주기를 보여 준다.

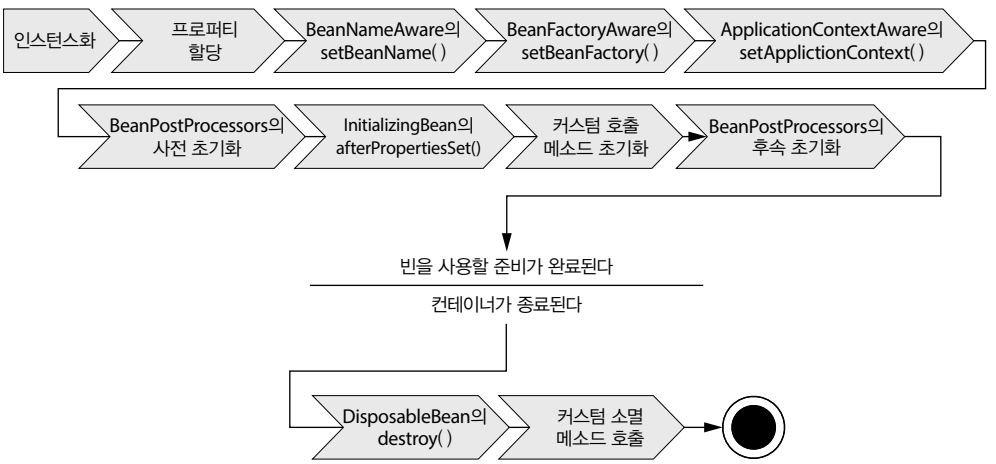



**그림 1.5 빈은 스프링 컨테이너에서 탄생과 소멸 사이에 몇 단계의 과정을 거치게 된다. 각 단계는 스프링이 빈을 관리하는 방법을 제어할 수 있는 기회이기도 하다.**

보다시피 빈 팩토리는 빈이 사용 가능한 상태가 되기 전에 몇 가지 준비(setup) 과정을 수행한다. 그림 1.5의 각 과정을 상세히 설명하면 다음과 같다.

1. 스프링이 빈을 인스턴스화한다.
2. 스프링이 값과 빈의 레퍼런스를 빈의 프로퍼티에 주입한다.
3. 빈이 BeanNameAware를 구현하면 스프링이 빈의 ID를 setBeanName( ) 메소드에 넘긴다.
4. 빈이 BeanFactoryAware를 구현하면 setBeanFactory( ) 메소드를 호출하여 빈 팩토리 자체를 넘긴다.
5. 빈이 ApplicationContextAware를 구현하면 스프링이 setApplicationContext( ) 메소드를 호출하고 둘러싼 애플리케이션 컨텍스트(enclosing application context)에 대한 참조를 넘긴다.

6. 빈이 BeanPostProcessor 인터페이스를 구현하면 스프링은 postProcessBeforeInitialization() 메소드를 호출한다.
7. 빈이 InitializingBean 인터페이스를 구현하면 스프링은 afterPropertiesSet() 메소드를 호출한다. 마찬가지로 빈이 init-method와 함께 선언됐으면 지정한 초기화 메소드가 호출된다.
8. 빈이 BeanPostProcessor를 구현하면 스프링은 postProcessAfterInitialization() 메소드를 호출한다.
9. 이 상태가 되면 빈은 애플리케이션에서 사용할 준비가 된 것이며, 애플리케이션 컨텍스트가 소멸될 때까지 애플리케이션 컨텍스트에 남아 있다.
10. 빈이 DisposableBean 인터페이스를 구현하면 스프링은 destroy() 메소드를 호출한다. 마찬가지로 빈이 destroy-method와 함께 선언됐으면 지정된 메소드가 호출된다.

이제 스프링 컨테이너가 어떻게 생성되고 로드되는지 알게 됐지만, 우리는 아직 컨테이너에 아무것도 넣지 않았으므로 별 쓸모가 없다. 스프링 DI를 활용하려면 컨테이너에 애플리케이션 객체를 넣고 서로 연결해 줘야 한다. 빈 와이어링(wiring)은 2장에서 좀 더 상세히 살펴보겠다.

우선, 스프링 프레임워크가 무엇으로 구성되어 있고 최신 버전의 스프링이 제공하는 기능이 무엇인지 현재의 스프링 현황을 조사해 보자.

## 1.3 스프링 현황

앞서 살펴봤듯이 스프링 프레임워크는 DI, AOP 그리고 상투적인 코드 축소를 통해 자바 개발을 간소화하는 데 초점을 맞춘다. 이 모두 스프링이 수행하는 작업이며 유용하다. 그러나 스프링에는 눈에 보이는 것보다 더 많은 기능이 존재한다.

스프링 프레임워크는 스프링이 자바 개발을 쉽게 할 수 있는 여러 가지 방법을 제공한다. 하지만 스프링 프레임워크 자체를 넘어서 웹 서비스, REST, 모바일 그리고 NoSQL 등의 영역으로 스프링을 확장하여 코어 프레임워크를 구축하려는 프로젝트의 더 큰 생태계가 있다.

먼저, 코어 스프링 프레임워크가 어떤 도움을 주는지 자세히 분석해 보자. 그런 다음, 안목을 넓혀서 더 큰 스프링 포트폴리오의 다른 구성 요소를 검토하도록 하자.

### 1.3.1 스프링 모듈

스프링을 다운로드하여 libs 폴더를 보면 여러 JAR 파일이 있다. 스프링 4.0부터는 20개의 서로 다른 모듈이 스프링 프레임워크에 있으며, 각 모듈마다 세 개의 JAR 파일이 함께 있다(바이너리 클래스 라이브러리, 소스 JAR 파일, JavaDoc JAR 파일). 라이브러리 JAR 파일의 전체 리스트는 그림 1.6과 같다.

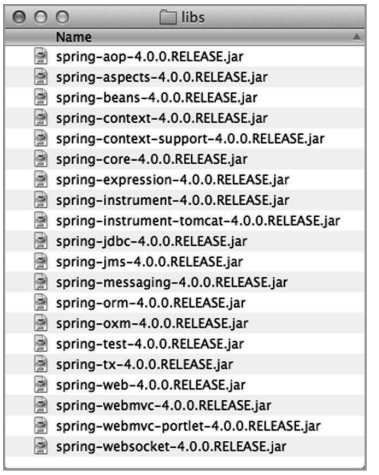



**그림 1.6 스프링 4.0은 20개의 서로 다른 모듈로 이루어져 있다.**

이 모듈들은 6개의 기능 카테고리에 속하게 되는데, 그림 1.7과 같다.

이 모듈들은 전체적으로 보면 엔터프라이즈 애플리케이션 개발에 필요한 모든 것을 제공한다. 그러나 애플리케이션을 만들 때 항상 이 스프링 프레임워크 모듈 전체를 이용해야 하는 것은 아니다. 스프링 프레임워크는 다른 프레임워크나 라이브러리와 쉽게 통합될 수 있으므로 원하는 기능이 스프링에 없어도 직접 개발할 필요 없이 다른 프레임워크나 라이브러리를 사용할 수 있다.

