

[LOCA TechTalk] Spring Intro

Spring 프레임워크란?

Spring?

특징

DI(의존성 주입, Dependency Injection)

AOP(관점 지향 프로그래밍, Aspect-Oriented Programming)

IOC(제어의 역전, Inversion Of Control)

MVC

간편하다 (빠르고 쉬움, 보안성, WAS내장)

Spring 주요 개념 - Container와 Bean

Container

Bean

Spring Container 생성 및 Bean 객체 호출 과정

(참고) pom.xml

applicationContext.xml 설정파일 (Bean 생성)

GenericXmlApplicationContext 객체 (컨테이너 생성)

getBean() (컨테이너를 통해 Bean 접근)

요즘방식

Spring 프로젝트 구조

웹 프로그래밍 설계 모델

Model1 방식

Model2 방식, MVC

Spring MVC 프레임워크 구조 및 흐름

1. 사용자의 모든 요청은 DispatcherServlet 이 처음 받는다.

2. DispatcherServlet → HandlerMapping → DispatcherServlet

3. DispatcherServlet → HandlerAdapter → 해당 Controller의 메소드 탐색하여 실행

4. Controller → HandlerAdapter로 ModelAndView 객체 반환.

5. HandlerAdapter → DispatcherServlet로 ModelAndView 객체 반환.

6. DispatcherServlet → ViewResolver → View

정리



Spring 프레임워크란?

Spring?

▼ 구조

Spring Core

Spring Core는 Spring Container을 의미합니다. core라는 말 그대로 Container는 Spring Framework의 핵심이며 그중 핵심은 Bean Factory Container입니다. 그 이유는 바로 Bean Factory는 IOC패턴을 적용하여 객체 구성부터 의존성 처리까지 모든 일을 처리하는 역할을 하고 있기 때문입니다.

Spring Context

Spring context는 Spring Framework의 context 정보들을 제공하는 설정 파일입니다. Spring Context에는 JNDI, EJB, Validation, Scheduling, Internalization 등 엔터프라이즈 서비스들을 포함하고 있습니다.

Spring AOP

Spring AOP module은 Spring Framework에서 관점지향 프로그래밍을 할 수 있고 AOP를 적용 할 수 있게 도와주는 Module입니다. 해당 AOP에 대한 내용은 위에서 설명 했기 때문에 넘어 가도록 하겠습니다.

Spring DAO

DAO란 Data Access Object의 약자로 Database Data에 접근하는 객체입니다. Spring JDBC DAO는 추상 레이어를 지원함으로써 코딩이나 예외처리 하는 부분을 간편화 시켜 일관된 방법으로 코드를 짤 수 있게 도와줍니다.

Spring ORM

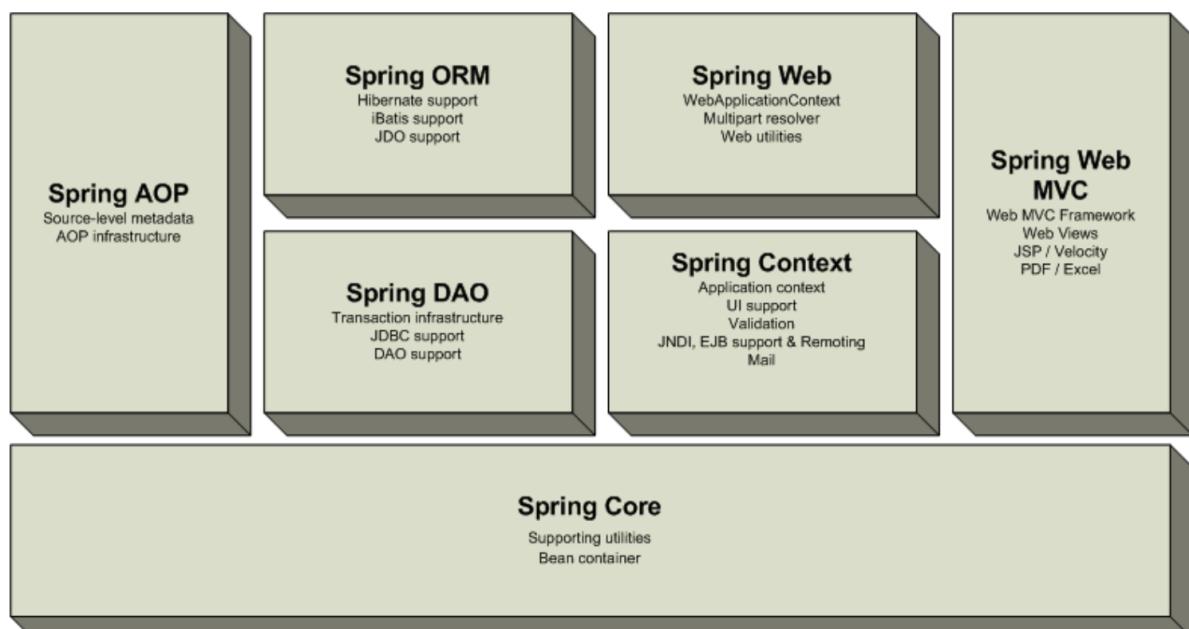
ORM이란 Object relational mapping의 약자로 간단하게 객체와의 관계 설정을 하는 것입니다. Spring에서는 Ibatis, Hibernate, JDO 등 인기있는 객체 관계형 도구(OR 도구)를 사용 할 수 있도록 지원합니다.

