8장 스프링이란 무엇인가?

8.1 스프링의 정의

> 자바 엔터프라이즈 개발을 편하게 해주는 오픈소스 경량급 애플리케이션 프레임워크

- 애플리케이션 프레임워크
  - 특정 기술이나 기술, 업무 분야에 국한되지 않고 애플리케이션 전 영역을 포괄하는 범용적인 프레임워크를 말한다. 애플리케이션 개발의 전 과정을 빠르고 편리하며 효율적으로 진행하는데 일차적인 목표를 두는 프레임워크다.
  - 스프링의 일차적인 존재 목적은 **핵심 기술에** 담긴 프로그래밍 모델을 **일관**되게 적용해서 엔터프라이즈 애플리케이션 전 계층과 전 영역에 **전략과 기능**을 제공해 줌으로써 애플리케이션을 **편리하게 개발**하게 해주는 애플리케이션 프레임워크로 사용되는 것이다.
- 경량급
  - 불필요하게 무겁지 않다

    EJB에 비해 단순한 개발툴과 기본적인 개발환경에서 엔터프라이즈 개발이 충분하다.
  - 스프링을 기반으로 제작되는 코드가 기존 EJB나 여태 프레임워크에서 동작하기 위해 만들어진 코드에 비해 상대적으로 작고 단순하다는 뜻이기도 하다.
- 자바 엔터프라이즈 개발을 편하게
  - **근본적인 부분에서 엔터프라이즈 개발의 복잡함을 제거**하고 진정으로 개발을 편하게 해주는 해결책을 제시한다.
  - 엔터프라이즈 개발에서 필연적으로 요구되는 기술적인 요구를 충족하면서 개발을 복잡하게 만들지 않는다.
- 오픈소스
  - 스프링의 개발 과정은 공개되어 있지만 공식적인 개발은 제한된 인원의 개발자에 한정된다.

- 오픈 소스 개발이지만 프레임워크 사용자에게 지속적인 신뢰를 줄 수 있도록 개발을 책임지고 진행할 수 있는 전문 기업을 만들었다.→ 안정적이고 전문화된 개발과 품질관리가 가능해졌다.

8.2 스프링의 목적

8.2.1 엔터프라이즈 개발의 복잡함

- 복잡함의 근본적인 이유

  1. 기술적인 제약조건과 요구사항이 늘어가기 때문이다.

     `엔터프라이즈 시스템`: 서버에서 동작하며 기업과 조직의 업무를 처리해주는 시스템 → 많은 사용자의 요청을 동시에 처리해야하기 때문에 서버 자원을 효율적으로 공유하고 분배해서 사용해야 한다. 보안과 안정성, 확장성면에서 뛰어나야 한다. ⇒ 순수한 비지니스 로직을 구현하는 것 외에도 기술적으로 고려할 사항이 많다.

  2. 엔터프라이즈 애플리케이션이 구현해야 할 핵심기능인 비지니스 로직의 복잡함이 증가하기 때문이다.

     엔터프라이스 시스템이 관여하는 업무의 비율이 급격하게 커졌다.

- 복잡함을 가중시키는 원인
  - 비지니스 로직의 복잡함과 기술적인 복잡함이다.
  - 세부 요소가 이해하기 힘든 방식으로 얽혀 있고, 그 때문에 쉽게 다루기 어렵다.
  - 근본적인 비지니스 로직과 엔터프라이즈 기술이라는 두가지 복잡함이 한 데 얽혀 있어 더 어렵다.

8.2.2 복잡함을 해결하려는 도전

- 제거될 수 없는 근본적인 복잡함

  근본적으로 엔터프라이즈 개발에 나타나는 복잡함의 원인은 제거 대상이 아니다. → 그 복잡함을 효과적으로 상대할 수 있는 전략과 기법이 필요하다.

  ⇒ 성격이 다른 두 가지 복잡함을 분리해야 한다.

- 실패한 해결책: EJB

EJB는 일부 기술적인 복잡함을 덜어주려는 시도를 하다가 오히려 더 큰 복잡함을 추가하게 됐다. EJB틀안에서 자바 코드를 만들게 강제함을써 자바 언어가 원래 갖고 있던 장점도 잃게 됐고, 더 이상 상속구조를 적용하지 못하게 만들거나, 다형성 적용을 근본적으로 제한했다.→ 그 기술과 관련된 코드나 규약 등이 코드에 등장하는 경우를 침투적인 기술이라고 한다.

