**Spring 필기**

**1 스프링 개요**

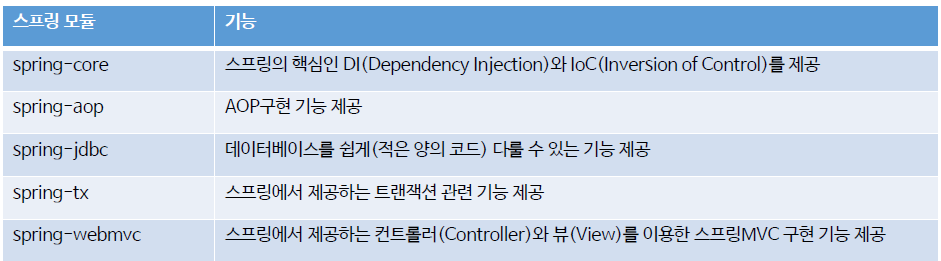
* 1. **스프링 프레임워크**

스프링 프레임워크는 주요 기능으로 DI, AOP, MVC, JDBC 등을 제공한다.

프레임워크는 개발자들의 업무를 도와주는 틀. (안전 운전만 할 수 있도록 해주는 네비게이션)

프레임워크가 없다면 모든 기능을 개발자가 고민, 개발해야 하는 번거로움이 있다.

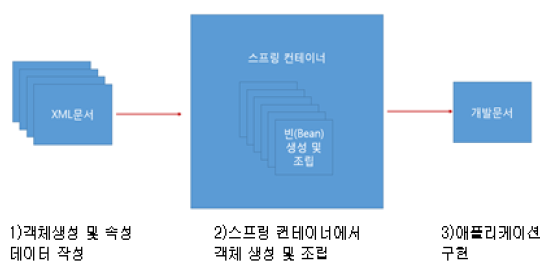
**1.2 스프링 프레임워크 모듈**



모듈을 사용하려면, 모듈에 대한 의존설정을 개발 프로젝트의 XML 파일 등을 이용해서 개발자가 직접 하면 된다.

* 1. **스프링 컨테이너(IoC)**

스프링에서 객체를 생성하고 조립하는 컨테이너(Container)로, 컨테이너를 통해 생성된 객체를 빈(Bean)이라고 한다.



**3 스프링 프로젝트 생성**

**3.1 프로젝트 생성**

Maven Project를 생성한다.

**3.2 pom.xml 작성**

**3.3 폴더 및 pom.xml 파일의 이해**

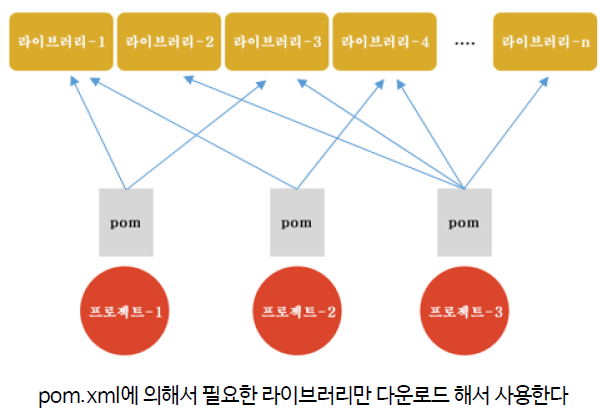
ch\_xx\_project/src/main/java 폴더 : 개발자가 자바 언어를 이용해 실제로 개발하는 부분

ch\_xx\_project/src/main/resources 폴더 : 개발함에 있어서 여러 보조적인 역할을 하는 파일들이 모인 부분. 스프링 설정 파일(XML) 또는 프로퍼티 파일 등이 관리되는 부분이다.

src/main 폴더가 java와 resources 폴더로 나뉜다는 것을 주의하자. 스프링 프레임워크의 기본 구조를 이루는 폴더로, 개발자는 이대로 폴더를 유지해야 한다.

pom.xml 파일은 메이븐 설정파일로, 메이븐은 라이브러리를 연결해주고, 빌드를 위한 플랫폼이다.