Spring Web

Spirng에서 Web context module은 Application module에 내장되어 있고 Web기반의 응용프로그램에 대한 Context를 제공하여 일반적인 Web Application 개발에 필요한 기본적인 기능을 지원합니다. 그로인해 Jakarta Structs 와의 통합을 지원하고 있습니다.

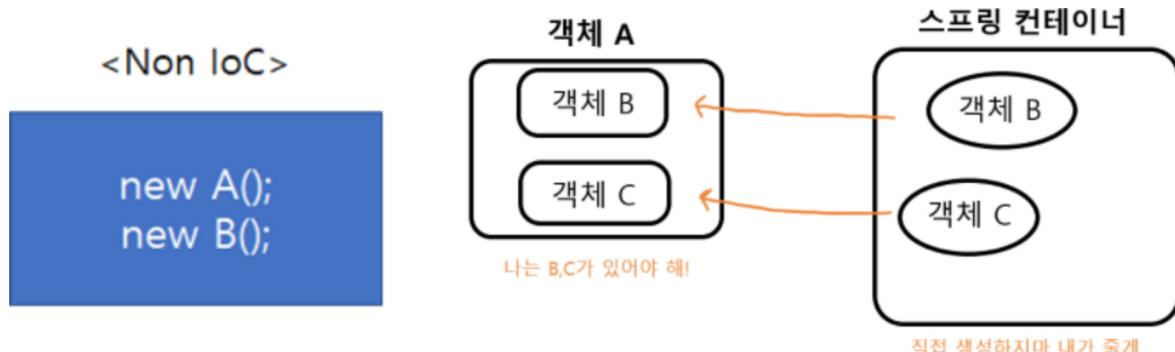
Spring MVC

Spring에서는 MVC에서는 Model2 구조로 Application을 만들 수 있도록 지원합니다. MVC (Model-View-Controller) 프레임 워크는 웹 응용 프로그램을 작성하기 위한 완전한 기능을 갖춘 MVC를 구현합니다. MVC 프레임 워크는 전략 인터페이스를 통해 고급 구성 가능하며 JSP, Velocity, Tiles, iText 및 POI를 포함한 수많은 뷰 기술을 지원하고 있습니다.



- Spring은 JAVA 기반의 웹 프레임워크, 여러 기술들의 모음이라고 할 수 있다.
- 웹 프로그래밍을 위한 기능들을 추상적으로 정의해놓은 틀로, 개발자는 이러한 틀 안에서 필요한 기능만 구현하면 된다.
- 이렇게 기능을 추상화해놓은 틀 하나하나를 모듈이라고 하며, 그런 기능들을 모아둔 라이브러리라고 생각하면 된다.