- 비침투적인 방식을 통한 효과적인 해결책: 스프링

  기술적인 복잡함을 애플리케이션 핵심 로직의 복잡함에서 제거하는데 목표를 뒀다.

  비침투적인 기술: 기술의 적용 사실이 코드에 직접 반영되지 않는다.

8.2.3 복잡함을 상대하는 스프링의 전략

- 기술적 복잡함을 상대하는 전략

  1. 기술에 대한 접근 방식이 일관성이 없고, 특정환경에 종속적이다.

     일관성 없는 기술과 서버환경의 변화에 대한 스프링의 공략 방법은! **서비스 추상화**다.

     기술적인 복잡함은 일단 **추상화**를 통해 로우레벨의 기술 구현 부분과 기술을 사용하는 인터페이스로 분리하고, 환경과 세부 기술에 **독립적인 접근 인터페이스**를 제공하는 것이 가장 좋은 해결책이다.

  2. 기술적인 처리를 담담하는 코드가 성격이 다른 코드에 섞여서 등장한다.

     트랜잭션, 보안 적용, 데이터와 예외의 일괄 변환, 로깅 등...

     기술과 비지니스 로직의 혼재로 발생하는 복잡함을 해결하기 위한 스프링의 접근 방법은 **AOP**다.

     AOP는 최후까지 애플리케이션 로직을 담당하는 코드에 남아 있는 기술 관련코드를 깔끔하게 분리해서 별도의 모듈로 관리하게 해주는 강력한 기술이다.

- 비지니스와 애플리케이션 로직의 복잡함을 상대하는 전략

  비지니스 로직의 복잡함을 상대하는 전략은 자바라는 객체지향 기술 그 자체다. 스프링은 단지 객체지향 언어의 장점을 제대로 살리지 못하게 방해했던 요소를 제거하도록 도와줄 뿐이다.

- 핵심도구: 객체지향과 DI

스프링이 기술과 비지니스 로직의 복잡함을 해결하는데 공통적으로 사용한 도구가 객체지향이다.

객체지향의 설계 기법을 잘 적용할 수 있는 구조를 만들기 위해 DI같은 유용한 기술을 편하게 적용하도록 도와주는 것이 스프링의 기본 전략이다.

스프링에서 기술적인 복잡함을 효과적으로 다루게 해주는 기법은 모두 DI를 바탕으로 한다.

DI는 객체 지향설계 기술 없이는 존재 의미가 없다.

객체지향적인 특성을 잘 살린 설계는 상속과 다형성, 위임을 포함해 많은 객체지향 디자인 패턴과 설계 기법이 잘 녹아들어갈 수 있다.

스프링의 기술과 전략은 객체지향이라는 자바 언어가 가진 강력한 도구를 극대화해서 사용할 수 있도록 **돕는 것**이다.

8.3 POJO 프로그래밍

8.3.1 스프링의 핵심: POJO

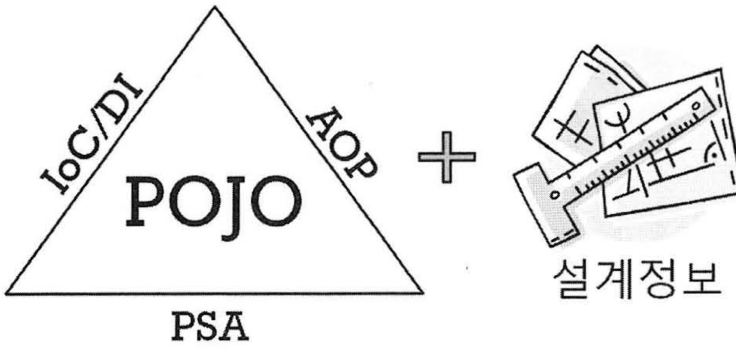

그림 8-1 스프링 삼각형

스프링 애플리케이션은 POJO를 이용해 만든 애플리케이션 코드와 POJO가 어떻게 관계를 맺고 동작하는지를 정의해놓은 설계정보다.