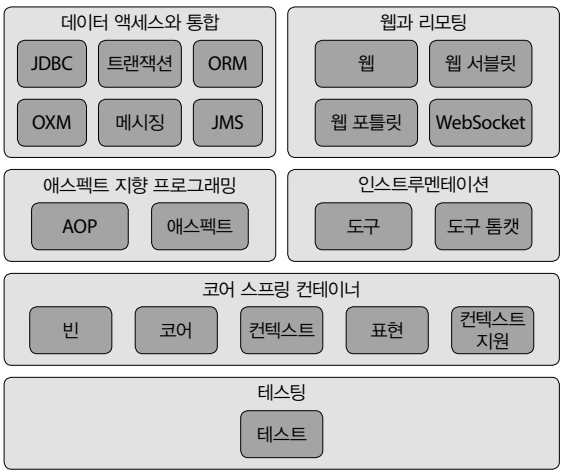



**그림 1.7 스프링 프레임워크는 6개의 잘 정의된 모듈로 구성되어 있다.**

스프링 모듈들이 전체적인 구조에서 각각 어떤 기능을 담당하는지 하나씩 살펴보기로 하자.

### 코어 스프링 컨테이너

스프링 프레임워크의 핵심은 스프링 애플리케이션에서 빈의 생성, 설정 그리고 처리 방법을 관리하는 컨테이너다. 이 모듈 내에서 DI를 제공하는 스프링 빈 팩토리를 확인한다. 빈 팩토리를 구성하다 보면 여러 가지 스프링의 애플리케이션 컨텍스트 구현체가 보이는데, 각각은 스프링을 구성하는 다른 방법을 제공한다.

빈 팩토리와 애플리케이션 컨텍스트 외에도 이 모듈은 이메일, JNDI 액세스, EJB 통합 그리고 스케줄링 등의 다양한 엔터프라이즈 서비스도 제공한다.

모든 스프링의 모듈은 코어 컨테이너 위에 구축된다. 애플리케이션 구성 시 암묵적으로 코어 컨테이너 클래스를 사용한다. 코어 모듈은 스프링의 DI를 살펴보는 2장을 시작으로 이 책의 전반에 걸쳐 다룬다.

### 스프링의 AOP 모듈

스프링은 AOP 모듈을 통해 애스펙트 지향 프로그래밍을 풍부하게 지원한다. AOP 모듈은 스프링 애플리케이션에서 애스펙트를 개발할 수 있는 기반이 되는 것으로서, DI처럼 애플리케이션 객체 간의 결합도를 낮추는 데 기여한다. AOP는 주로 애플리케이션 전체에 걸친 관심사(트랜잭션이나 보안 등)와 각 객체 간의 결합도를 낮추는 데 이용된다.

스프링의 AOP 지원에 대해서는 4장에서 자세히 살펴본다.

### 데이터 액세스와 통합

JDBC를 이용해 작업하다 보면 커넥션을 얻어 오고, 질의객체를 생성하고, 결과 집합을 처리하고, 커넥션을 닫는 코드를 수없이 반복하게 된다. 스프링의 JDBC와 데이터 액세스 객체(DAO, Data Access Objects) 모듈은 이렇게 반복되는 코드를 추상화하므로 이 모듈을 이용하면 데이터베이스 관련 코드를 깔끔하고 간단하게 만들 수 있고, 데이터베이스 리소스를 닫지 않아서 발생할 수 있는 문제를 예방한다. 또한, 이 모듈에는 여러 종류의 데이터베이스 서버가 제공하는 오류 메시지에 의미 있는 예외 계층이 추가되어 있다. 이 모듈을 이용하면 더 이상 암호문 같은 SQL 오류 메시지를 이해하려고 머리를 싸매지 않아도 된다!

스프링의 ORM 모듈은 JDBC보다 객체 관계 매핑(ORM, Object-Relational Mapping) 도구를 선호하는 사람들을 위한 것이다. 스프링의 ORM 모듈은 DAO 모듈 위에 올라가서 ORM 솔루션용 DAO를 만드는 편리한 방법을 제공한다. 스프링은 고유한 ORM 솔루션을 구현하지 않고, 하이버네이트(Hibernate), 자바 퍼시스턴스(Java Persistence), iBATIS SQL Maps 등 널리 사용되는

ORM 프레임워크와의 연결 고리를 제공한다. 스프링의 트랜잭션 관리 기능은 JDBC뿐만 아니라 이 ORM 프레임워크들도 지원한다.

10장에서 스프링 데이터 액세스를 살펴볼 때 스프링의 템플릿 기반 JDBC 추상화가 어떻게 JDBC 코드를 크게 간소화하는지 알아본다.

이 모듈은 또한 메시징을 통해 다른 애플리케이션과의 비동기식 통합을 위하여 JMS(Java Message Service)를 이용한 스프링 추상화를 포함한다. 그리고 스프링 3.0에서 이 모듈은 원래 스프링 웹 서비스(Spring Web Services) 프로젝트의 일부분이었던 객체-XML 매핑(OXM, Object-XML Mapping)을 포함한다.

또한, 이 모듈은 스프링의 AOP 모듈을 이용하여 스프링 애플리케이션에서 객체들의 트랜잭션 관리 서비스를 제공한다.

#### 웹과 리모팅

MVC(Model-View-Controller) 패러다임은 사용자 인터페이스가 애플리케이션 로직과 분리되는 웹 애플리케이션을 만드는 경우에 일반적으로 사용되는 접근 방식이다. 자바의 유명한 MVC 프레임워크 가운데에는 아파치 스트러츠(Apache Struts), JSF, 웹워크(WebWork), 태피스트리(Tapestry) 등이 있다.

스프링이 다양한 유명 MVC 프레임워크와 잘 통합되기는 하지만, 웹과 리모팅 모듈에는 애플리케이션의 웹 계층[5]에서 결합도를 낮추는 (스프링의 기술을 잘 반영하는) MVC 프레임워크가 별도로 만들어져 있다. 스프링의 MVC 프레임워크는 5~7장에서 살펴본다.

사용자 접촉(user-facing) 웹 애플리케이션뿐만 아니라 이 모듈은 다른 애플리케이션과 상호작용하는 애플리케이션을 개발하기 위한 다양한 리모팅 옵션도 제공한다. 스프링의 리모팅 기능에는 RMI(Remote Method Invocation), Hessian, Burlap, JAX-WS 그리고 스프링의 HTTP 호출자 등이 있다. 스프링은 또한 REST API에 대해 최고의 지원을 제공한다.

15장에서는 스프링 리모팅을 알아보고, REST API 생성과 사용 방법을 16장에서 배운다.

#### 인스트루멘테이션

스프링의 인스트루멘테이션(instrumentation) 모듈은 JVM에 에이전트를 추가하는 기능을 제공한다. 정확히는 톰캣용 위빙 에이전트(weaving agent)를 제공하는데, 톰캣은 클래스로더

5 역주 layer와 tier는 의미가 다소 다르지만, 이 책에서는 의미를 봤을 때 둘 다 큰 차이가 없기에 layer와 tier 모두 '계층'으로 번역했다.

(classloader)에 의해 로드되는 클래스 파일을 변환한다.

이해해야 할 게 많아 보여도 크게 걱정하지 않아도 된다. 이 모듈이 제공하는 인스트루멘테이션은 유스 케이스(use case)가 적은 특징을 가지고 있고, 이 책에서는 이 모듈에 대해서 다루지 않을 것이다.

#### 테스팅

개발자가 작성하는 테스트의 중요성을 인식하여 스프링은 스프링 애플리케이션 테스트에 전념하는 모듈을 제공한다.

이 모듈 안에서 JNDI, 서블릿(servlet) 그리고 포틀릿(portlet)과 동작하는 코드의 단위 테스트 작성을 위한 모의 객체 구현을 확인한다. 통합 테스트의 경우, 이 모듈은 스프링 애플리케이션 컨텍스트에서 빈을 로드하고 이 컨텍스트에 있는 빈과의 작업을 지원한다.