특징

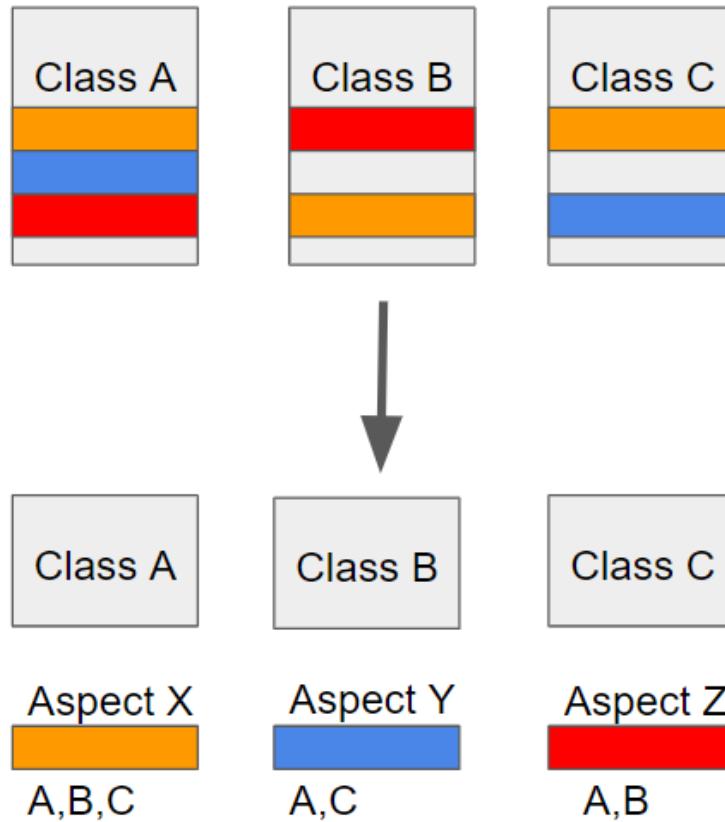


DI(의존성 주입, Dependency Injection)

- 어떤 기능을 만들어서 필요할 때마다 주입해서 사용하는 것
- Spring은 객체를 주체 클래스에서 직접 생성하는 것이 아니라 외부(Spring)에서 생성하여 사용하려는 주체 클래스에 주입시키는 방식이다.
- 주체클래스에서 직접 객체를 생성하는 경우 의존도가 높지만, 외부에서 생성하여 관리하는 Spring의 경우 객체간의 의존도와 결합도가 줄어든다.

AOP(관점 지향 프로그래밍, Aspect-Oriented Programming)

- 관점을 기준으로 다양한 기능을 분리하여 보는 프로그래밍이다.
- 관점 지향은 어떤 로직을 기준으로 핵심적인 관점, 부가적인 관점으로 나누어서 보고 그 관점을 기준으로 모듈화 하겠다는 것이다.
- 예를 들어 핵심적인 관점은 비즈니스 로직이 될 수 있고, 부가적인 관점은 핵심 로직을 실행하기 위해 행해지는 데이터베이스 연결, 로깅, 파일 입출력 등이 될 수 있다.
- AOP는 흩어진 관심사(Crosscutting Concerns)를 모듈화 할 수 있는 프로그래밍 기법이다



- 그림과 같이 클래스 A, B, C에서 공통적으로 나타나는 색깔 블록은 중복되는 메서드, 필드, 코드 등이다. 이때 예를 들어 클래스 A의 주황색 블록 부분을 수정해야 한다면 클래스 B, C의 주황색 부분도 일일이 찾아 수정해야 한다. 이는 유지보수를 어렵게 만든다. 이런 식으로 소스 코드상에서 계속 반복해서 사용되는 부분들을 **흩어진 관심사(Crosscutting Concerns)**라고 한다.
- 결국 AOP에서 각 관점을 기준으로 로직을 모듈화한다는 것은 **흩어진 관심사를 모듈화하겠다는 의미**다. 그림과 같이 주황색, 파란색, 빨간색 블록처럼 모듈화 시켜놓고 어디에 적용시킬지만 정의해 주면 되는 것이다. 이때 모듈화 시켜놓은 블록을 **Aspect**라고 한다.