DI의 기본 개념(유연하게 확장 가능한 오브젝트를 만들어두고 그 관계는 외부에서 다이나믹하게 설정해준다)을 애플리케이션 전반에 적용하는 것이 스프링 프로그래밍 모델이다.

스프링 주요 기술인 IoC/DI, AOP, PSA는 애플리케이션을 POJO로 개발할 수 있게 해주는 가능 기술이라고 불린다.

8.3.2 POJO란 무엇인가

Plain Java Object

> “ 간단한 자바 오브젝트를 사용하는데요 ” = “POJO방식의 기술을 사용합니다”

8.3.3 POJO의 조건

- 특정 규약에 종속되지 않는다.

  별다른 가치를 주지 않는 규약에 종속되지 않아야 하고, 객체지향 설계의 자유로운 적용이 가능한 오브젝트여야 한다.

- 특정 환경에 종속되지 않는다.

  POJO는 환경에 독립적이어야 하고, 비지니스 로직을 담고 있는 POJO클래스는 웹이라는 환경정보나 웹기술을 담고 있는 클래스나 인터페이스를 사용해서는 안 된다.

  어노테이션이 단지 코드로 표현하기에는 적절치 않은 부가적인 정보를 담고 있고, 그 때문에 환경에 종속되지만 않는다면 POJO다.

  자바의 문법을 지키고 순수하게 JavaSE API만 사용했다고 해서 그 코드가 POJO라곤 할 수 없다 → POJO는 **객체지향적인 자바 언어의 기본에 충실**하게 만들어져야 한다.

POJO 프로그래밍: 객체지향적인 원리에 충실하면서, 환경과 기술에 종속되지 않고 필요에 따라 재활용될 수 있는 방식으로 설계된 POJO에 애플리케이션 핵심 로직과 기능을 담아 설계하고 개발하는 방법

8.3.4 POJO의 장점

- 특정한 기술과 환경에 종속되지 않는 오브젝트는 그만큼 깔끔한 코드가 될 수 있다.
- 자동화된 테스트에 매우 유리하다.
- 객체지향적인 설계를 자유롭게 적용할 수 있다.

8.3.5 POJO 프레임워크

스프링은 엔터프라이즈 애플리케이션 개발의 모든 영역과 계층에서 POJO방식의 구현이 가능하게 하려는 목적으로 만들어졌다.

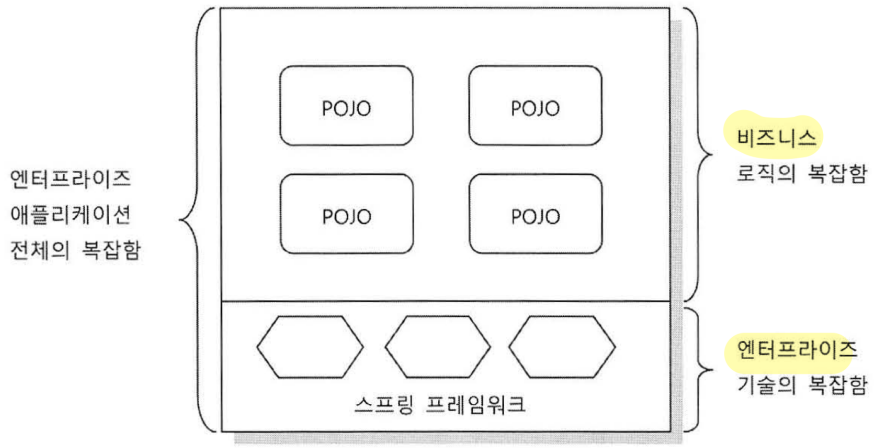

그림 8-2 엔터프라이즈 개발의 복잡함을 상대하는 스프링 프레임워크와 POJO

POJO프레임워크로서 스프링은 자신을 직접 노출하지 않으면서 애플리케이션을 POJO로 쉽게 개발할 수 있게 지원해준다.

8.4 스프링의 기술

스프링의 기술들은 스프링 프레임워크가 만들어진 진정한 목표인 POJO기반의 엔터프라이즈 개발을 편리하게 해주는 도구일 뿐이다.

