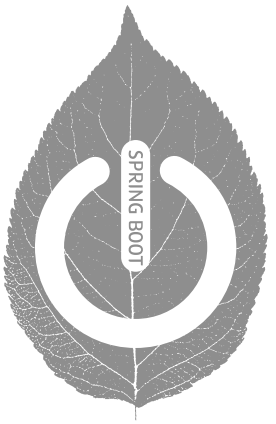



# 스프링 부트로 배우는 자바 웹 개발

# 스프링 부트로 배우는 자바 웹 개발



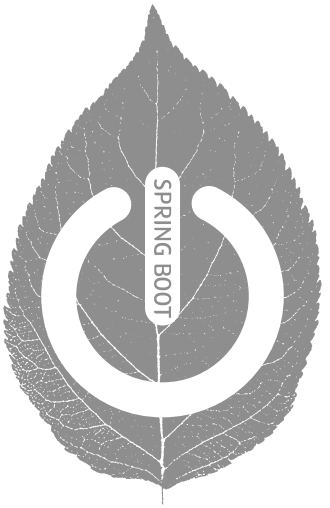
**1쇄 발행** 2018년 6월 29일

**지은이** 윤석진
**펴낸이** 장성두
**펴낸곳** 주식회사 제이펍

**출판신고** 2009년 11월 10일 제406-2009-000087호
**주소** 경기도 파주시 회동길 159 3층 3-B호
**전화** 070-8201-9010 / **팩스** 02-6280-0405
**홈페이지** www.jpub.kr / **원고투고** jeipub@gmail.com
**독자문의** readers.jpub@gmail.com / **교재문의** jeipubmarketer@gmail.com

**편집부** 이종무, 황혜나, 최병찬, 이 슬, 이주원 / **소통·기획팀** 민지환 / **회계팀** 김유미
**교정·교열** 배규호 / **본문디자인** 북아이 / **표지디자인** 미디어픽스
**용지** 신승지류유통 / **인쇄** 해외정판사 / **제본** 광우제책사

**ISBN** 979-11-88621-27-9 (93000)
**값** 27,000원




# 스프링 부트로 배우는 자바 웹 개발

**윤석진** 지음

Jpub 제이펍

**※ 드리는 말씀**

- 이 책에 기재된 내용을 기반으로 한 운용 결과에 대해 저자, 소프트웨어 개발자 및 제공자, 제이펍 출판사는 일체의 책임을 지지 않으므로 양해 바랍니다.
- 이 책에 기재한 회사명 및 제품명은 각 회사의 등록 상표(또는 상표)이며, 본문 중에는 ™, ©, ® 등의 기호를 생략하고 있습니다.
- 이 책에서 설명하고 있는 실제 제품 버전은 독자의 학습 시점에 따라 책의 버전과 다를 수 있습니다.
- 책의 목적이 스프링 부트 기술 자체보다 웹 개발에 초점이 맞춰져 있어서 스프링 부트 최신 버전 대신 안정적인 1.5 버전 기준으로 설명하였으며, 스프링 부트 2.0에서도 잘 작동합니다.
- 책 내용과 관련된 문의사항은 지은이나 출판사로 연락 주시기 바랍니다.
  - 지은이: https://bit.ly/2kW4Hfj
  - 출판사: readers.jpub@gmail.com

# 차례


처음 스프링을 공부하기 시작했을 때 XML 설정이 생소해서 익숙해지는 데 어려움이 많았고, hello만 있는 웹 페이지 하나를 보기 위해 해야 하는 설정들이 힘들어서 포기한 적도 많았다. 실제로 업무를 하려면 API들을 많이 사용해 봐야 하는데, 앞부분 설정 때문에 정작 봐야 할 뒷부분은 공부하지 못했던 기억이 있다. 이후에 설정이 간소화된 스프링 부트를 써보면서 스프링 부트를 기반으로 교육자료를 만들면 시작하는 사람들이 좀 더 편하게 공부할 수 있을 것 같아 이 책을 쓰게 되었다.

사실, 이 책의 시작은 3년 전으로 거슬러 올라간다. 직장을 그만두고 호기롭게 시작했는데, 메시징, NoSQL, MSA 등을 한 권에 담기에는 주제가 너무나 광범위했고, 새로운 기술을 책에 담고자 하는 마음 때문에 시간이 지날수록 오히려 집필해야 하는 양이 늘어나는 형국이 되었다. 그러다가 작년 즈음에 다시 목차를 손보고 내용을 정리해서 지금의 모습이 되었다.

## 고마운 분들

이 책을 쓰도록 추천해 준 현지환 님, 예제까지 꼼꼼히 테스트해 준《앵귤러 첫걸음》의 저자 조우진 님, 1장부터 5장까지 매끄러운 문장을 위해 도움 준 박현우(lqez) 님, 11장 캐시 장을 리뷰해 준 응봉동 입개발자 강대명 님, 3·6·12장을 리뷰해 준 라인플러스의 핀테크 개발자 장진달 님, 데이터 관련 장에서 많은 의견을 준 우아한형제의 이경원 님에게 깊은 감사를 드린다.

## 관련 기술 현황

이 책에서 사용하는 스프링 부트의 버전은 1.5.8이다. 스프링 부트는 현재 2.1.0 버전까지 릴리스되었다. 클라우드 사용이 늘어나면서 별도의 서버 설치 없이 JAR로 실행할 수 있고, 설정들이 자동화되어 있는 장점들 덕분에 사용처가 늘고 있는 추세다.

## 대상 독자

이 책은 자바 웹서비스 개발을 시작하는 사람들을 위한 입문서다. 입문서로서 자바 웹 개발의 시작이라고 할 수 있는 서블릿부터 Spring Data JPA, REST API, Actuator를 활용한 모니터링, 클라우드 서비스를 이용한 배포까지 웹 서비스 개발에 필요한 전반적인 내용을 다룬다. 따라서 자바 입문서를 읽었거나 C 언어를 학습한 이후 자바 기반의 웹 서비스 개발을 배우려는 사람, 스프링 부트를 차근차근 배우고 싶은 사람, 서블릿부터 스프링 부트까지 체계적으로 복습하고 싶은 사람이 보면 좋겠다.

## 책의 구성

- **1장 개발 환경의 변화와 자바** 개요로서, 기술 설명 이전에 전반적인 웹 서비스 아키텍처 변화의 흐름과 자바 웹 개발의 기본 파일 포맷인 WAR 파일과 클래스 로더, 웹 애플리케이션 컨테이너에 관해 다룬다.
- **2장 서블릿** 기본적인 HTTP 요청/응답 방법과 J2EE 디자인 패턴을 다룬다
- **3장 스프링 프레임워크** 스프링 프레임워크의 의존성 주입 방법, Java Config 방식으로 빈을 등록하는 방법, 스프링 MVC에 관해서 다룬다.
- **4장 스프링 부트 웹 개발** 스프링 부트를 이용해 jQuery와 같은 프론트 라이브러리를 추가하고, thymeleaf를 이용해 페이지 레이아웃을 구성하고, 정적 자원 관리 방법을 다룬다.
- **5장 REST API 서버 만들기** 스프링 부트를 이용해 REST API 만드는 방법을 다룬다.
- **6장 스프링 부트와 데이터** 스프링 부트에서 데이터를 처리하기 위해 필요한 설정, JPA 및 MyBatis 사용 방법을 다룬다.
- **7장 커스텀 스프링 부트 스타터** 스프링 부트의 내부 구조를 이해하고 간단하게나마 스프링 부트 스타터를 직접 만들어 보는 방법을 다룬다.