IOC(제어의 역전, Inversion Of Control)

- 일반적으로 처음에 배우는 자바 프로그램에서는 각 객체들이 프로그램의 흐름을 결정하고 각 객체를 직접 생성하고 조작하는 작업(객체를 직접 생성하여 메소드 호출)을 했다. 즉, 모든 작업을 사용자가 제어하는 구조였다.
 - 예를 들어 A 객체에서 B 객체에 있는 메소드를 사용하고 싶으면, B 객체를 직접 A 객체 내에서 생성하고 메소드를 호출.
- 하지만 IOC가 적용된 경우, 객체의 생성을 특별한 관리 위임 주체(Spring)에게 맡긴다. 이 경우 사용자는 객체를 직접 생성하지 않고, 객체의 생명주기를 컨트롤하는 주체는 다른 주체(Spring)가 된다. 즉, 사용자의 제어권을 다른 주체에게 넘기는 것을 IOC(제어의 역전)라고 한다.
- 정리
 - 일반 JAVA: `클래스 내부의 객체 생성 -> 의존성 객체의 메소드 호출`
 - Spring: `스프링에게 제어를 위임하여 스프링이 만든 객체를 주입 -> 의존성 객체의 메소드 호출` 구조.

- 스프링에서는 모든 의존성 객체를 스프링이 실행될 때 만들어주고 필요한 곳에 주입해준다.

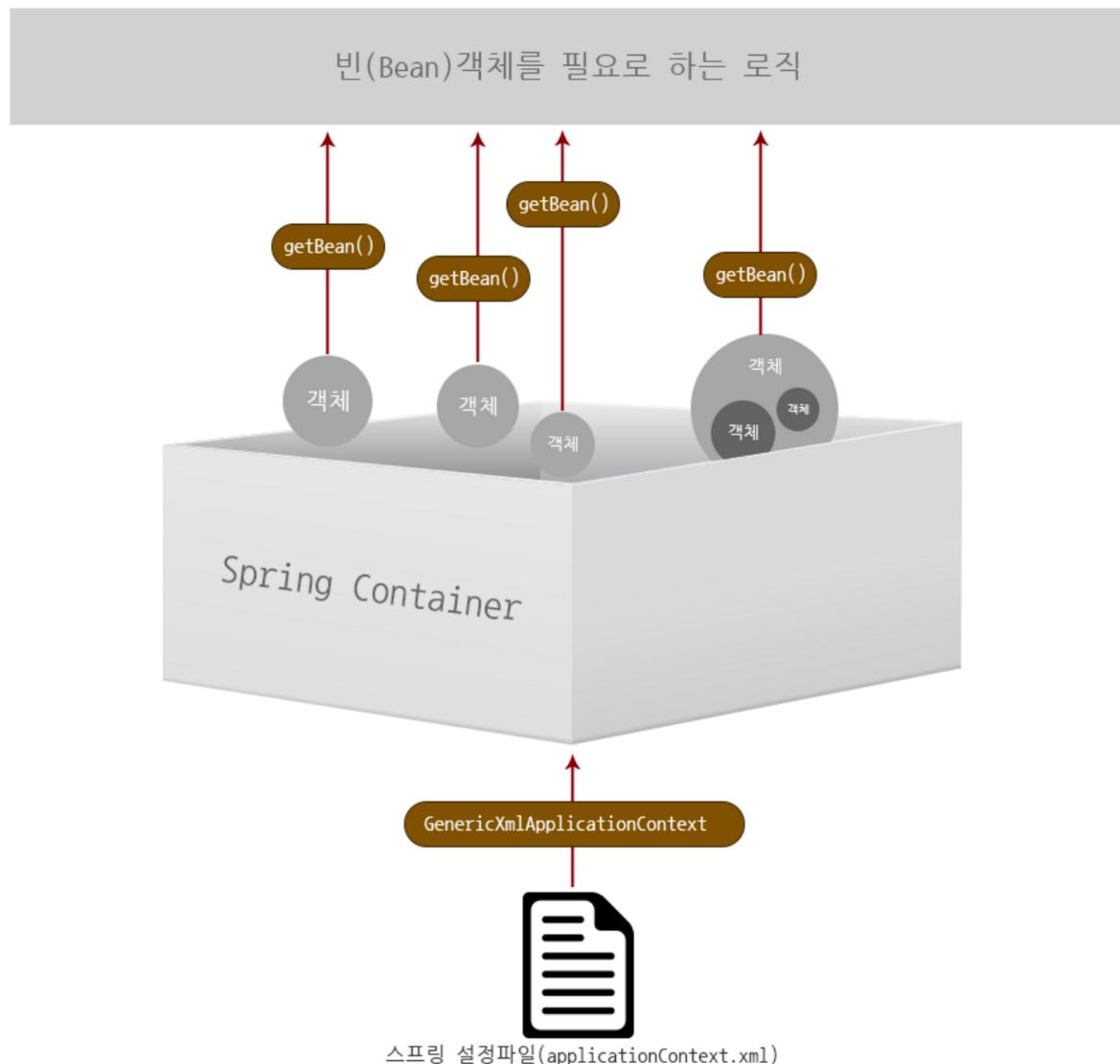
MVC

- Model, View, Control로 코드를 분리해서 구조화하여 프로그래밍하는 방식
- (뒤에서 설명)