8.4.1 제어의 역전(IoC)/의존관계 주입(DI)

스프링의 가장 기본이 되는 기술이자 스프링의 핵심 개발원칙. 템플릿,콜백 패턴, AOP,PSA 도 IoC/DI가 핵심원리이다.

> 유연한 확장이 가능하며 개방폐쇄 원칙(OCP)으로 설명될 수 있다. 개방→유연한 확장, 폐쇄→ 재사용가능 의 장점이 있다.

- DI 활용 방법
  - 핵심기능의 변경

    DI의 가장 대표적인 적용방법은 의존 대상의 구현을 바꾸는 것이다. 대표적인 예가 전략패턴 이다.
  - 핵심기능의 동적인 변경

의존 오브젝트의 핵심기능 자체를 바꾸는 것인데 동적으로 매번 다르게 변경하는 것이다. ⇒ 애플리케이션이 동작하는 중간에 의존 대상을 동적으로 변경할 수 있다.

동적인 방식으로 핵심기능을 변경하는 건 다이내믹 라우팅 프록시나 프록시 오브젝트 기법을 활용한 것이다.

- 부가기능의 추가

  데코레이터 패턴. 핵심 기능은 그대로 둔 채 부가기능을 추가하는 것이다.

  인터페이스를 두고 사용하게 하고, 실제 사용할 오브젝트는 외부에서 주입하는 DI를 적용하면 된다. 트랜잭션 기능이 대표적인 예다. 부가 기능의 추가 방식을 특정 오브젝트가 아닌 여러 대상으로 일반화해서 적용하면 AOP가 된다.

- 인터페이스의 변경

  클라이언트가 사용하는 인터페이스와 실제 오브젝트 사이에 인터페이스가 일치하지 않는 경우에도 DI가 유용하다.

  A는 C오브젝트를 사용하려 하고, A는 원래 B인터페이스를 사용하도록 되어있고 C는 B인터페이스를 구현하지 않았다. 이때 A가 DI를 통해 B의 구현 오브젝트를 받도록 만들어져 있다면 B인터페이스를 구현하면서 내부에 C를 호출해주는 기능을 가진 **어댑터 오브젝트**를 만들어 A에 DI해주면 된다.

  A→B(C로 위임)→C

  아예 인터페이스가 다른 다양한 구현을 같은 방식으로 사용하도록 인터페이스 어댑터 역할을 해주는 레이어를 중간에 추가하는 방법도 있다. → 스프링의 일관성 있는 서비스 추상화의 방법이다.

- 프록시

  프록시 패턴의 전형적인 응용방법이다. 필요한 시점에서 실제 사용할 오브젝트를 초기화하고 리소스를 준비하게 해주는 지연된 로딩을 적용하려면 프록시가 필요하고, 원격 오브젝트를 호출할 때 마치 로컬에 존재하는 오브젝트처럼 사용할 수 있게 해주는 원격 프록시를 적용할때도 프록시가 필요하다. 두 방법 모두 DI가 필요하다.

- 템플릿과 콜백

  템플릿 콜백 패턴은 DI의 특별한 적용 방법이다.

  콜백을 템플릿에 주입하는 방식으로 동작하게 하는 것은 DI원리에 가장 충실한 응용 방법이다. 콜백을 얼마든지 만들어서 사용할 수 있다는 건 개방을 통한 유

연한 확장성을 보여주는 것이며 템플릿을 계속 재사용할 수 있는 것이 OCP에 해당한다.

- 싱글톤과 오브젝트 스코프

  DI할 오브젝트의 생명주기를 제어할 수 있다는 면에서 중요하다.DI를 프레임워크로 이용한다는 건 DI대상 오브젝트를 컨테이너가 관리한다는 의미다.

  스프링에서는 싱글톤외에도 다양한 스코프를 갖는 오브젝트를 만들어 DI에 사용할 수 있다.

- 테스트

  DI를 통해 의존 오브젝트를 대신해 스텁 또는 목 오브젝트 같은 테스트 대역을 활용할 수 있다. → 여타 오브젝트와 협력해서 동작하는 오브젝트를 고립시켜 효과적으로 테스트할 수 있게 한다.