내 프로젝트에 필요한 라이브러리를 포함시켜주는 파일.



**4 처음 해보는 스프링 프로젝트**

**4.1 Java 파일을 이용한 프로젝트 실행**

평범한 java 파일을 생성해서 실행한다.

**4.2 우선 따라해보는 스프링 프로젝트**

resources에 xml 파일로 컨테이너와 bean을 설정해준다. 이렇게 해주면 스프링 콘테이너에 bean 객체가 생성이 되기 때문에 코드에서 일일이 객체를 new로 생성하지 않아도 된다.

**5 또 다른 프로젝트 생성 방법**

**5.1 폴더(java, resources)와 파일(pom.xml) 만들기**

직접 ch\_05\_ownproject-src-main-java, resources 폴더 만들기

pom.xml도 에디터를 이용해 만든다.

**5.2 이클립스에서 import하기**

이클립스에서 ch\_05를 import해서 java파일 생성 후 테스트 해보기

**6 DI (Dependency Injection)**

**6.1 DI(Dependency Injection) 란?**

DI, 의존 주입 : 객체를 만들어서 외부에서 주입하는 방식.

객체지향 프로그램의 유연성

1) 배터리 일체형 장난감 : 배터리가 떨어지면 장난감을 새로 구해야 한다.

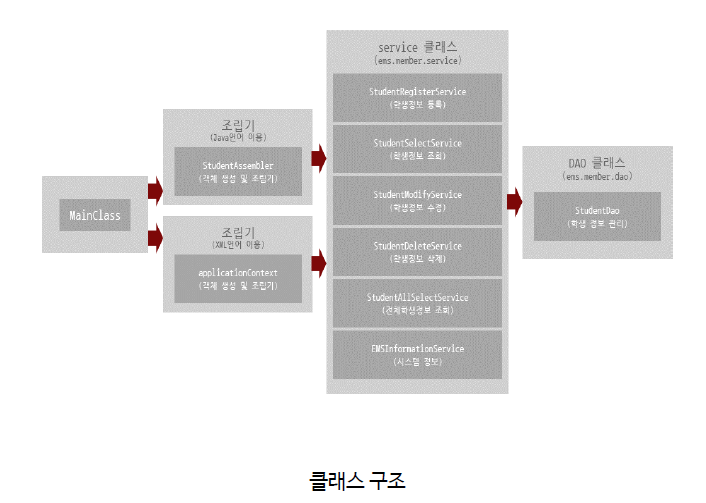
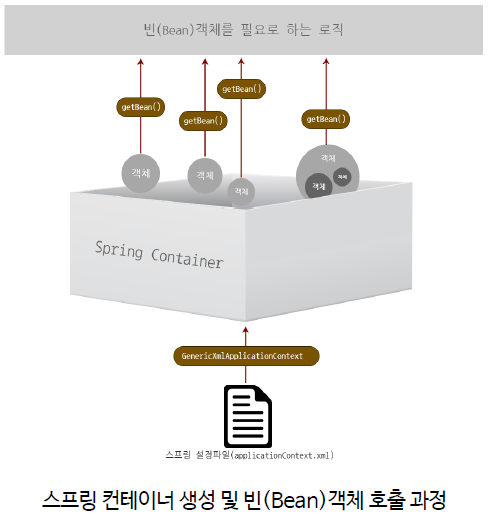
생성자에서 배터리를 생성한 후 변경 불가능

2) 배터리 분리형 장난감 : 배터리가 떨어지면 배터리만 교체하면 된다.

2-1) 생성자에서 장난감만 만들고 배터리를 setter로 설정

2-2) 생성자에서 배터리도 설정한 후 setter로 설정 -> 가장 유연하다.