- **8장 예외 처리 및 테스트** 자바의 기본 예외 처리 방법에 대한 기본 개념과 스프링 부트를 이용한 테스트 방법을 다룬다.
- **9장 배포** 클라우드 서비스 중 하나인 헤로쿠와 Github의 소스 저장소를 연동해서 스프링 부트로 만든 서비스의 배포 방법을 다룬다.
- **10장 모니터링** 액추에이터와 JConsole을 이용해 모니터링하는 방법을 다룬다.
- **11장 캐시** 캐시에 대한 기본 개념과 Ehcache 사용 방법을 다룬다.
- **12장 회원 관리** 책 내용 전체를 예제로 복습하는 장인데, 웹 서비스 개발에 필요한 공통적 기능인 회원가입과 로그인을 다룬다.

## 소스 코드 다운로드

1~11장까지의 내용은 다음 경로에서 다운로드할 수 있다.

URL https://github.com/thecodinglive/JPub-JavaWebService

12장 마지막 예제는 다음 경로에서 다운로드할 수 있다.

URL https://github.com/thecodinglive/memberApp

## 독자 Q&A

책 내용과 관련된 문의는 다음의 페이지를 이용해 주기 바란다.

URL https://bit.ly/2kW4Hfj


### 공민서(엔트로피랩)

자바 웹 개발의 역사 소개와 실습을 함께 진행한 책으로, 느낌이 굉장히 좋았습니다. 자바 웹 개발의 기초부터 배포 및 모니터링, 캐시, 테스트 등을 한 권에 담아 좋았고, 잘 모르고 있던 프레임워크의 역사와 장단점을 짚어 준 부분이 도움이 되었습니다. 자바 웹 개발에 저보다 훨씬 능숙한 친구에게 책 내용을 설명하니 흥미를 보이면서 필수적인 내용이 잘 담겨 있는 것 같다고 합니다. 다음에는 이 책과 유사한 형태로 파이썬 장고(Django)를 다루는 책이 나왔으면 좋겠습니다.

### 구민정(SK주식회사)

웹과 관련하여 자주 들어 봤지만 잘 활용하기 어려웠던 기술들을 예제 코드와 함께 설명해 줘서 무척 좋은 책이라고 생각합니다. 다양한 웹 기술 세트를 알려 주고 간단한 예제 코드로 활용해 볼 수 있는 부분이 좋았고, 한 장씩 넘어갈수록 코드가 점차 업그레이드되는 즐거움도 느낄 수 있었습니다. 넓은 범위의 내용을 담고 있어서 초보자보다는 웹 개발을 이미 경험해 보신 분들께 더 적합할 것 같습니다. 더 좋은 코드를 작성하고 싶다고 생각하는 웹 개발자라면 이 책을 읽어 보시길 추천합니다. 다만, 일부 오탈자, 용어 표기, 코드 줄 맞춤 등은 보완되어 출간되기를 바랍니다.

### 김정헌(BTC)

시대적 흐름에 맞춰 자바 웹 개발 방법도 많이 바뀌고 있는 것 같습니다. 예전의 스프링에서는 복잡했던 것들이 스프링 부트에서는 쉽게 할 수 있는 것을 보고 자바도 많이 변했다는 생각을 했습니다. 이 책은 기초에서부터 배포, 모니터링까지 자바 웹 개발에 대한 거의 모든 것을 다루고 있습니다. 최신 자바 웹 개발 트렌드가 궁금하다면 한번 읽어 보시기 바랍니다. 후반부에 좀 더 큰 규모의 프로젝트가 포함되었더라면 더 좋았을 듯합니다. 영문 스펠링이 정확하지 않은 것들이 제법 보였는데, 출간 전에 수정되기를 바랍니다.

**김준호(티맥스 클라우드)**

스프링은 우리나라에서 높은 점유율을 가진 엔터프라이즈 프레임워크입니다. 스프링 책은 흔히 어떻게 사용해야 한다(How to use)에 집중하지만, 이 책은 거기에 더하여 어떻게 구성되어 동작하는가(How it works)에 대한 내용도 다루고 있습니다. 서블릿에 관한 설명부터 시작하여 왜 스프링이 필요하게 되었는지, 각 구성 요소가 전체 프로젝트 구조에서 어떤 역할을 하는지 조망할 수 있습니다. 스프링을 사용한다면 읽어 봄직한 책입니다.

**김진영(프리랜서)**

자바 개발자로서 목차가 상당히 마음에 든 책이었습니다. 그리고 단순히 이런 기술이 있고 사용법 위주의 책이라 생각했는데, 중간중간 스프링 프레임워크의 원리와 관련 예시를 들어 주는 부분도 마음에 들었습니다. 그리고 이 책은 기존에 스프링 프레임워크와 스프링 부트 설정 기반 지식을 보유한 분들이 보면 좋을 듯합니다. 프레임워크의 핵심 특징들을 기술하고 있지만, 업무에 사용할 수 있는 다양한 내용이 한 권에 포함되어 있어서 레퍼런스로 활용하면 좋을 것 같습니다. 원리의 이해보다는 실무에서 사용할 기술에 대한 참고사항들을 훑어보고 사용하고자 하는 분들께 추천합니다.

**송재욱(카카오)**

스프링 웹 개발에 대한 전반적인 내용을 다룬 책입니다. 한정된 분량 안에서 이러한 개념과 구조를 설명하다 보니 깊이보다는 단편적인 실습을 통한 개괄적 이해에 초점이 맞춰져 있습니다. 따라서 자바 웹 개발을 시작하려는 분들이 전체 그림을 그려보는 용도로 보기에 적당한 입문서라 생각합니다. 일부 오탈자와 코드 컨벤션은 출간 전에 수정되었으면 하는 바람입니다.

**이지현**

막연하게 사용하고 있던 내용을 정확히 알고 사용할 수 있도록 도와주는 책이었습니다!


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

CHAPTER

# 1

# 개발 환경의 변화와 자바

# 1.1 인프라와 스프링 프레임워크의 변화

예전에 인형이나 장난감이 어린이들만의 것이었다면 최근에는 어른들이 수집하는 피규어나 건담 프라모델이 어른들만의 장난감이 되고 있다. 프라모델은 완성품을 만들기 위한 부품들을 제공하는데 최근의 웹 개발 트렌드도 프라모델과 다르지 않다. 프라모델 부품들을 조립하는 것처럼 적은 인원으로 빠른 시간에 서비스를 완성하기 위해서 **오픈 소스**(open source)들을 조합해서 서비스를 출시하는 경우가 일반화되고 있다.

