Spring 정의

2012년 1월 31일 화요일
오전 9:17

1. 개요

1.1. 목적

수 많은 프로젝트에서 프레임워크나 아키텍체에 대한 관심 없이 대부분의 개발을 개발자의 능력에 전담시키는 것이 일반적이다. 이는 프로젝트의 위험 요소를 증가시킬 뿐만 아니라 개발 완료 후 유지보수 비용을 증가시킴으로써 추가적인 비용 부담을 초래할 뿐더러 안정성에도 문제가 되곤 한다.
이에 본 내용은 Spring Framework를 통해 앞에서의 문제점들을 해결할 수 있는데 초점을 맞췄으며, Spring Framework은 J2EE 애플리케이션 개발을 보다 쉽고 완성도 높은 결과물을 보장해 준다.

1.2 대상

모든 Java 개발자를 대상으로 하지만 특히 Enterprise Java 개발자에 초점을 맞추고 있다.
Spring Framework의 상당 부분이 Enterprise Java의 대안으로 활용될 수 있게 설계되었으며 이를 통해 보다 쉽게 개발할 수 있는 방법들을 제공한다.
참고로 Enterprise Java 개발자가 아닐지라도 Spring이 제공하는 기능들을 통해 보다 효율적이고 실용적인 개발을 할 수 있을 것이다.

1.3 범위

Spring Framework의 개념과 주요 기능에 대한 설명과 함께 주로 Spring MVC를 통한 Web Application 개발에 주된 범위를 두고 있다.

1.4 Spring Framework 활용 시 결정 사항

Java를 기반으로 Web Application 개발을 시작하는 초급 개발자에겐 접근하기 어려운 프레임워크이다. 먼저 J2EE 환경에 대한 기본적인 지식과 경험을 갖춘 다음에 접근하는 것이 Spring Framework를 이해하는데 많은 도움이 될 것이다. 참고로 본 프레임워크를 선정할 시엔 개발자들의 역량과 해당 프로젝트에 적합한지를 고려해야 할 것이다.

- J2EE(Java, JSP, Servlet, EJB) 개발 환경에 1년 이상의 경험을 가진 개발자.
- 효율적인 애플리케이션 개발 방법을 찾고 있는 개발자.
- 개발 생산성과 유지보수성의 향상을 높이고자 하는 개발자.
- 체계화된 개발을 하고자 하는 개발자.
- 표준 프레임워크를 도입하고자 하는 개발자. Spring Framework는 표준 프레임워크로 채택해도 손색이 없을 만큼 모든 영역을 포괄하고 있다.

2. Spring 특징

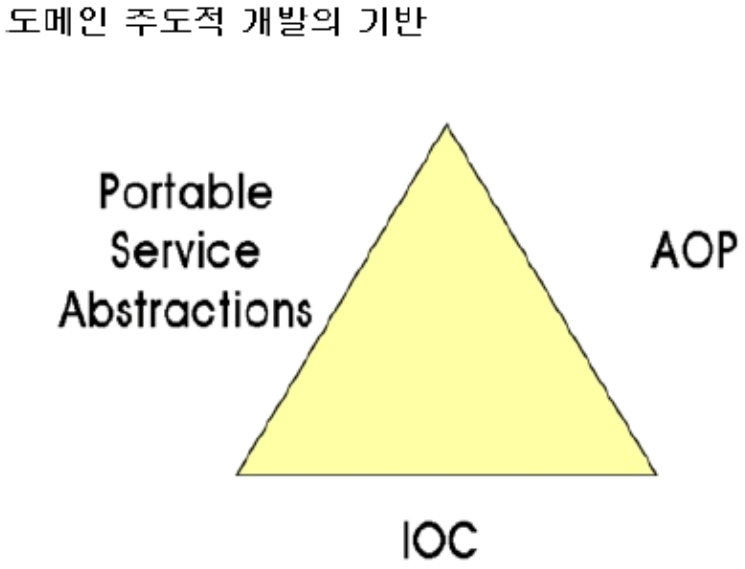
2.1. Spring 공약 / 철학

- 좋은 설계가 기술 자체보다 더 중요하다.
- Interface를 통한 느슨한 결합의 Java Beans는 훌륭한 모델이다.
- 코드는 테스트하기 쉬워야 한다.