이 책의 많은 예제가 스프링이 제공하는 테스트 기능을 활용한 테스트에서 가동된다.

### 1.3.2 스프링 포트폴리오

스프링은 눈에 보이는 게 다가 아니다. 실제로 스프링 프레임워크 다운로드에 있는 것보다 더 많은 기능이 존재한다. 만일 코어 스프링 프레임워크에서 멈춘다면, 다양한 스프링 포트폴리오가 제공하는 풍부한 잠재력을 놓치게 된다. 전체 스프링 포트폴리오에는 코어 스프링 프레임워크와 서로 연관되어 구축된 다양한 프레임워크와 라이브러리가 있다. 전체 스프링 포트폴리오는 자바 개발의 모든 측면에서 스프링 프로그래밍 모델을 제공한다.

스프링 포트폴리오가 제공하는 모든 기능을 설명하려면 책 몇 권의 분량이 필요하다. 따라서 여기서는 코어 스프링 프레임워크의 범위를 넘어선 일부 내용을 포함하여 스프링 포트폴리오의 주요 내용을 몇 가지 살펴보겠다.

#### 스프링 웹 플로

스프링 웹 플로(Spring Web Flow)는 목표를 향해 사용자를 안내하는(마법사나 장바구니를 떠올리면 된다) 대화형, 흐름 기반 웹 애플리케이션 구축을 지원하기 위한 스프링의 핵심 MVC 프레임워크다. 스프링 웹 플로는 8장에서 자세히 논의하겠으며, 더 많은 정보는 http://projects.spring.io/spring-webflow/에서 확인하기 바란다.

### 스프링 웹 서비스

코어 스프링 프레임워크는 웹 서비스로 스프링 빈을 선언적으로 배포할 수 있지만, 이러한 서비스는 반드시 구조적으로 열악한 구현 우선(contract-last) 모델 기반으로 한다. 서비스에 대한 규약은 빈의 인터페이스에서 결정된다. 스프링 웹 서비스(Spring Web Services)는 규약 우선(contract-first) 웹 서비스 모델을 제공하는데, 여기서 서비스 규약을 충족하기 위해서 서비스 구현체가 작성된다.

이 책에서는 스프링 웹 서비스에 대해서는 논하지 않겠다. 그렇지만 http://docs.spring.io/spring-ws/site/를 통해서 스프링 웹 서비스에 대해 많은 것을 얻을 수 있다.

### 스프링 시큐리티

보안은 많은 애플리케이션의 핵심 요소다. 스프링 AOP를 이용하여 구현된 스프링 시큐리티(Spring Security)는 스프링 기반 애플리케이션에 선언적 보안 메커니즘을 제공한다. 애플리케이션의 웹 레이어(web layer)에 스프링 시큐리티를 추가하는 방법은 9장에서 살펴보겠다. 14장에서는 스프링 시큐리티로 다시 돌아가서 메소드 호출을 어떻게 안전하게 확보하는지 알아보자. 더 많은 정보는 스프링 시큐리티 홈페이지 http://projects.spring.io/spring-security/에서 확인한다.

### 스프링 인티그레이션

많은 엔터프라이즈 애플리케이션은 다른 엔터프라이즈 애플리케이션과 상호작용해야 한다. 스프링 인티그레이션(Spring Integration)은 몇 가지 공통적인 통합 패턴의 구현체를 스프링의 선언적 방식으로 제공한다.

이 책에서는 스프링 인티그레이션은 다루지 않는다. 하지만 스프링 인티그레이션에 대한 더 많은 정보는 마크 피셔(Mark Fisher), 요나스 파트너(Jonas Partner), 마리우스 보고예비치(Marius Bogoevici) 그리고 이바인 펄드(Iwein Fuld)가 쓴 《Spring Integration in Action》(Manning, 2012, www.manning.com/fisher/)을 읽어 보거나 스프링 인티그레이션 홈페이지인 http://projects.spring.io/spring-integration/을 살펴보기 바란다.

### 스프링 배치

데이터의 일괄 작업이 필요하다면 배치(batch) 처리만큼 좋은 방법은 없다. 배치 애플리케이션을 개발한다면 스프링 배치(Spring Batch)를 이용해 스프링의 신뢰성 있는 POJO 지향 개발 모델을 활용한다.

스프링 배치는 이 책의 범위 밖이다. 하지만 아르노 코골르느(Arnaud Cogoluègnes)와 티에리 탕플리에(Thierry Templier), 개리 그레고리(Gary Gregory) 그리고 올리비에 바조드(Olivier Bazoud)가 쓴 《Spring Batch in Action》(Manning, 2011, www.manning.com/templier/)에서 많은 정보를 얻을 수 있다. 또한 스프링 배치 홈페이지 http://projects.spring.io/spring-batch/에서 스프링 배치에 대해 배울 수 있다.

### 스프링 데이터

스프링 데이터는 스프링 내 모든 종류의 데이터베이스와의 구동을 쉽게 해 준다. 오랜 시간, 엔터프라이즈 애플리케이션에서 관계형 데이터베이스가 흔했지만, 최근의 애플리케이션은 모든 데이터가 테이블 내의 행(column), 열(row)을 가지고 제공되지 않는다는 것을 알고 있다. 보통 NoSQL 데이터베이스[6]로 불리는, 새로운 종류의 데이터베이스는 새로운 방식의 구동을 제공하는데, 전통적인 관계형 데이터베이스보다 훨씬 적절한 데이터를 가지고 구동된다.

MongoDB 같은 문서 데이터베이스나 Neo4j 같은 그래프 데이터베이스 또는 심지어 전통적인 관계형 데이터베이스를 사용하더라도, 스프링 데이터가 제공하는 것은 지속성을 위한 간단한 프로그래밍 모델이다. 이것은 많은 데이터베이스 타입에 대해 자동 저장소 메커니즘이 저장소 구현체를 생성해 준다는 것을 포함한다.

Java Persistence API(JPA) 개발을 간소화하기 위한 스프링 데이터 사용에 대해 11장에서 알아보고, 12장에서 몇몇의 NoSQL 데이터베이스로 논의를 넓힌다.

### 스프링 소셜

소셜 네트워크(social network)는 인터넷에서 떠오르는 트렌드로, 페이스북(Facebook)이나 트위터(Twitter) 등의 소셜 네트워크 사이트와의 통합 기능을 갖춘 애플리케이션도 계속 증가하는 추세다. 관심 있는 분야라면 스프링의 소셜 네트워크 확장 기능인 스프링 소셜(Spring Social)을 살펴보기 바란다.

하지만 스프링 소셜은 그저 몇 마디 말이나 친구 그 이상에 관한 것이다. 그 이름에도 불구하고, 스프링 소셜은 social이라는 단어보다는 connect라는 단어에 더 가깝다. 소셜의 목적이 없을지도 모르는 많은 REST API로 스프링 애플리케이션을 연동하는 것을 도와준다.

---

6 나는 NoSQL보다는 '비관계형' 또는 '스키마 없는'이라고 표현하는 것을 더 좋아한다. NoSQL로 부르는 것은 데이터베이스 모델이 아닌 쿼리 언어를 문제시하는 것 같다.

공간의 제약상 스프링 소셜을 이 책에서 다루지 않지만, 스프링이 페이스북이나 트위터에 접속하는 데 어떻게 도움을 주는지 관심 있다면, https://spring.io/guides/gs/accessing-facebook/ 그리고 https://spring.io/guides/gs/accessing-twitter/에서 시작 가이드를 볼 수 있다.