간편하다 (빠르고 쉬움, 보안성, WAS내장)

- 쿠키나 세션 처리, 로그인/로그아웃 처리, 권한 처리, 데이터베이스 처리 등 웹 프로그램을 위해 만들어야 할 기능이 프레임워크에 포함되어있기 때문에 기능을 익혀 사용하기만 하면 된다.
- 프레임워크에 보안 기능이 내장되어있기 때문에, 보안 공격에 대한 코드를 따로 짤 필요가 없다.
- 스프링부트에는 내장 톰캣 서버를 지원한다.

Spring 주요 개념 - Container와 Bean



Container

- 스프링에서 객체(Bean)를 생성하고 조립하는 주체
- 설정파일인 XML문서를 토대로 Spring Container에서 객체(Bean)을 생성한다.
- JAVA 파일에 개발 시, 이미 만들어져있는 객체를 Container에서 꺼내서 사용하기만 하면 된다.

Bean

- 컨테이너를 통해 생성된 객체를 빈(Bean)이라고 부른다.

Spring Container 생성 및 Bean객체 호출 과정

(참고) pom.xml

- 메이븐 설정파일로, 각종 라이브러리를 연결해준다.
- 필요한 라이브러리를 명시하면 이를 다운로드해서 사용한다.

applicationContext.xml 설정파일 (Bean 생성)

- maven 설정파일인 pom.xml처럼 Spring 설정파일 있다.
- 이 파일 안에 Bean의 정보들이 명시되어있고, 이를 토대로 Container에 Bean을 생성한다.
- 스프링은 컨테이너 안에 객체(Bean)를 생성하여 모두 모아두고, 필요할 때마다 꺼내 쓰는데, 스프링 컨테이너에 Bean을 만들 때 그 토대가 되는 문서이다.
- 구성

```
<bean id="{id}" class="{package}.{classname}"/>
```

- id는 키값을 입력
- class는 해당 bean과 매핑될 class 풀네임을 입력
- 예시

```

application-context.xml × context-api.xml × context-datasource.xml × context-ifo.xml × pom.xml × context-mongo.xml × conte
<?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">

    <!-- 로그 처리 설정 -->
    <bean id="ifoApiLogHandleListener" class="com.lottecard.framework.api.IfoApiLogHandleListener">
        <property name="loggerName" value="com.lottecard.info" />
    </bean>

    <!-- API Request Generator -->
    <bean id="apiRequestGenerator" class="com.lottecard.lp.pers.comm.DefaultApiRequestGenerator">
        <property name="systemName" value="LPAPP001" />
        <property name="srcSysCd" value="LLP" />
    </bean>

    <!-- API 처리 핸들러 -->
    <bean id="ifoApiHandler" class="com.lottecard.framework.api.outbound.IfoApiHandler">
        <property name="maxConnections" value="100" />
        <property name="apiGatewayUrl" value="#{cmProp['api.gateway.url']}"/>
        <property name="connectionRequestTimeout" value="5000" />
        <property name="socketTimeout" value="5000" />
    </bean>

    <!-- API 서비스 설정 -->
    <bean id="ifo ApiService" class="com.lottecard.framework.api.outbound.service.impl.Ifo ApiServiceImpl">
        <property name="ifoApiHandleListener" ref="ifoApiLogHandleListener" />
        <property name="ifoApiHandler" ref="ifoApiHandler" />
        <property name="apiRequestGenerator" ref="apiRequestGenerator" />
    </bean>

    <!-- 로컬 테스트를 위한 개발기 설정 -->
    <beans profile="dev">
        <!-- API Bypass 처리 핸들러 -->
        <bean id="ifoApiBypassHandler" class="com.lottecard.framework.api.outbound.dev.IfoApiBypassHandler">
            <property name="apiGatewayUrl" value="#{cmProp['api.gateway.url']}"/>
            <property name="logEnable" value="true" />
        </bean>
    </beans>
</beans>