2.2. Spring 이란?

Spring 프레임워크를 살펴보다 보면 Dependency Injection(이하 DI), Inversion Of Control(이하 IoC), Light Weight Container(이하 경량 컨테이너)라는 단어들이 무수히 많이 나온다.
먼저 Spring 프레임워크를 요약하자면 객체의 라이프 사이클을 관리하기 위하여 DI를 사용하는 경량 컨테이너 이다.
좀더 명확하게 설명하자면, Spring은 복잡한 Enterprise Application 개발을 겨냥해 만들어졌다. 평범한 Java Beans를 사용함에도 EJB에서만 가능했던 일들이 모두 이루어진다. 또한 서버측 개발에만 국한되지 않는 모든 Java Application 개발에 활용할 수 있으며 이를 통해 단순성, 테스트 용이성, 느슨한 결합성을 보장 받을 수 있다.

2.3. Spring 사용 이유



1) AOP(기능별 모듈화, 진정한 OOP 제공)
컨테이너는 일관성을 유지시켜 주고 투명한 환경 내에서 느슨한 컴포넌트(POJO)의 집합에서 복잡한 시스템을 조립할 수 있는 능력을 제공하며 조직을 해치지 않음.

2) IOC(제어 역행)
애플리케이션 객체를 연결해 주고 자동화된 설정 및 집중화된 설정을 제공하는 가장 완전한 경량 컨테이너.

3) Test Unit(편리한 테스트) 제공
컨테이너는 민첩함을 제공하고 지렛대 역할을 하며 소프트웨어 컴포넌트를 먼저 개발하고 고립시켜 테스트할 수 있게 함으로써 테스트와 확장성을 향상시킨다.

4) 트랜잭션
트랜잭션 관리를 위한 공통의 추상화된 레이어, 트랜잭션 관리자를 플러그인할 수 있어서 저 수준 트랜잭션을 문제없이 처리한다.

5) JDBC 추상화 레이어
중요한 예외 계층을 제공하며 예외처리를 단순화시켜 코드의 양을 덜어준다.

6) ORM 프레임워크 연동 제공
Hibernate, iBatis, JDO 등과 같은 ORM 프레임워크와 통합되어 있다.

7) 좀더 쉬운 J2EE 개발 지향(저비용 유연한 코드 유지)

8) 다양한 프리젠테이션 계층 제공(jsp, velocity, excel, pdf ...)