### 스프링 모바일

모바일 애플리케이션은 소프트웨어 개발에서 또 하나의 중요한 분야다. 많은 사용자들이 선호하는 단말이 스마트폰과 태블릿 기기로 이동했다. 스프링 모바일(Spring Mobile)은 모바일 웹 애플리케이션 개발을 지원하는 스프링 MVC의 새로운 확장 기능이다.

### 안드로이드용 스프링

스프링 모바일에 관련된 것은 스프링 안드로이드(Spring Android) 프로젝트다. 이 프로젝트의 목표는 스프링 프레임워크가 제공하는 간소화의 일부를 사용한, 안드로이드 기반 장치용 기본 애플리케이션의 개발이다. 처음에 이 프로젝트가 제공한 것은 스프링의 RestTemplate의 한 버전으로, 안드로이드 애플리케이션에서 이용할 수 있다. 그것은 또한 REST API와 연결되는 native 안드로이드 앱을 활성화시키기 위해 스프링 소셜과도 연동된다.

이 책에서 안드로이드용 스프링을 논하지는 않겠지만, http://projects.spring.io/spring-android/에서 배울 수 있다.

### 스프링 부트

스프링은 많은 프로그래밍 작업을 훌륭하게 단순화한다. 어쩔 수 없이 코딩해야 했을 상용구 코드 대부분을 줄이거나 제거까지 가능하게 해준다. 스프링 부트(Spring Boot)는 흥미로운 새 프로젝트인데, 스프링 그 자체를 간소화하기 위해 스프링으로 개발한다.

스프링 부트는 자동 설정(automatic configuration) 기술을 많이 사용하는데, 이것을 통해 대부분의 스프링 설정을 제거한다. 그리고 Marven을 사용하든 Gradle을 사용하든 다수의 스타터(starter) 프로젝트가 스프링 프로젝트 빌드 파일의 크기를 줄인다.

이 책의 마지막인 21장에서 스프링 부트를 살펴본다.

## 1.4 스프링의 새로운 기능

이 책의 제3판이 나왔을 때, 스프링의 최신 버전은 버전 3.0.5였다. 그때가 대략 3년 전이었고, 그 이후로 많은 변화가 있었다. 스프링 프레임워크는 세 개의 중요한 릴리스(3.1, 3.2 그리고 이제 4.0)를 선보였고, 각각의 릴리스는 새로운 특징과 개선으로 편리한 애플리케이션 개발을 가져왔다. 그리고 스프링 포트폴리오의 다른 여러 구성들에 중요한 수정이 있었다.

《스프링 인 액션》 제4판에서는 새롭게 릴리스된 흥미롭고 유용한 특징을 많이 다루고자 했다. 새로워진 스프링의 기능을 간단히 파악해 보자.

### 1.4.1 스프링 3.1에서 새로워진 기능

스프링 3.1에서는 여러 가지 유용한 새 기능과 개선이 있었으며, 이 중 대부분이 설정을 간소화하고 개선하는 데 초점이 맞춰졌다. 게다가 스프링 3.1은 선언적 캐싱 지원뿐만 아니라 스프링 MVC도 많이 개선되었다. 아래에 스프링 3.1의 중요 부분을 간단히 나열했다.

- 다양한 환경(개발, 테스트, 제조 같은)에서 서로 다른 설정을 선택하는 일반적인 문제를 다루기 위해 스프링 3.1은 환경 프로파일을 도입했다. 예를 들면, 프로파일은 다양한 데이터 소스 빈(data source bean)을 고를 수 있게 하는데, 이 데이터 소스 빈은 애플리케이션이 있는 환경에 의존적이다.
- 스프링 3.0의 자바 기반 설정을 바탕으로 한 구축과 함께 스프링 3.1이 추가한 것은 여러 enable 애너테이션이다. 이는 단일 애너테이션과 함께 있는 스프링의 어떤 특징을 전환해 준다.
- 스프링에도 차츰 선언적 캐싱 지원이 쓰이게 되었다. 이것은 캐싱 경계를 정의할 수 있도록 해 주며, 간단한 애너테이션으로 룰을 정의할 수 있게 해 준다. 이미 다뤄 본 트랜잭션 경계 선언 방법과 비슷하다.
- 새로운 c-네임스페이스 덕분에 생성자 주입이 가능해졌다. 스프링 2.0에서 p-네임스페이스 덕분에 프로퍼티 주입이 가능해진 것처럼 이 역시 간결한 애트리뷰트(attribute) 지향적 스타일을 가진다.
- 스프링은 서블릿 3.0을 지원하기 시작했는데, 서블릿 선언 기능과 web.xml 대신 자바 기반의 설정 내에서 필터를 선언하는 기능을 포함한다.
- 스프링의 JPA 지원이 개선되면서 persistence.xml 없이 JPA를 설정하는 것이 가능해졌다.

스프링 3.1은 스프링 MVC에 대한 부분 또한 강화되었다.

- 애트리뷰트 모델에 대한 경로 변수의 자동 바인딩이 추가되었다.
- 요청을 위한 Accept와 Content-Type 헤더에 매칭하기 위한 @RequestMapping의 produces와 consumes 애트리뷰트를 포함한다.
- @RequestPart 애너테이션으로 핸들러 메소드 파라미터에 대한 다중 요청 부분의 바인딩이 가능하다.
- 플래시(Flash) 애트리뷰트(redirect에서 살아남는 애트리뷰트)와 RedirectAttributes 타입을 지원하며, 요청과 요청 사이에서 플래시 애트리뷰트를 전달한다.

스프링 3.1의 새로워진 점만큼이나 중요한 것은 스프링 3.1부터 더 이상 유효하지 않은 사항들이다. 특히, 스프링의 JpaTemplate과 JpaDaoSupport 클래스들은 중요도가 사라지면서 원래 있던 EntityManager가 지지를 받아 사용되었다. 비록 중요도는 떨어졌지만 스프링 3.2에 여전히 존재는 했다. 그러나 이제는 이 클래스들을 사용하면 안 되는데, JPA 2.0 지원을 위한 업그레이드가 되지 않았고 스프링 4에서 제거되었기 때문이다.

이제 스프링 3.2의 새로워진 점을 살펴보자.

### 1.4.2 스프링 3.2에서 새로워진 기능

스프링 3.1이 설정 개선에 초점을 맞추고, 스프링 MVC를 포함한 기타 개선점이 적었던 반면, 스프링 3.2는 주로 스프링 MVC에 초점을 맞춘 릴리스였다. 스프링 MVC 3.2는 아래와 같이 향상된 사항을 자랑한다.

- 스프링 3.2 컨트롤러는 서블릿 3의 비동기식 요청을 이용하여 요청 처리를 분리시키고, 분리된 스레드(thread) 내에서 처리되도록 한다.
- 스프링 2.5 이후로 스프링 MVC 콘트롤러 테스트가 POJO처럼 용이해졌다고는 하지만, 스프링 3.2에서는 스프링 MVC 테스트 프레임워크가 포함됨으로써 컨트롤러에 대한 더욱 다양한 테스트 작성이 가능해졌다. 이제 서블릿 컨테이너 없이도 컨트롤러로서의 역할을 수행한다.
- 스프링 3.2는 개선된 컨트롤러 테스트뿐만 아니라 RestTemplate 기반의 클라이언트 테스트를 지원하며, 실제 REST 엔드포인트에 요청을 보내지 않고도 가능하다.
- @ControllerAdvice 애너테이션은 공통의 @ExceptionHandler, @InitBinder, @ModelAttribute 메소드가 하나의 클래스에 집합되도록, 그리고 모든 컨트롤러에 적용되도록 만들어 준다.