```

GenericXmlApplicationContext 객체 (컨테이너 생성)

```
GenericXmlApplicationContext ctx = new GenericXmlApplicationContext("classpath:applicationContext.xml");
```

- `GenericXmlApplicationContext` 는 컨테이너에 접근하기 위한 데이터타입이다.
 - 파라미터로 `applicationContext` resource를 적어준다. (`applicationContext.xml`)
 - 해당 xml 설정파일을 토대로 객체를 생성한다는 의미.
- `applicationContext.xml` 파일을 이용하여 Container를 만들었다고 생각하면 된다 (`ctx == Container`)
- 컨테이너를 생성하면, Bean 생성은 컨테이너가 알아서 한다.
 - (소멸도 알아서 관리, Spring Container와 Bean의 생명주기는 같다)

getBean() (컨테이너를 통해 Bean 접근)

```

{변수} = ctx.getBean("{id}", {데이터타입});

//ex
customClass cust = ctx.getBean("id", customClass.class);
cust.method();

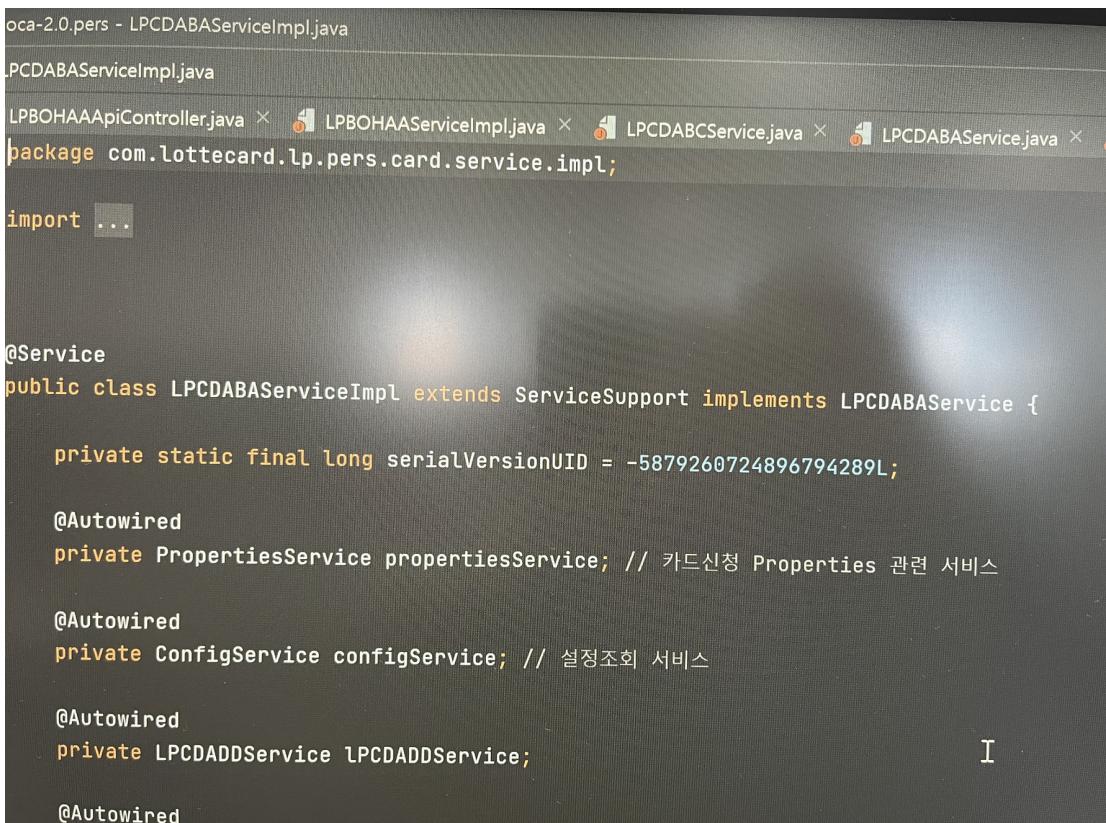
ctx.close();

```

- `getBean()` 메서드를 이용하여 객체(Bean)를 가져와서 사용한다.