9) 좋은 설계(아키텍처) 제공

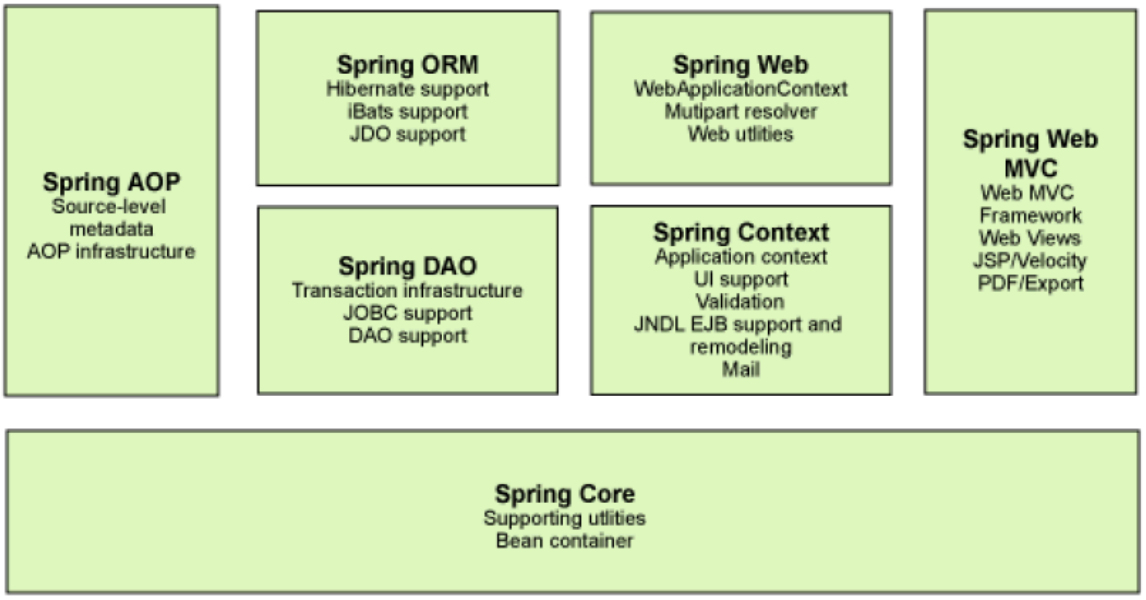
10) 분산(원격) 서비스

11) 보안

Spring Framework는 비즈니스의 목적에 부합하기 위해 다양한 컴포넌트들의 집합체이며, 상호 의존성이 없는 유연한 구조를 통해 재사용과 단위 테스트를 쉽게 해준다. 이를 통해 연관관계를 관리할 책임이 없다. 그대신, 컨테이너에 의해 컴포넌트 간의 협업을 위한 참조가 주어질 뿐이다.

2.4. 스프링 모듈

Spring은 7개의 잘 정의된 모듈들로 구성되며 전체적으로 이들 모듈은 엔터프라이즈 애플리케이션 개발에 필요한 모든 것을 제공한다. 이는 애플리케이션이 완전히 Spring 프레임워크를 기반으로 해야 하는 것은 아니다. 즉, 애플리케이션에 적합한 모듈을 선택하여 적용하고 나머지 모듈들은 무시해도 된다. Spring 모듈은 모두 핵심 컨테이너 위에 구축되어있다. 핵심 컨테이너는 Bean 생성, 설정, 관리하는 방법을 정의하는데 이는 곧 Spring의 근본적인 기능이다.



1) 코어 컨테이너(core container)
Spring 프레임워크의 핵심 기능을 제공한다. 코어 컨테이너의 주요 컴포넌트는 Bean-Factory(Factory 패턴의 구현)이다. BeanFactory는 Inversion of Control (IOC) 패턴을 사용하여 애플리케이션의 설정 / 의존성 스팩을 실제 애플리케이션 코드에서 분리시킨다.

2) Spring 컨텍스트(Spring context)
Spring을 컨테이너로 만든 것이 핵심 모듈의 BeanFactory라면, Spring을 프레임워크로 만든 것은 컨텍스트 모듈이다. 이 모듈은 국제화된 메시지, 애플리케이션 생명주기 이벤트, 유효성 검증 등을 지원함으로써 BeanFactory의 개념을 확장한다.
이 모듈은 이메일, JNDI 접근, EJB 연계, 리모팅, 스케쥴링 등과 같은 다수의 엔터프라이즈 서비스를 추가로 제공한다. 또한 템플릿 프레임워크와의 통합도 지원한다.

3) Spring AOP 모듈(Spring AOP)
설정 관리 기능을 통해 aspect 지향 프로그래밍 기능을 Spring 프레임워크와 직접 통합시킨다. 따라서 Spring 프레임워크에서 관리되는 모든 객체에서 AOP가 가능하다. Spring AOP 모듈은 Spring 기반 애플리케이션에서 객체에 트랜잭션 관리 서비스를 제공한다.
Spring AOP에서는 EJB 컴포넌트에 의존하지 않고도 선언적 트랜잭션 관리를 애플리케이션과 결합할 수 있다.

4) Spring DAO
Spring JDBC DAO 추상 레이어는 다른 데이터베이스 벤더들의 예외 핸들링과 오류 메시지를 관리하는 중요한 예외 계층을 제공한다. 이 예외 계층은 오류 핸들링을 간소화하고, 예외 코드의 양도 줄여준다.
Spring DAO의 JDBC 예외는 일반 DAO 예외 계층에 순응한
다.

5) Spring ORM
프레임워크는 여러 ORM 프레임워크에 플러그인 되어, Object Relational 툴 (JDO, Hibernate, iBatis SQL Map)을 제공한다. 이 모든 것은 Spring의 일반 트랜잭션과 DAO 예외 계층에 순응한다.

6) Spring Web module
웹 컨텍스트 모듈은 애플리케이션 컨텍스트 모듈의 상단에 구현되어, 웹 기반 애플리케이션에 컨텍스트를 제공한다. Spring 프레임워크는 Jakarta Struts와의 통합을 지원한다.
웹 모듈은 다중 요청을 핸들링하고, 요청 매개변수를 도메인 객체로 바인딩하는 작업을 수월하게 한다.