8.4.2 애스펙트 지향 프로그래밍(AOP)

AOP는 객체지향 기술의 한계와 단점을 극복하도록 도와주는 보조적인 프로그래밍 기술이다.

- AOP 적용 기법

  1. 스프링과 같이 다이내믹 프록시를 사용하는 방법

     기존 코드에 영향을 주지 않고 부가기능을 적용하게 해주는 데코레이터패턴을 응용한 것. 부가기능을 부여할 수 있는 곳은 메소드 호출이 일어난느 지점뿐이라는 제약이 있다.

  2. 자바 언어의 한계를 넘어서는 언어의 확장을 이용하는 방법

     AspectJ 라는 오픈소스 AOP툴을 써서 다양한 조인 포인트를 쓸 수 있다. 자바 언어와 JDK지원만으로는 불가능하고 별도의 AOP컴파일러를 이용한 빌드 과정을 거치거나 클래스가 메모리로 로딩될 때 그 바이트 코드를 조작하는 위빙과 같은 별도의 방법을 이용해야한다.

- AOP 적용 단계

  1. 미리 준비된 AOP 이용

     스프링이 미리 만들어 제공하는 AOP기능을 그대로 가져다 적용해본다. 대표적으로 트랜잭션이 있다. AOP 설정을 통해 트랜잭션이 어떻게 많은 오브젝트에 적용되는 지 관찰해보고 AOP 특성과 동작원리를 이해해본다.

@Configurable를 이용해 도메인 오브젝트에 DI를 자동적용해주는 AOP기능도 있다. 도메인 오브젝트를 전용 계층에 두고 접근하는 아키텍쳐 방식을 따를 때 반드시 필요한데, AspectJ를 이용한 AOP가 필요하다.

2. 전담팀을 통한 정책 AOP 적용

   애플리케이션 전체적으로 이용가능한 것을 소수의 AOP 담당자 관리하에 적용해 본다. 비지니스로직을 가진 오브젝트에 대한 보안, 특정 계층의 오브젝트 이용 전 후의 작업 기록을 남기는 로깅, 데이터 추적을 위한 트레이싱, 특정 구간의 실시간 성능 모니터링과 같은 기능을 적용해본다.

3. AOP의 자유로운 이용

   개발자가 구현하는 기능에 적용하면 유용한 세부적인 AOP를 이용한다.

8.4.3 포터블 서비스 추상화(PSA)

POJO코드가 특정환경과 기술에 직접 노출되어 만들어지지 않기 위해 스프링이 일관성이 있는 서비스 추상화 기술을 제공한다. 테스트가 어렵게 만들어진 API나 설정을 통해 주요 기능을 외부에서 제어하게 만들고 싶을때도 이용한다.

서비스 추상화를 위해 필요한 기술은 DI 뿐이다. → 서비스 추상화는 자연스럽게 만들어 쓸 수 있다.

8.5 정리

1. 스프링은 그 개발철학과 목표를 분명히 이해하고 사용해야 한다.
2. 스프링은 오픈소스 소프트웨어이며, 애플리케이션 개발의 모든 기술과 영역을 종합적으로 다루는 애플리케이션 프레임워크다.
3. 엔터프라이즈 애플리케이션 개발의 복잡함은 비지니스 로직과 엔터프라이즈 시스템의 기술적인 요구에 의해 발생된다. 기존의 접근 방법은 이 복잡도를 낮추지 못하며 자바의 객체지향적인 장점을 포기해야 한다는 문제점이 있다.
4. 자바의 근본인 객체지향적인 원리에 충실하게 개발할 수 있으며, 환경과 규약에 의존적이지 않은 POJO를 이용한 애플리케이션 개발은 엔터프라이즈 시스템 개발의 복잡함이 주는 많은 문제를 해결할 수 있다.
5. 스프링의 목적은 이런 POJO를 이용해 엔터프라이즈 애플리케이션을 쉽고 효과적으로 개발할 수 있도록 지원해주는 데 있다.

6. POJO방식의 개발을 돕기 위해 스프링은 IoC/DI, AOP, PSA와 같은 가능기술을 프레임워크와 컨테이너라는 방식을 통해 제공한다.