요즘방식

- 사실 앞서 소개한 방식은 기본 Spring의 동작과정이다.
- **스프링은 원래 XML에만 빈 정의가 가능했지만, 스프링 2.5버전부터는 어노테이션을 이용한 의존성 주입이 가능해졌다.**
 - `@Component` 가 사용된 클래스들은 스프링 빈으로 등록되고 자동으로 빈 탐색의 대상이 된다.
 - `@Controller, @Service`는 `@Component`의 구체화된 형태로 해당 어노테이션이 사용된 곳은 **`@Component`과 마찬가지로 자동으로 스프링 빈으로 등록된다.**



```

oca-2.0.pers - LPCDABAServiceImpl.java
LPDABAServiceImpl.java
LPBOHAAApiController.java × LPBOHAAServiceImpl.java × LPCDABCService.java × LPCDABAService.java ×
package com.lottecard.lp.pers.card.service.impl;

import ...

@Service
public class LPCDABAServiceImpl extends ServiceSupport implements LPCDABAService {

    private static final long serialVersionUID = -5879260724896794289L;

    @Autowired
    private PropertiesService propertiesService; // 카드신청 Properties 관련 서비스

    @Autowired
    private ConfigService configService; // 설정조회 서비스

    @Autowired
    private LPCDADDService LPCDADDService;

    @Autowired
}

```

```

import ...

/**
 * <h3>신용카드 > 기존발급회원 Controller</h3>
 * @author PJ02079
 * @since 2021-05-10
 * @version 2021-05-10, PJ02079, 최초등록
 */
@Api(tags = "05_카드 > 카드안내/신청 > 신용카드 > 신청구분(기존발급회원)")
@RequestMapping("/app/api")
@Controller
public class LPCDABCApiController extends ControllerPublicSupport {

    private static final long serialVersionUID = -6087457257746469906L;

    @Autowired
    private LPCDABCSERVICE lPCDABCSERVICE;

    /**
     * <h3>신청구분 화면 초기 설정</h3>
     * @param LPCDABC13OPVO
     * @return ModelAndView
    
```

- 옛날방식의 {컨테이너}.getBean()으로 Bean 가져오는 것을 이제는 그냥 변수선언할 때 위에 @Autowired 붙이면 알아서 같은 타입의 Bean객체를 가져오는 것!!

Spring 프로젝트 구조

웹 프로그래밍 설계 모델

Model1 방식

- 서비스, DAO, view가 한 파일에 작성된다.

Model1



- 과정

1. 브라우저(클라이언트)에서 서버(WAS)로 요청(request)
2. 서버에서 처리 → DB 접근하여 가져옴
3. 가져온 데이터를 가공하여 브라우저에 응답(response) - html 형태로 응답

- 장점

- html파일 안에 java코드와 태그 등을 한 문서에 넣어서 개발속도가 빠름

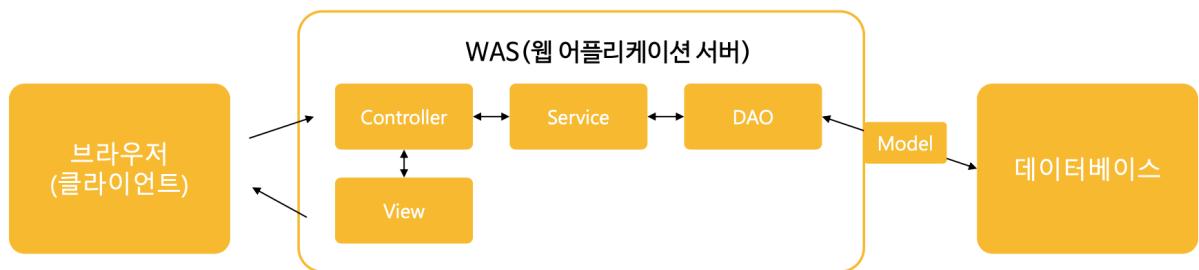
- 단점

- 여러 언어가 한 파일에 작성 → 유지보수 차원에서 좋지 않다.