사람이 땅을 딛지 않고 설 수 없듯이 많은 오픈 소스가 활성화되기까지는 개인 컴퓨터의 부품에서부터 서비스를 지탱하는 인프라까지 많은 변화가 필요하다. 초기에는 웹 서비스에 대한 요구사항도 높지 않았고 사용자도 많지 않았으므로 예전에 웹을 접했거나 다른 분야의 개발을 했던 사람들이 본다면 혼란스럽고 다양한 개발 환경이 큰 부담으로 다가올 것이다. 그래서 이번 장에서는 배경 지식을 습득하고 분위기를 익히는 차원에서 인프라의 변화와 이에 따른 자바 기술의 변화에 관해서 다루겠다.

## 1.1.1 아키텍처의 변화

초기에는 일반 사용자를 위한 B2C(Business-to-Customer) 사업 모델이 많지 않았다. 대부분의 광고도 신문이나 방송사와 같은 매체에서 진행하는 것이 대부분이었고, 모두가 사용하는 인터넷보다는 회사 내에서 혹은 특정 인원들만 인트라넷으로 사용해 왔다. 그러나 웹이 활성화되고 닷컴 열풍이 불자 각종 커뮤니티 및 온라인 쇼핑몰 등 B2C로의 트렌드 변화가 인프라의 아키텍처를 바꾸었다.

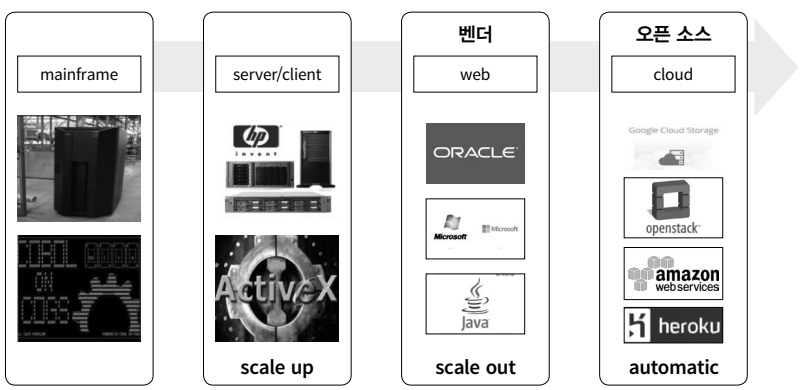



그림 1-1 **인프라의 변화**

그림 1-1에서 보듯이 오른쪽으로 갈수록 점점 더 많은 사용자를 대상으로 하는 시스템이다. 첫 번째 단계는 서버/클라이언트 개념이 등장하면서 사용자를 위한 GUI(Graphical User Interface) 개발이 본격화되고, 델파이를 이용한 업무 프로그램과 ActiveX를 이용해 웹에서도 애플리케이션을 실행하는 방식들이 자주 사용되었다. 그리고 두 번째 단계는 서버에서 파일로 데이터를 저장하거나 자체 개발을 통해 데이터를 저장하고 소켓 통신을 통해 클라이언트와 통신하는 방식이 주류를 이뤘다. 세 번째 단계는 전자상거래 활성화와 웹에 대한 선풍적인 인기, 상용 관계형 데이터베이스의 흥행 등으로 시스템을 3티어(tier)로 구성하는 방식이 일반화되었다. 기존에는 서버와 클라이언트가 직접 통신을 했다면, 이 시기에 데이터는 데이터베이스에 저장되어 있고 JSP, PHP, ASP와 같은 서버 언어들로 데이터를 전달하고 클라이언트에서 전달받은 데이터를 처리하는 방식이 자리 잡았다.

인터넷 데이터 센터(IDC, Internet Data Center)에 물리장비를 이용해서 직접 설치하고 운영하는 방식은 성능이 좋지만, 게임이나 SNS 서비스처럼 사용자 수의 변동 폭이 매우 큰 서비스들을 개발할 경우에는 일정한 서비스의 품질을 유지하기가 매우 까다롭다는 단점이 있다. 그래서 최근에는 서비스를 시작할 때 클라우드(Cloud) 서비스를 활용하기도 한다. 하지만 클라우드 서비스는 전화요금제를 선택하는 것처럼 각 사양에 맞춰 이용해야만 하고, 일부만 임의로 증설하거나 변경하는 게 어렵고, 클라우드 업체의 기술에 종속되는 문제가 단점으로 작용하기도 한다.

**표 1-1 인프라 변화에 따른 기술의 변화**

| | 메인프레임 | 서버/클라이언트 | 웹 | 클라우드 |
|---|---|---|---|---|
| 인터페이스 | 터미널 | 웹 브라우저/GUI | 웹 서버(iis, apache, tomcat) | 인스턴스 또는 컨테이너 |
| 주 언어 | 코볼, 포트란 | 델파이, C++, 펄 | PHP, JSP, ASP | 파이썬 |
| 목적 | B2B | B2C | B2C | B2C |
| 플랫폼 지원 수준 | 단일 접속 | 클라이언트 설치를 통한 | 웹 서버에 의한 접속 처리 | 물리 설치 없이 인스턴스 형태로 |
| | 순차 배치 처리 | 접속 지원 | 별도의 데이터베이스 서버 활용 빈도수가 높아짐 | 자유롭게 확장 가능 |

표 1-1은 인프라 변화에 따른 기술의 변화를 단계별로 도식화한 것이다. 그림 1-1이 시간순으로 배열되어 있었기에 오른쪽에 비해 왼쪽으로 갈수록 부족하다고 생각할 수 있다. 하지만 어떤 업무에 적용하느냐에 따라 다르다. 게임이나 장비 제어와 같은 특수 분야를 제외하고 회사에서 하는 개발은 크게 두 가지로 구분할 수 있다. 첫 번째가 엔터프라이즈, 두 번째가 웹 서비스다. 엔터프라이즈란, 말 그대로 금융 업무, 날씨 같은 부분을 차지한다. 이 분야들은 보안과 안정성에

민감하고 반드시 트랜잭션이 보장되어야 한다. 그래서 해당 업무를 수행하는 회사들은 메인프레임(mainframe)을 선호하는 경우도 많다. 또 통신사의 경우에는 데이터를 주고 받은 후에 특정 장비나 단말기에 표출하는 경우가 많으므로 별도의 UI(User Interface)가 필요 없고, 서버/클라이언트 방식으로 소켓을 통해 데이터를 주고 받기도 한다.