7) Spring MVC framework
MVC 프레임워크는 완전한 기능을 갖춘 MVC 구현이다. MVC 프레임워크는 전략 인터페이스를 통해 설정할 수 있으며, JSP, Velocity, Tiles, iText, POI 같은 다양한 뷰 기술을 허용한다.

원본 위치 <http://ingenuity.egloos.com/3100396>

1. 스프링 프레임워크 소개

* 스프링이란?
- 스프링(Spring)은 간단히 말하면 엔터프라이즈 어플리케이션에서 필요로 하는 기능을 제공하는 프레임워크이다. 스프링은 J2EE가 제공하는 다수의 기능을 지원하고 있기 때문에, J2EE를 대체하는 프레임워크로 자리 잡고 있다.

* 스프링 프레임워크 특징

- 스프링은 경량 컨테이너이다. 스프링은 자바 객체를 담고 있는 컨테이너이다. 스프링은 이들 자바 객체의 생성, 소멸과 같은 라이프 사이클을 관리하며, 스프링으로부터 필요한 객체를 가져와 사용 할 수 있다.
- 스프링은 DI(Dependency Injection) 패턴을 지원한다. 스프링은 설정 파일을 통해서 객체 간의 의존 관계를 설정할 수 있도록 하고 있다. 따라서 객체는 직접 의존하고 있는 객체를 생성 하거나 검색할 필요가 없다.
- 스프링은 AOP(Aspect Oriented Programming)를 지원한다.스프링은 자체적으로 AOP를 지원하고 있기 때문에 트랜잭션이나 로깅, 보안과 같이 여러 모듈에서 공통으로 필요로 하지만 실제 모듈의 핵심은 아닌 기능들을 분리해서 각 모듈에 적용할 수 있습니다.
- 스프링은 POJO(Plain Old Java Object)를 지원한다. 스프링 컨테이너에 저장되는 자바 객체는 특정한 인터페이스를 구현하거나 클래스를 상속받지 않아도 된다. 따라서 기존에 작성한 코드를 수정할 필요 없이 스프링에서 사용할 수 있다.
- 트랙젝션 처리를 위한 일관된 방법을 제공한다. JDBC를 사용하든, JTA를 사용하든, 또는 컨테이너가 제공하는 트랙잭션을 사용하든, 설정 파일을 통해 트랜잭션 관련 정보를 입력 하기 때문에 트랙잭션 구현에 상관없이 동일한 코드를 여러 환경에서 사용 할 수 있다.
- 영속성과 관련된 다양한 API를 지원한다. 스프링은 JDBC를 비롯하여 iBATIS, Hibernate, JPA, JDO등 데이터베이스 처리와 관련하여 널리 사용되는 라이브러리와의 연동을 지원하고 있다.
- 다양한 API에 대한 연동을 지원한다. 스프링은 JMS, 메일, 스케쥴링 등 엔터프라이즈 어플리케이션을 개발하는데 필요한 다양한 API를 설정 파일을 통해서 손쉽게 사용할 수 있도록 하고 있다.

* IoC(Inversion of Control)란?

- Spring 프레임워크가 가지는 가장 핵심적인 기능은 IoC(Inversion of Control)이다.자바가 등장한 최초에는 객체 생성 및 의존관계에 대한 모든 제어권이 개발자에 있었다. 그러나, 서블릿, EJB가 등장하면서 제어권이 서블릿과 EJB를 관리하는 서블릿컨테이너 및 EJB 컨테이너에게 넘어가게 되었다. Spring 프레임워크도 객체에 대한 생성 및 생명주기를 관리할 수 있는 기능을 제공하고 있다. 이와 같은 이유때문에 Spring 프레임워크를 Spring 컨테이너, IoC컨테이너와 같은 용어로 부르기도 한다.

: 물론 모든 객체에 대한 제어권을 컨테이너에게 넘겨버린 것은 아니다. 서블릿 컨테이너와 EJB 컨테이너에서도 서블릿과 EJB에 대한 제어권만 컨테이너가 담당하고 나머지 객체에 대한 제어권은 개발자들이 직접 담당하고 있다. 이처럼 Spring 컨테이너 일부 POJO(Plain Old Java Object)에 대한 제어권을 가진다. Spring컨테이너에서 관리되는 POJO는 각 계층의 인터페이스를 담당하는 클래스들에 대한 제어권을 가지는 것이 대부분이다.