Model2 방식, MVC

- Model1의 단점을 보완한 방식, **MVC**
- Spring에 기본적으로 사용되는 모델

Model2



- 과정

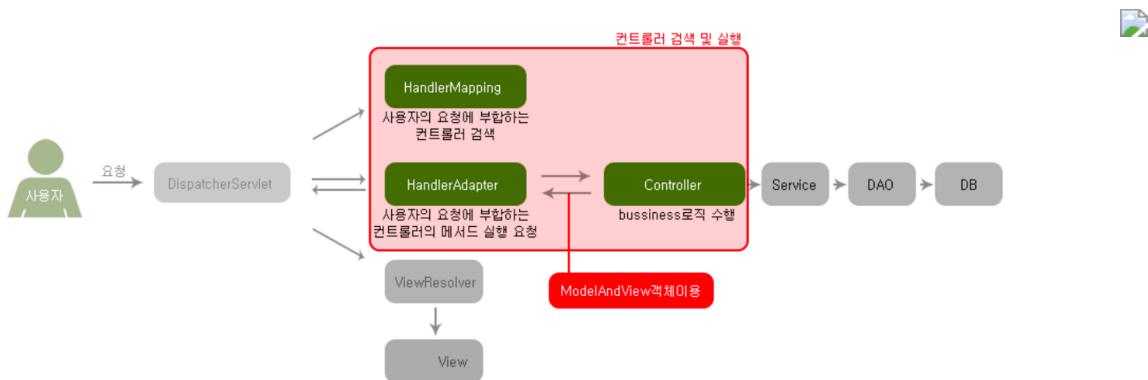
1. 브라우저에서 요청을 Controller가 받아서 해당하는 Service로 연결해준다.
 - Service는 하나하나의 기능들이라고 보면 된다.
2. DAO를 이용한 Model이라는 객체로 DB와 통신한다.
3. 통신하여 얻은 데이터를 다시 컨트롤러에 넘긴다
4. 컨트롤러는 View라는 객체로 클라이언트에 응답한다. (JSP파일)

- 장점

- 기능별로 분리되어있기 때문에 유지보수가 수월하다.

Spring MVC 프레임워크 구조 및 흐름

- 스프링 없이 구현했으면 HandlerAdapter, ViewResolver, HandlerMapping가 하는 기능들도 다 직접 처리를 했어야 하는 부분들이다.
- 스프링을 사용함으로써 우리는 Controller 부분만 신경쓸 수 있게 되었다.



1. 사용자의 모든 요청은 **DispatcherServlet** 이 처음 받는다.

- 클라이언트에서 요청을 받는 객체, 프론트컨트롤러라고도 불리며, 사실상 스프링에서 가장 중심이 되는 중개자 역할.

2. **DispatcherServlet** → **HandlerMapping** → **DispatcherServlet**

- 많은 Controller들 중 가장 적합한 Controller를 선택해주는 역할
- 알맞은 Controller를 선택하고 다시 **DispatcherServlet**으로 온다.
 - `@Controller` 어노테이션이 붙은 클래스 대상 탐색
 - 요청받은 URL을 분석하여 해당 Controller를 찾아준다.

3. **DispatcherServlet** → **HandlerAdapter** → 해당 **Controller** 의 메소드 탐색하여 실행

- 한 Controller 안에 있는 많은 Method들 중에 클라이언트의 요청에 적합한 Method를 찾아 실행해주는 역할
 - 메소드에 붙어있는 `@RequestMapping` 어노테이션으로 탐색

4. **Controller** → **HandlerAdapter**로 **ModelAndView** 객체 반환.

- Controller객체의 메소드가 실행된 후 Controller객체는 HandlerAdapter객체에 ModelAndView객체를 반환하는데, 여기에는 응답에 필요한 데이터와 뷰(JSP파일)가 담겨있다.

5. **HandlerAdapter** → **DispatcherServlet**로 **ModelAndView** 객체 반환.

6. **DispatcherServlet** → **ViewResolver** → **View**

- ModelAndView정보를 토대로 적합한 JSP 문서(view)를 찾고, JSP문서로 응답.

정리

- Spring 프레임워크 설명 및 특징
- 주요개념인 Container와 Bean과 이들의 동작과정

- 웹프로그래밍 설계모델 - MVC 패턴
- 스프링 프로젝트의 폴더구조
- 스프링 MVC 프레임워크의 구조 및 흐름