## 1.1.2 스프링 프레임워크의 변화

인프라의 변화에 맞물려서 스프링 프레임워크(Spring Framework)[1]도 많은 변화가 있었다. 초기에는 오라클(Oracle)이나 IBM과 같은 업체들이 제공하는 솔루션을 주로 사용했다. 그래서 웹로직(WebLogic), 웹스피어(WebSphere)와 같은 서버들을 많이 사용했고, 개발할 때는 EJB(Enterprise JavaBean)를 이용해서 개발을 진행했다. 그리고 점차 톰캣 서버가 버전업되면서 성능이 좋아지고, 스프링 프레임워크도 2.5 버전 이후로 안정화되면서 웹로직, 웹스피어 서버에 스프링 기반의 프로젝트가 증가하였다. 또한, 전자정부 프레임워크에서도 기반 기술로 스프링 프레임워크를 채택하면서 스프링 프레임워크와 톰캣 조합이 표준처럼 쓰이게 되었다.

최근에는 스타트업들이 대거 등장하면서 작은 규모로 빠르게 서비스를 런칭하는 경우가 많아졌고, 클라우드의 사용 빈도가 더 늘어났다. 특히 일반 사용자를 대상으로 하는 B2C 서비스의 경우에는 PaaS(Platform as a Service)로 서버를 물리적으로 증설하지 않고 인스턴스를 추가로 사용함에 따라 JDK와 톰캣을 설치하고, 스프링 관련 XML을 설정하는 일련의 작업들을 간소화할 방법이 필요해졌다. 그래서 스타트업들은 루비온레일스(RubyOnRails)나 장고(Django)를 이용해 빠르게 개발하는 것을 선호하게 되었다. 해당 기술들은 톰캣과 같은 별도의 애플리케이션 서버 설치 없이 웹 서버만으로 클라우드 서비스를 이용해 쉽게 확장할 수 있고, 개발도 스켈레톤[2]을 제공하므로 좀 더 빠르게 시작할 수 있다.

일례로 "Hello World"를 출력하는 데 소요되는 시간으로 비교할 수 있다. 자동차로 비유하면 스프링 프레임워크는 출력은 좋지만 시동을 걸고 최고속도에 도달할 때까지 해야 할 일들이 너무 많은 차다. 스프링 프레임워크의 초기 콘셉트는 가벼움이었다. 스프링 프레임워크를 사용하기 이전에는 EJB를 사용했다. EJB는 세션빈(SessionBean), 엔티티빈(EntityBean)과 같은 요소들을 설정해야 하고, 스프링에 비해서 테스트하기 어렵고 무거워서 로드 존슨(Rod Johnson)은 《J2EE 설계와 개발(expert one-on one)》이라는 책을 쓰면서 스프링 프레임워크를 만들었는데, 시간이 지나

---

1 스프링 프레임워크에 대한 내용은 3장에서 다룬다.

2 스켈레톤은 뼈대라는 단어 뜻처럼 프로젝트의 골격이 되는 코드와 디렉토리 구조, 라이브러리 설정 등을 말한다.

고 나니 이제 스프링 프레임워크가 다른 언어의 프레임워크에 비해 무겁고 설정할 것이 많은 프레임워크가 되었다. 스프링 진영에서도 이런 문제를 해결하기 위해 스프링 부트(Spring Boot)를 만들었다.

스프링 부트는 설정 자동화(AutoConfigure)를 이용해서 스프링 MVC 모듈의 DispatcherServlet 설정, JDBC DataSource 설정 등 웹 개발을 하는 데 필요한 인프라성 코드들을 제공해 줌으로써 복잡한 XML 설정을 하지 않아도 개발을 시작할 수 있다. 또한, 실행 시에도 임베디드 톰캣(Embedded Tomcat)을 이용해 main 메서드로 실행할 수 있다. 그리고 클라우드 환경에서도 별도의 작업 없이 스프링 부트를 이용하면 시간을 많이 단축할 수 있다.

처음에 스프링 MVC를 사용할 때는 web.xml, DispatcherServlet 설정 등에 시간을 보내고 설정 오류를 고치느라 정작 스프링이 제공하는 기능들은 해보지도 못하는 경우가 많다. 그래서 이 책에서는 기본이 되는 서블릿(Servlet)부터 스프링 코어를 다루고, REST API나 데이터베이스 관련 내용들은 스프링 부트를 이용해 학습할 수 있도록 구성했다.

# 1.2 웹 애플리케이션 컨테이너

**웹 애플리케이션 컨테이너**(Web Application Container)란, 웹 애플리케이션이 배포되는 공간을 뜻한다. 일반적으로 HTML과 같은 정적 파일들을 전달해 주는 역할을 하는 서버를 **웹 서버**라고 하고, PHP, JSP, ASP와 같은 언어들을 사용해서 동적인 페이지들을 생성 가능한 서버를 웹 애플리케이션 서버(Web Application Server) 또는 자바 계열에서는 웹 애플리케이션 컨테이너라고 하며, 일반적으로 WAS로 줄여서 부르기도 한다. WAS가 어떻게 웹 애플리케이션을 인식하고 동작시키는지 알기 위해서는 클래스 로더를 알아야 한다.

## 1.2.1 자바 개발을 위해 꼭 필요한 클래스 로더

자바의 특징을 설명하는 문구 중 빠지지 않고 들어가는 말이 있다. "한번 작성하면 플랫폼에 상관없이 쓸 수 있다(Write once, run anywhere)"는 것을 영업적으로 강조한 표현인데, 이 특징을 실행할 수 있게 한 기술이 바로 **클래스 로더**(class loader)다. 자바 코드를 작성한 후 컴파일하면 해당 코드는 JVM(Java Virtual Machine)에서 실행 가능한 상태가 된다. 이때 JVM이 클래스를 실행하기 위해서는 클래스를 로딩하는 과정이 필요하다. 그 과정을 수행해 주는 역할을 하는 것이

클래스 로더다.

우리는 종종 새로운 프레임워크를 테스트해 볼 때나 다른 사람이 만든 프로젝트를 로컬에서 실행할 때 ClassNotFoundException와 같은 에러를 만나게 된다. 해당 에러는 클래스 로더가 추가된 라이브러리 또는 클래스를 인식하지 못해서 발생하는 오류다. 클래스패스(classpath)에 해당 모듈 또는 라이브러리를 추가하면 오류는 사라진다. 클래스패스에 추가되면 클래스 로더는 식별자로 클래스파일 메타 정보 중 첫 번째 시작 필드를 이용해 클래스를 로딩한다. 다음 절에서 클래스패스를 추가하는 방법과 클래스 로더가 동작하는 방법을 살펴보자.

#### 1.2.1.1 클래스 로더의 특징

클래스 로더는 네 가지 특징이 있다. 첫 번째, 구조가 계층적이다. 상위 클래스 로더에서 하위 클래스 로더를 갖는 방식이며, 최상위 클래스 로더는 부트스트랩 클래스 로더다. 두 번째, 클래스 로딩을 위임할 수 있다. 세 번째, 클래스 로더는 가시적인 규약이 있다. 가시적인 규약이란, 클래스를 로딩할 때 가능한 범위를 말한다. 자식 클래스 로더는 클래스 로딩 요청 위임을 통해 부모 클래스 로더가 로딩한 클래스를 찾을 수 있지만, 부모 클래스 로더는 자식 클래스 로더가 로딩한 클래스를 알 수 없다. 네 번째, 클래스 언로딩 불가능이다. 즉 클래스 로더로 로딩한 클래스들을 언로딩할 수 없다. 클래스 로더가 로딩한 클래스를 언로딩할 수 없으므로 가비지 컬렉터(garbage collector)가 동작하거나 WAS가 재시작할 때 초기화된다.