**6.2 스프링 DI 설정 방법**



스프링 컨테이너에 빈(bean) 객체가 다 모여있다. 따라서 컨테이너에서 객체끼리 의존 주입이 되어있어야 한다.

**7 다양한 의존 객체 주입**

**7.1 생성자를 이용한 의존 객체 주입**

<constructor-arg> 태그 이용해서 매개변수 설정.

**7.2 setter를 이용한 의존 객체 주입**

property 태그에 멤버 변수를 name, 값을 value로 해서 설정

**7.3 List 타입 의존 객체 주입**

property와 name은 그대로 쓰고 <list>태그를 이용해 그 안에 <value>를 여러개 쓴다.

**7.4 Map 타입 의존 객체 주입**

<map>으로 묶고 <entry>로 묶어서 key와 value를 설정한다.

**8 스프링 설정 파일 분리**

**8.1 스프링 설정 파일 분리**

설정 파일이 너무 길어지면 비효율적이므로 효율적으로 관리하기 위해 설정 파일을 분리한다.

applicationContext.xml에 모두 설정해둔 것을 기능별로 나눠 appCtx1.xml, appCtx2.xml, appCtx3.xml으로 분리한다. 이름도 기능에따라 설정하는 게 좋다.

main 함수에서는 string 배열을 이용해 각 xml 파일을 로드한다.

import 태그를 이용해 Ctx1.xml Ctx2.xml을 import해서 한 파일로 xml을 합칠수도있다. 보통 배열로 사용하는 방법을 더 선호한다.

**8.2 빈(Bean)의 범위**

싱글톤(Singleton) : 스프링 컨테이너에 생성된 빈(Bean) 객체의 경우 동일한 타입에 대해서는 기본적으로 **한 개만 생성**이 되며, getBean() 메소드로 호출될 때 동일한 객체가 반환된다.

-> 컨테이너가 이미 객체를 한 개씩 생성했기 때문에, 생성된 동일한 객체를 반환하게 된다.

프로토타입(Prototype) : 싱글톤 범위와 반대의 개념. 프로토타입의 경우 개발자는 별도로 설정을 해 줘야 하는데, 스프링 설정 파일에서 빈(Bean)객체를 정의할 때 scope 속성을 명시해 줘야한다.

ex) <bean id="classA" class="com.ClassA" **scope="prototype"**>

getBean으로 객체를 참조할 때마다 새로운 객체를 생성하게 된다.

싱글톤이 디폴트이므로 필요할 때만 프로토타입을 설정한다.

**9 의존객체 자동 주입**

**9.1 의존객체 자동 주입이란?**

스프링 설정 파일에서 의존 객체를 주입할 때 <constructor-org> 또는 <property> 태그로 의존 객체 대상 객체를 명시하지 않아도, **스프링 컨테이너가 자동으로** 필요한 의존 대상 객체를 찾아서 의존 대상 객체가 필요한 객체에 주입해주는 기능이다.

구현 방법은 @Autowired, @Resource 어노테이션을 이용해서 쉽게 구현이 가능하다.

**9.2 @Autowired**

주입하려고 하는 **객체의 타입이 일치하는 객체**를 자동으로 주입한다.

생성자 위에 @Autowired를 쓴 후, xml 파일에 namespace들과, <context:annotation-config/> 태그를 추가하면, <constructor-arg ref=" "/>태그나 <property> 태그를 쓰지 않아도 된다.

생성자가 아닌 메소드나 변수에 @Autowired를 사용하려면 디폴트 생성자를 꼭 선언해 줘야한다.

**9.3 @Resource**

주입하려고 하는 **객체의 이름이 일치하는 객체**를 자동으로 주입한다.

@Resource는 생성자에는 쓰지 못한다. 메소드나 프로퍼티에만 사용이 가능하다. 마찬가지로 디폴트 생성자가 필수이다.