- 스프링 3.2 이전에는 전체 콘텐츠 협상(content negotiation) 지원은 ContentNegotiatingViewResolver을 통해서만 가능했었다. 하지만 스프링 3.2에서는 전체 콘텐츠 협상이 스프링 MVC를 통해서 가능하게 되었다. 콘텐츠 소비 및 생산을 위한 메시지 컨버터에 의존하는 컨트롤러 메소드에서조차도 가능하다.
- 스프링 MVC 3.2는 @MatrixVariable이라는 새로운 애너테이션을 포함하는데, 요청(request)의 행렬 변수를 핸들러 메소드 파라미터로 바인딩하기 위한 것이다.
- 추상화 기초 클래스 AbstractDispatcherServletInitializer는 web.xml 없이 DispatcherServlet 설정을 편리하게 사용할 수 있도록 지원한다. 마찬가지로 AbstractAnnotationConfigDispatcherServletInitializer라고 이름 지어진 서브클래스도 자바 기반의 설정을 가진 스프링을 설정하고 싶을 때 사용된다.
- ResponseEntityExceptionHandler 클래스는 DefaultHandlerExceptionResolver의 대안으로 사용되기 위해 추가되었다. ResponseEntityExceptionHandler 메소드는 ModelAndView 대신에 ResponseEntity<Object>를 리턴한다.
- RestTemplate과 @RequestBody 인자는 일반적인 타입을 지원한다.
- RestTemplate과 @RequestMapping 메소드는 HTTP PATCH 메소드를 지원한다.
- 매핑된 인터셉터들은 인터셉터 프로세싱으로부터 배제되기 위해 URL 패턴을 지원한다.

스프링 MVC가 스프링 3.2의 주된 이야기이긴 했지만, MVC 말고도 다른 개선점들이 추가되었다. 여기서 스프링 3.2의 가장 흥미로운 새로운 특징을 보자.

- @Autowired, @Value 그리고 @Bean 애너테이션은 커스텀 주입과 빈 선언 애너테이션을 생성하기 위한 메타-애너테이션(meta-annotation)으로 사용된다.
- @DateTimeFormat 애너테이션은 더 이상 JodaTime(자바의 날짜 관련 클래스)에 강력한 의존성을 갖지 않는다. 만약 JodaTime이 존재한다면 사용되고, 그렇지 않다면 SimpleDateFormat이 사용된다.
- 스프링의 선언적 캐싱 지원은 JCache 0.5를 위한 초기 지원을 가진다.
- 날짜와 시간을 파싱(parsing)하고 렌더링하는 글로벌 포맷(global format)을 정의한다.
- 통합 테스트는 WebApplicationContext를 설정하고 로드한다.
- 통합 테스트는 요청 범위와 세션 범위의 빈에 대하여 테스트한다.

이 책의 여러 장, 특히 웹과 REST 관련 장에서 스프링 3.2의 많은 특징을 살펴볼 수 있다.

### 1.4.3 스프링 4.0에서 새로워진 기능

스프링 4.0은 스프링의 가장 최신 릴리스다. 스프링 4.0에는 다음과 같이 흥미롭고 새로운 기능들이 많이 있다.

- 스프링은 이제 JSR-356의 웹 소켓을 위한 자바 API 지원을 포함하여 웹 소켓 프로그래밍을 지원한다.
- 웹 소켓의 저수준 API도 제공하지만, 매우 높은 수준의 추상화를 추구하는 SockJS 기반이나 STOMP 서브 프로토콜을 지원하는 매우 높은 수준의 메시지 기반 모델의 웹 소켓도 지원한다.
- 새로운 메시징 모듈은 많은 타입을 가지며, 스프링 통합 프로젝트에서 넘어왔다. 이 메시징 모듈은 스프링의 SockJS/STOMP를 지원하며, 메시지를 퍼블리싱(게시)하기 위한 템플릿 기반의 지원을 포함하고 있다.
- 스프링 4.0은 자바 8의 특징인 람다식(lambdas)을 지원하는 첫 번째 프레임워크 중 하나이다. 무엇보다도 읽기에 훨씬 깨끗하고 쉬운 콜백 인터페이스(JdbcTemplate과 함께 있는 RowMapper)로 작업할 수 있다.
- JSR-310의 데이터 및 시간 API를 포함하는 자바 8 지원을 통해, 개발자들이 java.util.Date 또는 java.util.Calendar에서 제공되는 것보다 훨씬 풍부한 API로 날짜와 시간을 작업하는 기회를 제공받을 수 있다.
- Groovy로 개발된 애플리케이션에 대해 매끄러운 프로그래밍 경험이 또한 추가되었으며, Groovy를 사용하여 스프링 애플리케이션을 쉽게 개발할 수 있다. Grails의 BeanBuilder를 사용하여 Groovy로 스프링 애플리케이션을 설정할 수 있다.
- 조건부 빈 생성에 대한 일반적 지원이 추가되면서 개발자가 정의한 조건이 만족되었을 때만 빈이 생성되도록 선언할 수 있다.
- 스프링 4.0이 포함하는 또 한 가지는 스프링의 RestTemplate에 대한 새로운 비동기식 구현으로, RestTemplate는 응답을 반환하고, 연산이 끝났을 때 콜백을 호출한다.
- JMS 2.0, JTA 1.2, JPA 2.1 그리고 빈 Validation 1.1을 포함한 많은 JEE 스펙(specs) 지원이 추가되었다.

보다시피 흥미로운 사항들이 스프링 프레임워크의 최신 버전에 오르게 되었다. 앞으로 이 책을 통해서 스프링이 가진 기존의 특징뿐 아니라 이러한 새로운 특징도 자세히 살펴보도록 하자.

## 1.5 요약

지금까지 스프링을 이용해 어떤 일을 할 수 있는지 알아봤다. 스프링은 엔터프라이즈 자바 개발자의 작업을 쉽게 하고, 결합도가 낮은 코드를 만드는 것을 목적으로 한다. 여기서 핵심은 DI와 AOP다.

1장에서는 스프링의 DI에 대해 간단히 살펴보았다. DI는 객체들이 상호 간의 종속관계나 구체적인 구현 방법을 알 필요가 없도록 애플리케이션 객체들을 연결하는 방법이다. 종속객체를 필요로 하는 객체가 직접 종속객체를 얻어 오는 것이 아니라, 종속객체를 필요로 하는 객체에 종속객체가 부여되는 방식이다. 종속객체를 필요로 하는 객체는 인터페이스를 통해서 주입된 객체만 알므로 결합도가 낮아진다.

DI에 이어 스프링의 AOP 지원에 대해서도 간단히 살펴보았다. AOP를 이용하면 일반적으로 애플리케이션의 여러 곳에 흩어져 있는 로직을 한 곳으로, 즉 애스펙트로 모을 수 있다. 각 애스펙트는 스프링이 빈을 와이어링할 때 런타임에 연결되어 효과적으로 빈에 새로운 기능을 부여한다.

DI와 AOP는 스프링의 중추에 해당한다. 스프링 프레임워크를 잘 사용하려면 이 두 가지 주요 기능을 잘 이해하고 사용해야 한다. 이번 장에서는 스프링의 DI와 AOP 기능의 기본적인 사항들을 알아봤지만, 뒤에 나오는 여러 장을 통해 DI와 AOP를 자세히 알아보겠다.