#### 1.2.1.2 클래스 로더의 유형

클래스 로더에는 네 가지 유형이 있다. 부트스트랩 클래스 로더(bootstrap class loader), 확장 클래스 로더(extension class loader), 시스템 클래스 로더(system class loader) 그리고 개발자가 만든 사용자 정의 클래스 로더(user-defined class loader)가 있다.

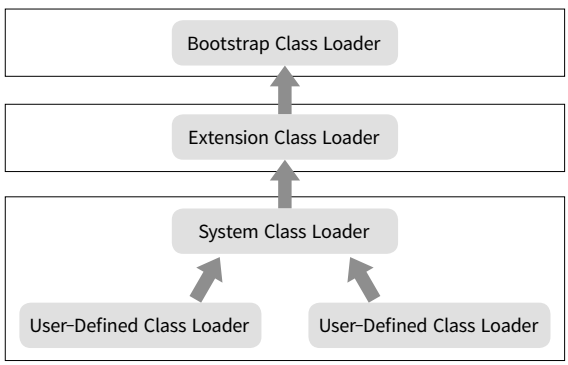



그림 1-2 **클래스 로더 구조**

부트스트랩 클래스 로더는 JVM 런타임 실행을 위해 기반이 되는 파일들을 로드한다. 부트스트랩 로더는 rt.jar 파일과 연관이 있다. 부트스트랩 로더가 로딩이 끝나면 확장 클래스 로더가 자바의 최상위 객체인 Object를 포함한 자바 API를 로드한다(자바 홈 폴더 하위의 ext 폴더 하위에 있는 JAR 파일들과 연관이 있다). 확장 로더의 로드가 끝나면 시스템 클래스 로더가 클래스패스에 포함된 클래스들을 로드한다.

클래스 로더는 이와 같은 계층 관계를 가지고 있다. 사용자는 시스템 클래스 로더가 로드하는 클래스 패스 영역에만 접근할 수 있다. 그래서 로컬에서 외부 라이브러리를 실행할 때 클래스패스를 지정해서 실행하기도 한다. 보통 독립적인 영역이 필요한 WAS의 경우에는 시스템 클래스 로더 하위에 사용자 정의 로더를 만들어서 사용한다. 대부분의 개발 환경 설정 관련 문서에서 톰캣 설치 위치를 CATALINA_HOME으로 지정하는 것은 WAS에서 생성한 클래스 로더를 기준으로 동작하기 위함이다.

## 1.3 WAR 파일의 특성

배포할 때 로컬 실행 프로그램은 JAR로 패키징하고 웹은 WAR로 패키징한다. WAR는 압축 파일에 자바 관련 규약이 포함된 것이다. 바로 WEB-INF 폴더다. 웹 애플리케이션 컨테이너는 WAR 파일의 WEB-INF 폴더를 기준으로 클래스 파일들을 로드한다.

content directory[3]는 WAR는 Web Application Archive의 약자이며, Web Application Resource를 뜻하기도 한다. webapp 또는 web과 같은 이름으로 프로젝트 설정에 따라서 조금씩 다르지만, HTML, 자바스크립트(JavaScript), CSS 또는 HTML 태그를 포함하고 있는 JSP 파일처럼 브라우저에서 보여 줘야 하는 정적 자원을 관리하기 위한 폴더다. 이 폴더는 브라우저상에서 직접 접근할 수 있어서 최근에는 content directory를 WAR 파일의 상위에 두지 않고 WEB-INF 하위에 설정하는 추세다. WAR로 패키징하면 클래스 파일들은 WEB-INF 하위 classes 폴더에 저장된다. 그림 1-3에서는 이해를 돕기 위해서 class가 아니라 자바로 표기했다. libs 폴더에는 JAR 형식의 외부 라이브러리들이 있다. 이 폴더에 있는 JAR 라이브러리들은 사용자 정의 클래스 로더, 웹 애플리케이션 컨테이너의 로더를 통해 클래스패스에 추가된다.

---

**3** 그림 1-3 참조

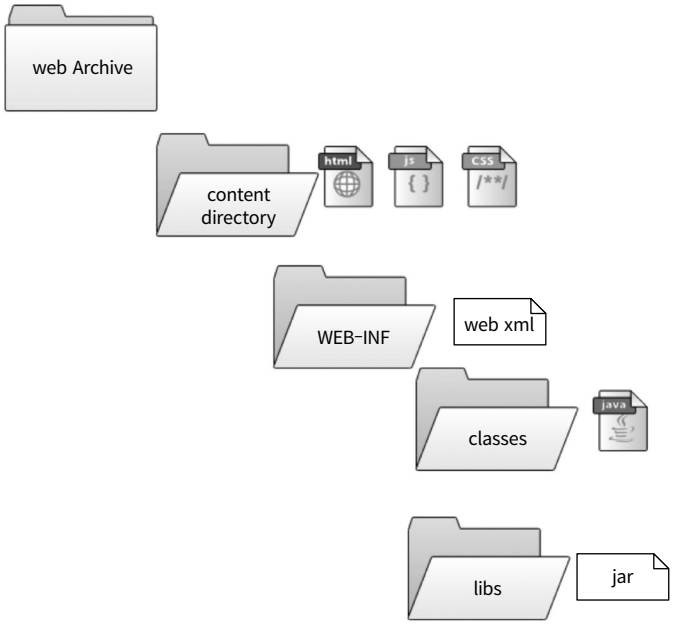



그림 1-3 **WAR 파일의 특성**

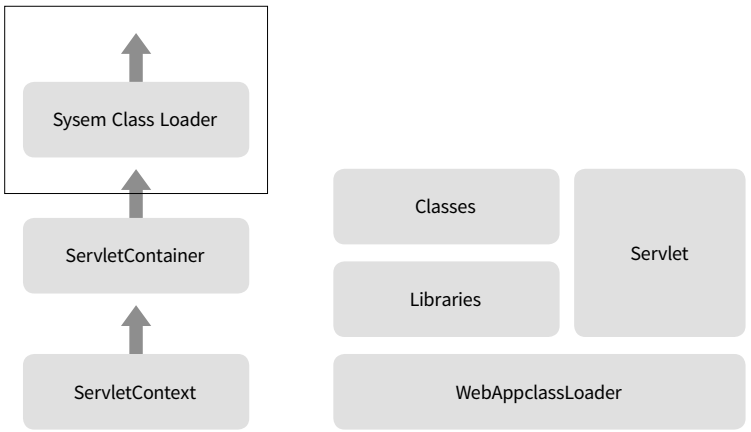



그림 1-4 **서블릿 컨테이너와 클래스 로더**

웹 애플리케이션 클래스 로더는 클래스 로더의 유형 중에서 시스템 클래스 로더 하위에 사용자가 정의한 클래스 로더에 해당한다. 웹 애플리케이션 컨테이너는 웹 애플리케이션 자체 API를 제공하기 위해 컨테이너를 로드하는 클래스 로더와 사용자가 추가한 JSP나 WAR 파일들을 다루기 위한 ServletContext Loader를 사용한다. 컨테이너가 시작되고 콘텍스트가 초기화되면 서블

릿 스펙의 권장 사항에 따라 WEB-INF/classes 파일을 먼저 검색해서 로딩하고, 그 후에 WEB-INF/libs에 있는 JAR 파일들을 로드한다.