**10 의존객체 선택**

**10.1 의존객체 선택**

다수의 빈(Bean) 객체 중 의존 객체의 대상이 되는 객체를 선택하는 방법.

동일한 객체가 2개 이상인 경우 스프링 컨테이너는 **자동 주입 대상 객체**를 판단하지 못해서 Exception이 발생된다.

<qualifier> 태그를 사용해서 여러 객체 중 사용할 객체를 선택할 수 있다. value 속성으로 이름 설정이 가능하다.

클래스의 멤버 변수 이름과 bean 객체의 id 이름이 같다면 qualifier 태그로 설정하지 않아도 예외가 발생하지 않는다. -> 그다지 좋은 방법은 아니니 그냥 qualifier 태그를 넣어주자.

**10.2 의존객체 자동 주입 체크**

java 파일에선 @autowired로 자동 주입을 해달라고 하고, 정작 xml 파일에서는 bean 객체를 설정하지 않는다면 예외가 발생한다.

이런 경우 @Autowired(required=false)로 설정해두면 된다.

하지만 이런건 초보들이나 하는 실수이니 거의 쓰이지는 않는다.

**10.3 @Inject**

@Autowired와 비슷하게 @Inject 어노테이션을 이용해서 의존 객체를 자동 주입할 수 있다.

차이점이라면 @Inject는 required 속성을 지원하지 않는다는 것이다.

required 속성이 거의 쓸모 없기 때문에, @Inject와 @Autowired는 사실상 동일하다고 해도 무방하다.

@Inject는 @qualifier 대신에 @Named(value="id")를 사용할 수 있다.

**11 생명 주기(Life Cycle)**

**11.1 스프링 컨테이너 생명주기**

1) GenericXmlApplicationContext ctx = new GenericXmlApplicationContext("classpath:appCtx.xml");

을 실행할 때 스프링 컨테이너가 생성되며 동시에 Bean 객체가 생성된다. 즉 스프링 컨테이너와 빈 객체의 생성 시점은 동일하다.

2) getBean()등을 이용해서 빈 객체를 이용한다.

3) ctx.close()로 스프링 컨테이너를 소멸하며 빈 객체 또한 소멸된다.

빈 객체가 생성되고 소멸될 때 특정한 작업을 하는 방법이 두 가지 있는데, 한 가지는 11.2의 인터페이스를 이용하는 방법이고, 나머지 방법은 11.3의 메소드의 속성을 이용하는 방법이다.

**11.2 빈(Bean)객체 생명주기**

빈 객체의 생명주기는 스프링 컨테이너의 생명주기와 같이 한다.

스프링 컨테이너는 InitializingBean과 DisposableBean이라는 인터페이스를 제공한다.

InitializingBean의 afterPropertiesSet() 메소드는 빈 객체 생성 시점에 호출되고,

DisposableBean의 destroy() 메소드는 소멸 시점에 호출된다.

**11.3 init-method, destroy-method 속성**

bean태그에 해당 속성을 쓴 후 속성값에 메소드 이름을 적으면 된다.

인터페이스 방식과 메소드 방식 중 원하는 방식을 골라서 사용하면 된다.

**12 어노테이션을 이용한 스프링 설정**

**12.1 XML 파일을 Java파일로 변경하기**

XML 파일을 이용해서 스프링 설정파일을 제작을 하던 방식에서 Java 파일로 제작하는 방법.

마찬가지로 하나의 Java 파일로 설정할 수도 있고, 여러 개의 파일로 나눈 후 import 할 수도 있다.

기존에는 xml 파일로 스프링 컨테이너와 빈 객체를 생성했으나, java 파일의 어노테이션을 이용해서 할 수 있다.

@Configuration 어노테이션을 통해 설정파일임을 알릴 수 있다.

@Bean 태그를 통해 빈 객체임을 알려야 한다.

AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext("config.class")

를 통해 컨테이너를 생성한다.