그럼, 이제 잔소리는 그만 접어 두고 2장으로 이동하여 스프링에서 DI를 이용해 객체들을 연결하는 방법을 알아보자.

CHAPTER

# 2

# 빈 와이어링(묶기)

**이 장에서 다룰 내용**

- 빈(bean) 선언
- 생성자 주입과 세터 주입
- 빈 와이어링
- 빈의 생성과 소멸 제어

영화가 끝난 후에도 자리에 앉아 엔딩 크레딧이 올라가는 것을 끝까지 보고 있으면, 영화 하나를 제작하더라도 놀라울 정도로 많은 사람이 참여한다는 것을 알 수 있다. 배우, 대본작가, 감독, 제작자 등 흔히 알고 있는 역할 외에도 작곡가, 특수효과 담당, 아트 디렉터 또한 영화에서 중요한 역할을 하는 사람들이다. 거기에 세트 조립, 음향기사, 의상, 메이크업 아티스트, 스턴트, 코디네이터, 홍보, 카메라맨 보조, 세트 디자이너, 전기 주임, 식사 조달 담당(아마 제일 중요한 사람)도 없어서는 안 될 사람들이다.

이 많은 사람이 서로 대화도 없이 일을 하는 경우 영화가 어떻게 될지 생각해 보자. 영화 스튜디오에 모인 사람들이 상호 간의 역할 조정 없이 각자 따로따로 일한다고 가정해 보자. 감독이 큐 사인을 내지 않고 묵묵히 앉아만 있으면, 카메라맨도 촬영을 시작하지 못하고 우두커니 기다릴 수밖에 없다. 하지만 이것은 사소한 문제다. 왜냐하면 여배우는 트레일러에 앉아 나오지도 않았고, 고용된 전기 담당자도 없어서 아직 전기도 안 들어왔기 때문이다. 개중에는 정말로 이런 상황에서 만들어지지 않았을까 하는 생각이 드는 영화도 있기는 하지만, 대부분의 영화(특히 좋은

영화)는 히트작을 만들겠다는 공통 목표를 향해 수천 명의 사람들이 함께 일한 결과로 만들어진다.

소프트웨어도 크게 다르지 않다. 아무리 작은 애플리케이션이라도 어떤 비즈니스 목적을 달성하기 위해 협력하는 많은 객체로 구성된다. 수많은 객체가 자신의 일을 하기 위해 필요한 상대방을 알고 있으며 서로 대화해야 한다. 예를 들어, 온라인 쇼핑 애플리케이션의 주문관리 컴포넌트는 상품관리 컴포넌트 그리고 신용카드 인증 컴포넌트와 함께 일해야 한다. 또한 각 컴포넌트는 데이터베이스를 읽고 쓰기 위해서 데이터 액세스 컴포넌트와 협력해야 한다.

하지만 1장에서 살펴보았듯이, 객체 생성(construction)이나 검색(lookup)을 통하여 애플리케이션 객체 사이의 관계를 형성하는 전통적인 접근 방법은, 재사용과 단위 테스트가 어려운 복잡한 코드가 작성되게 만든다. 운이 좋다 해도 여전히 객체들은 원래 할 일보다 많은 일을 해야 하며, 운이 나쁘면 서로의 결합도가 높아져서 재사용도 테스트도 모두 어려운 구조가 된다.

스프링을 사용하는 애플리케이션에서는 각 객체가 자신의 일을 하기 위해 필요한 다른 객체를 직접 찾거나 생성할 필요가 없다. 컨테이너가 협업할 객체에 대한 레퍼런스를 주기 때문이다. 예를 들어, 주문관리 컴포넌트의 경우 신용카드 인증 컴포넌트가 필요해도 이 컴포넌트를 직접 생성할 필요가 없다. 그냥 가만히 손만 들고 있으면 누군가가 필요한 걸 쥐어 준다.

애플리케이션 객체 간의 이러한 연관관계 형성 작업이 바로 종속객체 주입(DI) 개념의 핵심이며, 이를 보통 **와이어링**(wiring)이라 한다. 이번 장에서는 스프링을 이용한 빈 와이어링의 기초를 다룬다. DI는 스프링의 가장 밑바탕을 이루는 개념으로서 스프링 기반 애플리케이션을 개발할 때는 언제나 이용하게 되는 기법이다.

스프링에서는 빈을 엮는 많은 방법이 있다. 시작에 앞서, 스프링 컨테이너를 설정하기 위한 가장 흔한 세 가지 접근에 대해 알아보자.

## 2.1 스프링 설정 옵션 알아보기

1장에서 언급했듯이 스프링 컨테이너는 애플리케이션 내에서의 빈 생성 및 DI를 통해, 그러한 객체 사이의 관계 조정에 책임을 지고 있다. 하지만 어떤 빈을 생성할지 그들을 어떻게 엮을지 스프링에게 말해 주는 것은 개발자로서 당신의 책임이다. 빈 와이어링 명세서를 나타내는 데 있어 스프링은 믿을 수 없을 정도로 유연하며, 세 가지 기본적인 와이어링 메커니즘을 제공한다.

- XML에서의 명시적 설정
- 자바에서의 명시적 설정
- 내재되어 있는 빈을 찾아 자동으로 와이어링하기

언뜻 보기에는 이러한 세 가지 설정 옵션을 제공하는 것이 스프링을 복잡하게 만드는 것처럼 보일지도 모른다. 각각의 설정 테크닉이 제공하는 것에는 겹치는 부분이 있고, 어느 기술이 주어진 상황에 가장 적용할 만한지 결정하기 어려울 수 있다. 그러나 너무 상심하지 말자. 대부분 이 선택은 개인 취향의 문제이며 마음에 드는 것으로 선택한다.

스프링에서 빈을 어떻게 와이어링할지 다양한 선택을 할 수 있는건 좋지만, 하나를 고르긴 해야 한다.

여기에 정해진 정답은 없다. 어떤 방법을 선택하든 프로젝트에 적절하게 사용할 수 있다. 그리고 누가 하나의 선택만 해야 한다고 말하던가? 스프링의 설정 스타일은 짜맞추는 것이므로 어떤 빈을 와이어링하기 위해 XML을 선택할 수도 있고, 스프링의 자바 기반 설정(JavaConfig)을 사용할 수도 있으며, 스프링이 자동으로 찾아낼 수도 있다.

그렇긴 하지만, 할 수만 있다면 자동 설정을 추천한다. 명시적인 설정이 적을수록 좋다. 명시적인 빈 설정을 해야 할 때는(유지보수하지 않는 소스 코드에 대한 빈을 설정할 때와 같은), 타입 세이프(type-safe)를 보장하고 더욱 강력한 JavaConfig를 XML보다 선호한다. 마지막으로, 사용하고자 하는 XML 네임스페이스의 기능이 JavaConfig에 없을 경우 XML을 사용한다.

이 세 가지 테크닉을 이번 장에서 상세하게 알아볼 것이며 이 책의 전체에 적용해 보자. 지금은 맛보기 테스트를 하며 각각이 어떤지를 알아보자. 스프링 설정에 대한 첫 번째 맛보기로 스프링 자동 설정을 살펴보자.