**마무리**

이번 장에서는 많은 회사들이 스프링을 쓰게 되기까지의 인프라 변화와 아키텍처 변화에 관해서 알아보고 클래스 로더와 웹 개발 시에 결과물이 되는 형식인 WAR 파일의 구조에 관해서도 살펴봤다.
다음 장에서는 자바 웹 개발의 근간이 되는 서블릿과 HTML 폼에서 데이터를 주고 받는 법, 데이터를 주고 받을 때 GET / POST 방식을 구분하는 법, 세션, 쿠키 등에 대해서 알아보자.

CHAPTER

# 3

# 스프링 프레임워크

# 3.1 빈 + 컨테이너

스프링은 엔터프라이즈 애플리케이션을 개발하기에 적합한 프레임워크로 객체 관리를 해주는 빈 컨테이너 프레임워크다.

공장이나 대형트럭 위 공항 등에서 컨테이너를 본 적이 있을 것이다. 많은 양의 화물을 운반할 때 이 컨테이너를 대형 화물선에 실어서 운반한다. 많은 짐을 화물선에 실을 때 컨테이너를 이용하고 화물선에서 다시 꺼낼 때도 컨테이너를 이용한다. 그리고 화물선은 컨테이너만 관리하면 되고, 컨테이너 내부의 물건들은 컨테이너에서 꺼낼 때 관리하면 된다. 꺼내지 않았다면 물건들이 컨테이너에 적재되었는지 확인하면 된다. 그래서 아무리 많은 물건이 있어도 개별적으로 물건을 관리하지 않고 컨테이너 자체를 관리할 수 있으므로 컨테이너를 실을 수 있는 화물선이라면 안전하게 물건을 싣고 이동할 수 있다.

이 같은 경우는 웹 프로젝트에서도 마찬가지다. 화물선(서버)이 항해하는 동안 파도에 맞춰서 여러 가지 일들을 개별적으로 하는 것은 너무 힘이 든다. 그래서 화물(자바 빈)들을 대신해서 관리해 줄 컨테이너를 찾게 되었고, 화물이라는 단어가 연상시키는 커다란 이미지처럼 그 시작은 EJB(Enterprise Java Beans)였다.

그런데 사람들은 개발에 대한 요구 사항이 점점 복잡해지자 좀 더 경량화되고 간소화된 컨테이너를 선호하게 되었다. 이는 마치 복잡한 서울 시내 한복판에서는 경차가 주차 공간을 확보하거나 주행하기에 용이한 것과 같다. 그래서 사람들은 정말 EJB가 필요한지에 대해서 고민하기 시작했다. 그 시작은 POJO(Plain Old Java Object)다. POJO는 간단히 말해서 일반 자바 클래스다. 당시에는 EJB가 지배적이었으므로 새로운 개념은 아니지만, EJB와 상충되는 방법으로 개발을 진행하고 의견을 모으기 위해서 새로운 용어가 필요했고, 일반 자바 빈이라는 말 대신에 POJO를 사용하기 시작했다. 그리고 로드 존슨은 《J2EE 설계와 개발》이라는 책을 시작으로 웹 애플리케이션 컨테이너와 상관없이 독립적으로 빈의 생명주기를 관리할 수 있는 스프링 프레임워크를 만들게 되었고, 큰 인기를 끌게 된다.

현재 스프링은 Spring.IO(http://spring.io/projects)에 스프링 기반으로 추가된 프로젝트들을 공개하고 있다. 우리에게 익숙한 DataAceess/Web/Core 모듈 외에도 SNS 연동에 유용한 Spring Social, 메시징과 관련된 Spring AMQP, Hbase와 같은 빅데이터 처리를 위한 Spring XD 등 그 영역을 넓혀가고 있다. 이렇게 많은 모듈들을 지속적으로 확장해 나갈 수 있는 것은 Spring

Core Container를 통해 모듈 간의 의존도를 최소화했기 때문이다. 의존도를 낮추는 방법 중 하나는 IoC 패턴을 활용하는 것이다.

## 3.2 IoC 패턴 활용

IoC(Inversion of Control)는 우리나라 말로 '제어의 역전'이라고 부른다. 다른 많은 프로그램들과는 다르게 제어의 역전 패턴이 자바 웹 개발에서 특히 인기를 끄는 이유는 프로그램의 생명주기에 대한 주도권이 웹 애플리케이션 컨테이너에 있기 때문이다. 예를 들어, 게임 엔진을 만드는 경우에는 초기화, 실행, 종료와 같이 애플리케이션의 흐름 제어에 관한 부분을 직접 만든다. 그리고 실행에는 메인 루프(main loop)가 포함되어 프로그램이 종료되기 전까지 계속 실행 안에서 동작한다.

그런데 자바 기반의 웹 프로그램 개발 시에는 여러분들이 브라우저에 naver.com을 입력하면 그 URL이 DNS 서버를 거쳐 서버에 전달되고, 서버에서는 2장에서 배운 doGet, doPost와 같은 메서드들이 파라미터와 함께 호출된다. 이런 상황에서 파일 처리나 데이터베이스 처리를 하는 클래스들의 인스턴스 관리를 해야 하고, 일괄적으로 인스턴스 관리를 하기 위해서 객체의 생성을 관리할 수 있는 도구의 필요성이 대두되었다. 이와 같은 의미로 디자인 패턴의 원칙 중에는 **의존관계 역전 원칙**(Dependency Inversion Principle)이 있다.

의존관계 역전 원칙은 두 가지 내용을 담고 있다. 첫 번째로 하이레벨 모듈은 로우레벨 모듈에 의존해서는 안 되고, 모두 인터페이스에 의존해야 한다. 두 번째로 추상화는 세부 사항에 의존해서는 안 된다. 세부 사항이 추상화에 의존해야 한다는 내용인데, 단순하게 말해서 인터페이스를 활용함으로써 결합도를 낮추게 하자는 뜻이다. 그런데 자바에서는 인터페이스를 사용하더라도 결국 인스턴스화하기 위해서는 반드시 객체 생성에 필요한 코드가 수반되므로 결합도를 완전히 분리해 낼 수 없다. 결국 프로그램이 온전히 동작하기 위해서는 인스턴스화할 수 있는 코드에 대한 의존성을 갖게 된다. 그리고 이를 대신 해결해 주는 것이 바로 **의존성 주입**(DI, Dependency Injection)이다.