**12.2 Java 파일 분리**

하나의 config 파일로 설정파일을 통일하는 것 보다, 여러 개로 나누는 것이 유지, 보수에 효율적이다. 기능에 따라 설정파일을 나누는 것이 일반적이다. (ex. DaO, Service, DB, Util 등)

서로 의존된 객체는 @Autowired와 임시 객체 생성으로 해결할 수 있다. 그러면 스프링 컨테이너에서 자동 주입된다.

파일을 분리했을 때 context 객체는 config1.class, config2.class, config3.class로 객체 생성을 하면 된다.

**12.3 @Import 어노테이션**

@Configuration 아래에

@Import({config2.class, config3.class})를 통해 설정 파일을 하나로 합칠수도 있다.

public void config1{

}

**13 웹 프로그래밍 설계 모델**

**13.1 웹 프로그래밍을 구축하기 위한 설계 모델**

Model 1 방식

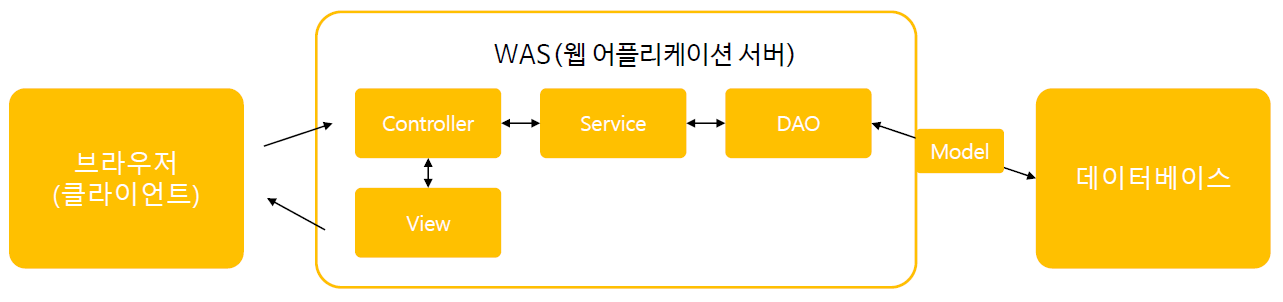


클라이언트와 서버사이의 요청 - 응답.

서버에서 하나의 문서에 HTML, Java코드, 태그 등이 합쳐저있다.

문서가 너무 장황해져서 유지보수가 힘들다.