## 2.2 자동으로 빈 와이어링하기

이번 장의 뒷부분에서는 자바와 XML에서 스프링 와이어링을 나타내는 방법을 보게 된다. 명시적 와이어링 테크닉의 많은 쓰임새를 보겠지만, 사용의 용이성 측면에서 스프링 자동 설정보다 나은 것은 없다. 스프링이 자동으로 설정된다면 명시적 와이어링 빈을 건드릴 필요가 있을까?

스프링은 두 가지 방법으로 오토와이어링을 수행한다.

- **컴포넌트 스캐닝** — 스프링은 애플리케이션 컨텍스트에서 생성되는 빈을 자동으로 발견한다.
- **오토와이어링** — 스프링은 자동으로 빈 의존성을 충족시킨다.

컴포넌트 스캐닝과 오토와이어링을 모두 사용하면 강력하고, 명시적 설정을 최소한으로 유지하는 데 도움이 된다.

컴포넌트 스캐닝과 오토와이어링 개념을 설명하기 위해 스테레오 시스템의 몇 가지 컴포넌트를 나타내는 몇 개의 빈을 만들어 보자. CompactDisc 클래스를 만들면 스프링이 시작 시에 이를 발견해서 빈으로 생성한다. 그리고 CDPlayer 클래스를 만들면 스프링이 이를 발견하고 여기에 CompactDisc 빈을 주입한다.

## 2.2.1 발견 가능한 빈 만들기

MP3 파일과 스트리밍 음악 시대에서 콤팩트 디스크는 매우 희귀하고 구식이다. 물론 카세트 테이프, 8트랙, 비닐 레코드만큼은 아니지만, CD는 마지막으로 남은 물리적 음악 저장 장치로 점점 더 희귀해질 것이다.

어쨌든 CD는 DI가 어떻게 동작하는지 설명하기에 좋은 사례다. CD 플레이어는 CD를 삽입(또는 주입)하지 않는다면 가치가 거의 없다. CD 플레이어는 일을 수행하기 위해서는 CD에 의존적이다.

스프링에서 어떠한 삶을 가지는지 알기 위해서 자바로 CD 개념을 만들어 보자. 다음 코드는 CD를 정의하는 인터페이스인 CompactDisc를 나타낸다.

**코드 2.1 CompactDisc 인터페이스는 자바언어를 사용하여 CD 개념을 정의**

```
package soundsystem;

public interface CompactDisc {
  void play();
}
```

CompactDisc 인터페이스의 스펙(specifics)은 중요하지 않다. 중요한 점은 인터페이스로 정의했다는 점이다. 인터페이스로서 CD 플레이어는 CD에서 동작하는 계약을 정의한다. 그리고 어떤 CD 플레이어 구현과 CD 그 자체 사이에 최소한의 커플링을 유지한다.

하지만 여전히 CompactDisc의 구현이 필요하다. 실제로 여러 개의 CompactDisc 구현을 가질 수 있다. 이 경우에 다음 코드에 나오는 SgtPeppers 클래스를 가지고 시작한다.

**코드 2.2 @Component로 애너테이트된 SgtPeppers의 CompactDisc 구현**

```
package soundsystem;
import org.springframework.stereotype.Component;

@Component
public class SgtPeppers implements CompactDisc {

  private String title = "Sgt. Pepper's Lonely Hearts Club Band";
  private String artist = "The Beatles";

  public void play() {
    System.out.println("Playing " + title + " by " + artist);
  }

}
```

CompactDisc 인터페이스를 사용할 때, SgtPeppers 스펙은 중요하지 않다. SgtPeppers가 @Component를 가지고 애너테이트됨을 주목해야 한다. 간단한 애너테이션은 클래스가 컴포넌트 클래스임을 나타내고, 클래스를 빈으로 만들어야 함을 스프링에 단서로 제공한다. SgtPeppers 빈을 명시적으로 설정할 필요는 없다.

이 클래스는 @Component로 애너테이트되었으므로 스프링이 빈으로 만든다. 하지만 컴포넌트 스캐닝은 기본적으로 켜 있지는 않다. 스프링을 통해 @Component으로 애너테이트된 클래스를 찾기 위해 명시적인 설정을 작성할 필요가 있다. 다음 코드에서 이를 가능케 할 최소한의 설정을 보여 준다.

**코드 2.3 컴포넌트 스캐닝을 가능케 하는 @ComponentScan**

```
package soundsystem;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan
public class CDPlayerConfig {
}
```

CDPlayerConfig 클래스는 자바로 스프링 와이어링 스펙을 정의한다. 자바 기반의 스프링 설정에 대해서는 2.3절에서 더 살펴본다. 그러나 지금은 CDPlayerConfig는 명시적으로 어떤 빈도 정의하지 않는다. 대신 스프링으로 컴포넌트 스캐닝을 가능케 하기 위해서 @ComponentScan으로 애너테이트된다.

더 이상의 설정 없이, @ComponentScan은 설정 클래스로서 동일한 클래스를 기본 스캐닝한다. 따라서 CDPlayerConfig는 soundsystem 패키지에 있으므로 스프링은 그 패키지와 하위 패키지를 스캔하고, @Component로 애너테이트된 클래스를 찾는다. CompactDisc 클래스를 찾고, 자동으로 스프링으로 빈을 만든다.

XML 설정을 통해서 컴포넌트 스캐닝을 활성화하려면 그때는 스프링의 컨텍스트 네임스페이스(context namespace)로부터 <context:component-scan> 요소를 사용한다. 다음은 컴포넌트 스캐닝을 활성화하기 위한 최소한의 XML 설정이다.

**코드 2.4 XML로 컴포넌트 스캐닝 활성화하기**

```
<?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.xsd">

  <context:component-scan base-package="soundsystem" />

</beans>
```

비록 XML이 컴포넌트 스캐닝을 가능케 하는 옵션이지만, 이 책에서는 더 선호하는 자바 기반의 설정을 사용하는 데 집중한다. XML이 더 스타일리시하긴 해도 <context:component-scan> 요소에는 @ComponentScan을 사용할 때 애트리뷰트 값과 사용할 애트리뷰트를 미러링(mirroring)하는 하위 요소가 있다.

믿건 믿지 않건, 두 개의 클래스만으로 우리는 이미 무언가를 할 준비가 되었다. 컴포넌트 스캐닝 작업을 테스트하기 위해서 스프링 애플리케이션 컨텍스트 생성과 CompactDisc 빈이 생성되었는지를 알기 위한 간단한 Junit 테스트를 만든다. 다음 코드의 CDPlayerTest가 바로 그 테스트다.

코드 2.5 **컴포넌트 스캐닝에 의해 발견된 CompactDisc 테스트하기**

```
package soundsystem;

import static org.junit.Assert.*;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(classes=CDPlayerConfig.class)
public class CDPlayerTest {

  @Autowired
  private CompactDisc cd;

  @Test
  public void cdShouldNotBeNull() {
    assertNotNull(cd);
  }

}
```

CDPlayerTest는 테스트 시작 시 자동으로 생성되는 스프링 애플리케이션 컨텍스트를 가지는 스프링 SpringJUnit4ClassRunner을 이용한다. @ContextConfiguration 애너테이션은 CDPlayer Config 클래스를 통해서 설정을 로드한다. 설정 클래스는 @ComponentScan을 가지므로 결과 애플리케이션 컨텍스트는 CompactDisc 빈을 포함한다.