* Dependency Injection

- DI는 Spring 프레임워크에서 지원하는 IoC의 한 형태이다. DI를 간단히 말하면, 객체 사이의 의존관계를 객체 자신이 아닌 외부의 조립기(assembler)가 수행한다는 개념이다.

일반적인 웹어플리케이션의 경우 클라이언트의 요청을 받아주는 컨트롤러 객체, 비지니스 로직을 수행하는 서비스 객체, 데이터에 접근을 수행하는 DAO객체 등으로 구성된다. 만약, WriteArticleServiceImple클래스가 ArticleDao 인터페이스에 의존하고 있을 경우를 생각해보자

의존관계를 형성하는 방법에는 아래와 같은 경우들이 있다.

첫번째, 코드에 직접 의존 객체를 명시하는 경우.

```
public class WriteArticleServiceImpl{

   private Article articleDao = new MysqlArticleDao();

....

}
```

이 경우 의존하는 클래스가 변경되는 경우 코드를 변경해야 하는 문제가 있다.

두번째, Factory 패턴이나 JNDI등을 사용해서 의존 클래스를 검색하는 방법이 있다.

```
public class WriteArticleServiceImpl{

   private ArticleDao articleDao = ArticleDaoFactory.create();

   ...

}
```

Factory나 JNDI를 사용하면 첫번째문제(즉, 의존 클래스가 변경되면 코드를 변경해야 하는 문제)를 없앨 수는 있지만, WriteArticleServiceImpl클래스를 테스트하려면 올바르게 동작하는 Factory또는 JNDI에 등록된 객체를 필요로 한다는 문제점이 있다.

세번째, DI패턴이다. 이 방식에서는 의존관계에 있는 객체가 아닌 외부 조립기(assembler)가 각 객체 사이의 의존 관계를 설정해준다.
WriteArticleServiceImpl클래스의 코드는 MysqlArticleDao 객체를 생성하거나 검색하기 위한 코드가 포함되어 있지 않다. 대신 조립기의 역할을 하는 Assembler가 MysqlArticleDao 객체를 생성한 뒤 WriteArticleServiceImpl객체에 전달해 주게 된다.

DI패턴을 적용할 경우 WriteArticleServiceImpl클래스는 의존하는 객체를 전달받기 위한 설정 메서드(setter method)나 생성자를 제공할 뿐 WriteArticleServiceImpl에서 직접 의존하는 클래스를 찾지 않는다.

* AOP(Aspect Oriented Programming)
- 로깅, 트랙잭션처리, 보안과 같은 핵심 로직이 아닌 cross-cutting concern(공통관심사항)을 객체 지향기법(상속이나 패턴등)을 사용해서 여러 모듈에 효과적으로 적용하는데 한계가 있었으며, 이런 한계를 극복하기 위해 AOP라는 기법이 소개되었다.

공통관심사항(cross-cutting concern)을 별도의 모듈로 구현한 뒤, 각 기능을 필요로 하는 곳에서 사용하게 될 경우, 각 모듈과 공통 모듈 사이의 의존관계는 복잡한 의존 관계를 맺게 된다.

AOP에서는 각클래스에서 공통 관심사항을 구현한 모듈에 대한 의존 관계를갖기보다는,

Aspect를 이용하여 핵심 로직을 구현한 각 클래스에 공통 기능을 적용하게 된다.

AOP에서는 핵심 로직을 구현한 클래스를 실행하기 전 후에 Aspect를 적용하고, 그 결과로 핵심 로직을 수행하면 그에 앞서 공통 모듈을 실행하거나 또는 로직 수행 이후에 공통 모듈을 수행하는 방식으로 공통 모듈을 적용하게 된다.

AOP에서 중요한 점은 Aspect가 핵심 로직 구현 클래스에 의존하지 않는다는 점이다. AOP에서는 설정 파일이나 설정클래스 등을 이용하여 Aspect를 여러 클래스에 적용할 수 있도록 하고 있다.

원본 위치 <http://20041204.tistory.com/39>