의존성 주입이란 말을 처음 들으면, 의존성이라는 단어가 무엇에 대한 의존성인지 그리고 왜 주입해야 하는지 잘 와닿지 않을 것이다. 클래스나 인터페이스들을 사용하기 위해서 해야 하는 행위들, 예를 들어서 고속도로 톨게이트를 통과할 때는 차를 멈추고 손을 뻗어서 통행권을 뽑고

가야 한다. 자동차로 고속도로를 주행할 수 있지만 고속도로를 주행하기 위해서 통행권을 뽑는 행위에 대한 의존성을 가지고 있는 것이다. 그런데 하이패스 단말기를 부착한 차량들은 의존성(통행권을 뽑고 요금을 내는 것)을 하이패스 단말기가 대신 주입해 주므로 의존성에 필요한 행위들을 하지 않고 통과할 수 있다. 이렇게 생각하면 왜 객체 생성 주입 인터페이스 주입 등등의 표현을 쓰지 않고 의존성 주입(DI)이란 용어로 고착되었는지 이해하기 쉬울 것이다. 최근에는 제어의 역전(IoC)이라는 말은 뜻이 너무 광범위하고 일반적이어서 제어의 역전보다는 의존성 주입을 많이 쓴다. 의존성 주입에 대한 이해를 돕기 위해서 의존성 주입을 적용하기 전 인터페이스를 이용한 간단한 프로그램을 만들어 보자.

## 3.2.1 인터페이스와 스프링

스프링에서 의존성을 주입할 때 기본적으로 타입을 기반으로 하게 된다. 자바에서 타입을 가장 잘 나타낼 수 있는 것은 인터페이스(interface)다. 인터페이스를 이용한 예제를 만들고 해당 예제에 스프링의 의존성 주입을 적용해 보자.

### 3.2.1.1 인터페이스를 활용한 기본 예제

인터페이스를 이용해서 같은 메서드를 가진 클래스를 두 개 만들자.

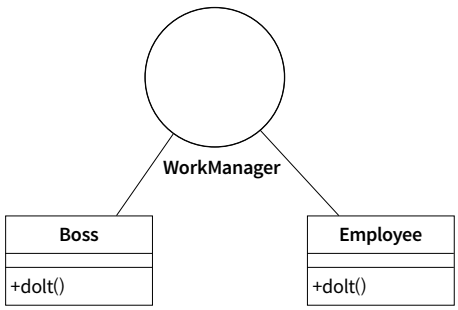



그림 3-1 **WorkManager를 상속받은 클래스들**

Boss와 Employee라는 두 개의 클래스를 만들자. 이 두 개의 클래스는 WorkManager 인터페이스를 상속받았으므로 doIt이라는 공통적인 메서드를 가지고 있다.

code file [/ch03/src/main/java/basic/Boss.java]

```
Package basic;

public class Boss implements WorkManager{
    @Override
    public String doIt() {
        return "do boss";
    }
}
```

Boss 클래스는 WorkManager 인터페이스를 상속받은 후 doIt 메서드를 오버라이드(Override)해서 문자열 타입으로 do boss를 리턴한다.

code file [/ch03/src/main/java/basic/Employee.java]

```
package basic;

public class Employee implements WorkManager{
    @Override
    public String doIt() {
        return "do work";
    }
}
```

Employee 클래스는 WorkManager 인터페이스를 상속받은 후 doIt 메서드를 오버라이드해서 문자열 타입으로 do work를 리턴한다.

code file [/ch03/src/main/java/basic/WorkManager.java]

```
package basic;

public interface WorkManager {
    public String doIt();
}
```

WorkManager는 인터페이스로 Empolyee와 Boss에서 공통적으로 사용하는 doIt 메서드를 가지고 있다. doIt는 인터페이스인 WorkManager를 제외하고, Boss와 Employee 모두 같은데 클래스에 따라 실제 수행하는 내용이 다르다. 그렇기 때문에 각 클래스에게 일을 시킬 때 직접 클래스를 생성하지 않고 인터페이스를 통해서 생성하면 필요에 따라서 역할을 변경할 수 있다. 이제 각 클래스들을 일하게 만드는 WorkService 클래스를 보자.

code file [/ch03/src/main/java/basic/WorkService.java]

```
package basic;

public class WorkService {
    WorkManager workManager;

    public void setWorkManager(WorkManager workManager){
        this.workManager = workManager;
    }

    public void askWork(){
        System.out.println( workManager.doIt() );
    }
}
```

setter 메서드(setWorkManager)를 통해서 WorkManager가 WorkService 클래스의 속성으로 생성된다. setter 메서드를 사용할 때 전달받는 클래스에 따라서 askWork에 내용이 달라진다. 실제로 askWork를 호출하는 main 메서드가 담긴 클래스를 보자.

code file [/ch03/src/main/java/basic/BasicApp.java]

```
package basic;

public class BasicApp {
    public static void main(String ar[]){
        WorkService workService = new WorkService();
        WorkManager employee = new Employee();
        WorkManager boss = new Boss();

        workService.setWorkManager(employee);
        workService.askWork();

        workService.setWorkManager(boss);
        workService.askWork();
    }
}
```

Employee 클래스를 생성한 후 WorkManager에 setter 메서드를 이용해서 Employee 클래스를 전달 후 askWork를 호출한다. askWork 메서드를 사용하기 위해서는 WorkManager 타입을 가진 Boss 또는 Employee 클래스의 인스턴스 생성이 필요하다. main 메서드를 실행해서 결과를 확인하자.

```
do work
do boss
```

CHAPTER

# 7

# 커스텀 스프링 부트 스타터

# 7.1 어노테이션

이 장에서는 스프링 부트의 구조와 많은 설정을 위해서 사용되는 어노테이션 그리고 스프링 부트에서 설정을 간소화할 수 있게 해주는 AutoConfigure에 대해 알아보고, 자신만의 스프링 부트 스타터를 만들어 보자.

어노테이션(annotation)은 자바 1.5 버전부터 지원되는 기능으로 일종의 메타데이터(metadata)다. 어노테이션의 사전적인 의미는 '주석'인데, 주석처럼 코드에 추가해서 사용할 수 있고, 컴파일 또는 런타임 시에 해석된다. 그동안 @SpringBootApplication, @Entity, @WebServlet과 같은 많은 어노테이션들을 사용했는데, 스프링 부트 스타터를 직접 만들기 위해서는 어노테이션에 대해 이해해야만 한다. 이 절에서는 어노테이션에 대해서 자세히 알아보자.

## 7.1.1 어노테이션 만들기

어노테이션을 선언하는 방법과 선언한 어노테이션을 읽는 방법을 알아보자.