증명을 위해 테스트 클래스는 CompactDisc 빈을 테스트로 주입하는 @Autowired로 애너테이트된 CompactDisc 타입 프로퍼티를 가진다(@Autowired에 대해선 더 자세히 이야기한다). 결국, 간단한 테스트 메소드는 cd 프로퍼티가 널(null)이 아님을 확인한다.[1] 널이 아니라는 것은 스프링이 CompactDisc 클래스를 찾아서 스프링 애플리케이션 컨텍스트에서 빈으로 자동 생성하고, 테스트로 주입한다.

테스트는 훌륭하게 통과할 것이다(또는 테스트 수행 결과에서 문제 없음을 알리는 녹색을 가진다). 첫 번째 간단한 컴포넌트 스캐닝 연습은 성공이다. 비록 하나의 빈을 만들기 위해서 컴포넌트 스캐닝을 사용한 경우라도, 동일한 작은 설정량으로 다수의 빈을 생성하고 커버할 수 있다. @Component로 애너테이트된 Soundsystem 동일 패키지와 그 하위의 클래스는 빈으로 생성된다.

1 assert는 검증하다로 해석 가능

@ComponentScan한 줄로 수많은 빈을 자동으로 생성하게 하는 것은 좋은 거래다. @ComponentScan과 @Component를 더 깊게 살펴보고, 컴포넌트 스캐닝으로 다른 무엇을 할 수 있는지를 살펴보자.

### 2.2.2 컴포넌트 스캔된 빈 명명하기

스프링 애플리케이션 컨텍스트에서 모든 빈은 ID가 주어진다. 이전 예제에서 확실하게 제시되지 않은 이유는 SgtPeppers 빈에 ID가 명시적으로 주어지지 않아도 클래스 명으로부터 유추되어 할당되었기 때문이다. 특히 구체적으로 빈은 클래스 명의 첫 글자를 소문자로 바꾼 sgtPepper ID를 가진다. 만약 빈에 다른 ID를 주고 싶다면 해야 할 일은 @Component 애너테이션에 원하는 ID를 값(value)으로 넣어 주는 것이 전부다. 예를 들면, lonelyHeartsClub으로 빈을 구별하고자 한다면 @Component를 가지고 SgtPeppers 클래스에 아래와 같이 적용한다.

```
@Component("lonelyHeartsClub")
public class SgtPeppers implements CompactDisc {
  ...
}
```

빈에 이름을 부여하는 다른 방법에서는 @Component 애너테이션을 더 이상 사용하지 않는다. 대신 빈 ID를 제공하기 위해서 자바 종속객체 주입(DI, Dependency Injection) 스펙(JSR-330)에 정의되어 있는 @Named 애너테이션을 사용한다.

```
package soundsystem;
import javax.inject.Named;

@Named("lonelyHeartsClub")
public class SgtPeppers implements CompactDisc {
  ...
}
```

스프링은 @Component의 대안으로 @Named 애너테이션을 지원한다. 미묘한 차이가 있지만, 대부분의 경우 호환 가능하다.

이미 말했듯이 이 책에서는 @Component 애너테이션을 아주 선호하며, @Named보다는 @Component가 더 많이 사용된다. 따라서 더 이상 @Named를 사용하지 않고, 예제에서도 사용하지 않는다.

### 2.2.3 컴포넌트 스캐닝을 위한 베이스 패키지 세팅

지금까지 애트리뷰트 없이 @ComponentScan을 사용하였다. 컴포넌트 검색을 위한 베이스 패키지로서 설정 클래스 패키지가 기본이라는 의미다. 그렇지만 다른 패키지를 원한다면? 또는 다중 베이스 패키지를 스캔해야 한다면?

베이스 패키지를 명시적으로 세팅해야 하는 한 가지 공통 이유는 애플리케이션 코드와 분리하여 설정 코드를 패키지 안에서 보관할 수 있기 때문이다. 그 경우 기본 베이스 패키지를 이용하는 것은 불가능하다.

다른 베이스 패키지를 지정하는 방법은 @ComponentScan의 값 속성에 기술하는 것뿐이다.

```
@Configuration
@ComponentScan("soundsystem")
public class CDPlayerConfig {}
```

또는 베이스 패키지를 설정하는 것을 보다 명확히 하고 싶으면 basePackages 애트리뷰트를 사용한다.

```
@Configuration
@ComponentScan(basePackages="soundsystem")
public class CDPlayerConfig {}
```

아마 basePackages가 복수형이라는 걸 눈치 챘을 것이다. 여러 개의 베이스 패키지를 지정할 수 있는지 궁금할 텐데, 할 수 있다. 패키지 배열에 대해 basePackages가 스캔되도록 설정하기만 하면 된다.

```
@Configuration
@ComponentScan(basePackages={"soundsystem", "video"})
public class CDPlayerConfig {}
```

여기서 보이는 것과 같은 베이스 페이지 설정은 String 값으로 표시된다. 이는 충분하지만, 타입 세이프하지는 않다. 패키지 이름을 변경한다면 작성했던 베이스 패키지가 잘못될 수 있다(변경 전의 이름이 써 있을 경우 모르고 넘어갈 수 있다).

패키지를 간단한 String 값으로 지정하지 않고, @ComponentScan에서 옵션으로 제공하는 패키지 내의 클래스나 인터페이스를 사용할 수 있다.

```
@Configuration
@ComponentScan(basePackageClasses={CDPlayer.class, DVDPlayer.class})
public class CDPlayerConfig {}
```

basePackages 애트리뷰트는 basePackageClasses로 교체된다. String 이름을 사용하여 패키지를 지정하는 대신, basePackageClasses에 클래스 배열로 지정한다. 클래스가 어떤 패키지 안에 있든 컴포넌트 스캐닝을 위한 베이스 패키지로서 사용된다.

비록 이 책에서는 basePackageClasses에 컴포넌트 클래스를 지정했지만, 스캔될 패키지 안의 비어 있는 마커 인터페이스 생성을 고려해야 한다. 마커 인터페이스를 사용하면, 리팩토링이 쉽도록 인터페이스에 대한 레퍼런스만 가지게 되고, 실제 애플리케이션 코드(컴포넌트 스캔하고자 했던 클래스의 밖으로 옮겨질 수 있는)에 대한 레퍼런스를 사용하지 않을 수 있다.

SgtPeppers 빈처럼 애플리케이션의 모든 객체가 혼자이고 의존성이 없다면, 컴포넌트 스캔으로 충분하다. 그러나 많은 객체는 일을 처리하기 위해 다른 객체에 의존관계를 가진다. 의존성을 가지고 컴포넌트 스캔된 빈을 묶기 위한 방법이 필요하다. 이를 위해서 자동 스프링 설정의 다른 측면인 오토와이어링이 필요하다.

### 2.2.4 오토와이어링되는 빈의 애너테이션

오토와이어링은 스프링이 빈의 요구 사항과 매칭되는 애플리케이션 컨텍스트상에서 다른 빈을 찾아 빈 간의 의존성을 자동으로 만족시키도록 하는 수단이다. 오토와이어링 수행을 하도록 지정하기 위해서는 다음 스프링의 @Autowired 애너테이션을 사용한다.

예를 들면, 다음 코드의 CDPlayer 클래스를 보자. 생성자는 @Autowired가 걸려 있는데, 이 애너테이션은 스프링이 CDPlayer 빈을 생성할 때, 생성자를 통해서 인스턴스화하고 CompactDisc에 대입 가능한 빈을 전달해 준다.

**코드 2.6 오토와이어링을 사용하여 CompactDisc를 CDPlayer 빈으로 주입하기**

```
package soundsystem;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class CDPlayer implements MediaPlayer {
  private CompactDisc cd;

  @Autowired
```