### 7.1.1.1 어노테이션 선언 형식

어노테이션은 다음과 같이 interface에 @를 붙여서 선언하고, 어노테이션이 적용될 대상과 동작 방식을 지정할 수 있다.

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
Public @interface Annotaion1{}
```

Target은 어노테이션이 적용되는 대상을 의미하는데 ElementType의 요소 중에서 선택해서 지정한다. 주로 메서드를 많이 사용하는데 @Target 어노테이션에 사용되는 전체 목록은 다음과 같다.

표 7-1 **어노테이션 요소 목록**

| 대상 요소명 | 적용 대상 |
|---|---|
| TYPE | 클래스 및 인터페이스 |
| FIELD | 클래스의 멤버 변수 |
| METHOD | 메서드 |
| PARAMETER | 파라미터 |
| CONSTRUCTOR | 생성자 |
| LOCAL_VARIABLE | 지역변수 |
| ANNOTATION_TYPE | 어노테이션 타입 |
| PACKAGE | 패키지 |
| TYPE_PARAMETER | 타입 파라미터 |
| TYPE_USE | 타입 사용 |

Retention은 어노테이션이 적용될 범위를 결정하는 데 세 가지 유형이 있다.

- **Class** 어노테이션 작성 시 기본값으로 클래스 파일에 포함되지만 JVM이 로드하지 않는다.
- **Runtime** 클래스 파일에 포함되고, JVM이 로드해서 리플렉션 API로 참조 가능하다.
- **Source** 컴파일 때만 사용되고, 클래스 파일에 포함되지 않는다.

#### 7.1.1.2 MyAnnotation 생성

문자열과 숫자 타입의 값을 세팅할 수 있는 어노테이션을 만들고 리플렉션 API를 통해 참조해 보자.

code file **[/ch07/UseCustomStarter/src/main/java/info/thecodinglive/config/MyAnnotation.java]**

```
1  package info.thecodinglive.config;
2
3  import java.lang.annotation.ElementType;
4  import java.lang.annotation.Retention;
5  import java.lang.annotation.RetentionPolicy;
6  import java.lang.annotation.Target;
7
8  @Target(ElementType.METHOD)
9  @Retention(RetentionPolicy.RUNTIME)
10 public @interface MyAnnotation {
11     String strValue();
12     int intValue();
13 }
```

MyAnnotation은 strValue와 intValue를 입력할 수 있는 어노테이션이다. 이 어노테이션을 사용하는 서비스 클래스를 만들자.

code file [/ch07/UseCustomStarter/src/main/java/info/thecodinglive/service/MyService.java]

```
package info.thecodinglive.service;

import info.thecodinglive.config.MyAnnotation;

public class MyService {
    @MyAnnotation(strValue = "hi", intValue = 0607)
    public void printSomething(){
        System.out.println("test my annotation");
    }
}
```

MyAnnotation을 만들 때 요소 타입을 메서드로 정의했으므로 MyService의 printSomething 메서드에 @MyAnnotation을 선언한다. 리플렉션 API로 printSomething 메서드에 선언한 MyAnnotation 값을 확인해 보자.

code file [/ch07/UseCustomStarter/src/main/java/info/thecodinglive/AnnotationApp.java]

```
package info.thecodinglive;

import info.thecodinglive.config.MyAnnotation;
import info.thecodinglive.service.MyService;

import java.lang.reflect.Method;

public class AnnotationApp {
    public static void main(String ar[]) throws Exception{
        Method[] methods = Class.forName(MyService.class.getName()).getMethods();

        for(int i=0; i< methods.length; i++){
            if(methods[i].isAnnotationPresent(MyAnnotation.class))
            {
                MyAnnotation an = methods[i].getAnnotation(MyAnnotation.class);

                System.out.println("my annotation str value:" + an.strValue());
                System.out.println("my annotation int value:" + an.intValue());
            }

        }
    }
}
```

MyService 클래스에서 메서드 목록을 얻은 다음에 메서드 중에서 MyAnnotation이 존재하는지 체크하고 MyAnnotation에 지정된 strValue와 intValue 값을 출력한다.

```
my annotation str value::hi
my annotation int value:391
```

스프링에서 사용하는 많은 어노테이션들도 이와 같이 어노테이션이 있고, 어노테이션을 해석하는 역할을 하는 클래스가 있다. 스프링 부트에서는 설정 자동화를 위해 사용되는 어노테이션들을 알아보자.

## 7.1.2 스프링 부트 어노테이션

스프링 부트에 사용되는 많은 어노테이션들 중에서 부트에서 내부적으로 사용되는 어노테이션들을 알아보자.

### 7.1.2.1 ImportSelector

스프링 부트에는 자바로 작성된 많은 설정 클래스들이 있다. 이런 설정 클래스들이 어노테이션의 값에 따라서 로딩 여부가 결정되는데, 이럴 때 사용하는 게 ImportSelector 인터페이스다. 간단한 설정 클래스들을 만들고 ImportSelector 사용법을 알아보자. 먼저, 동작을 확인하기 위해서 메시지를 필드로 가지고 있는 MyBean 클래스와 MyBean을 이용해 메시지를 출력하는 UseMyBean 클래스를 만들자.

code file **[/ch07/UseCustomStarter/src/main/java/info/thecodinglive/importSelect/MyBean.java]**

```
1  package info.thecodinglive.importSelect;
2
3  public class MyBean {
4      private String msg;
5
6      public MyBean(String msg){
7          this.msg = msg;
8      }
9
10     public String getMsg(){
11         return msg;
12     }
13 }
```

MyBean 클래스는 생성자로 msg 필드값을 전달받고 msg 필드값은 getMsg 메서드로 출력하는 클래스다. MyBean 클래스를 사용하는 UseMyBean 클래스를 만들자.

code file [/ch07/UseCustomStarter/src/main/java/info/thecodinglive/importSelect/UseMyBean.java]

```
package info.thecodinglive.importSelect;

import org.springframework.beans.factory.annotation.Autowired;

public class UseMyBean {
    @Autowired
    private MyBean myBean;

    public void printMsg(){
        System.out.println(myBean.getMsg());
    }
}
```

UseMyBean 클래스는 @Autowired 어노테이션으로 MyBean 클래스에 의존성을 주입하고 printMsg 메서드로 MyBean 클래스가 가지고 있는 메시지 값을 출력한다. 이제 MyBean 클래스를 빈으로 등록해 주는 설정 클래스들을 만들자.

code file [/ch07/UseCustomStarter/src/main/java/info/thecodinglive/importSelect/AConfig.java]

```
package info.thecodinglive.importSelect;

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

@Configuration
public class AConfig {
    @Bean
    MyBean myBean(){
        return new MyBean("from Aconfig");
    }
}
```

AConfig 클래스는 곧이어 살펴볼 BConfig 클래스와 동일한 설정 클래스로서 MyBean 클래스를 빈으로 등록하는 역할을 한다. ImportSelector 확인을 위해 비슷한 기능을 하는 BConfig 클래스를 만들자.