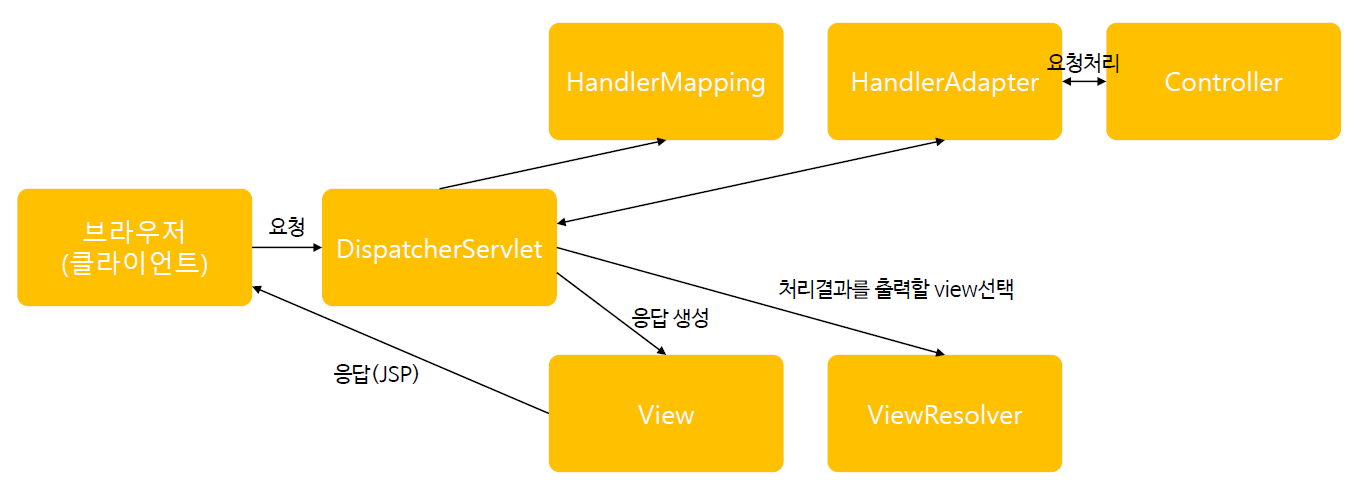
Model 2 방식



Controller, Service, View, DAO로 각각의 기능을 모듈화한다.

MVC가 웹 개발에서의 기본적인 방식으로 유지보수가 수월하다.

**13.2 스프링 MVC 프레임워크 설계 구조**



MVC의 기본적인 구조. 머리속에 넣어두자.

1. User의 요청을 DIspatcherServlet이 HandlerMapping에 보내고, HandlerMapping이 알맞은 Controller를 선택해준다.

2. HandlerAdapter가 Controller의 적합한 메소드를 선택

3. DispatcherServlet이 ViewResolver에 처리결과를 보낸다.

4. ViewResolver가 적합한 View를 선택한다.

5. View가 클라이언트에 응답한다.

**13.3 DIspatcherServlet 설정**

DispatcherServlet : web.xml에 서블릿을 매핑. 모든 요청을 처리하므로 / (root)로 url을 매핑한다.

DispatcherServlet을 서블릿으로 등록을 한 후 init-param(초기 파라미터)로 스프링 설정파일을 설정해줘야한다. 스프링 설정파일을 설정하지 않으면 디폴트 설정파일을 생성한다.

이렇게 생성한 스프링 컨테이너에 HandlerMapping, HandlerAdapter, VierResolver 등이 포함된다.

**13.4 Controller 객체 - @Controller**

스프링 설정파일에 <annotation-driven/> 태그를 포함해야한다.

컨트롤러로 사용할 클래스 이름 앞에 @Controller 어노테이션을 명시한다.

**13.5 Controller 객체 - @RequestMapping**

해당 메소드 앞에 @RequestMapping를 쓴 후 ("/~~")로 매핑할 url을 연결한다.

ex) @RequestMapping("/success")로 success 메소드 앞에 명시해두면, 해당 Controller/success로 요청이 올 시 이 메소드가 실행된다.

**13.6 Controller 객체 - Model 타입의 파라미터**

개발자는(Controller 객체) Model 객체에 데이터를 담아서 DispatcherServlet에 전달할 수 있다.

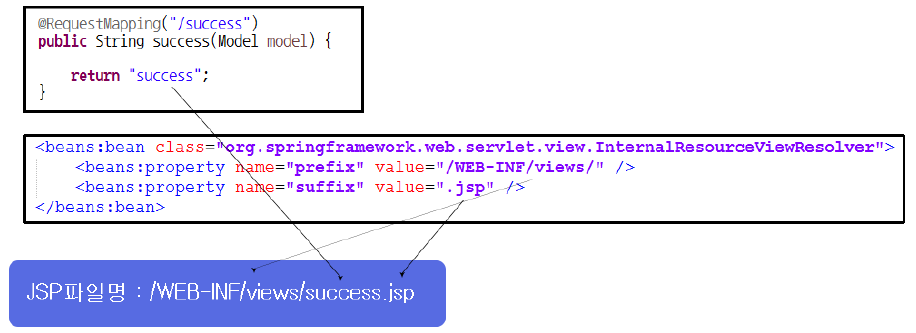
전달된 Model 데이터는 View에서 가공되어 클라이언트에 응답처리 된다.

**13.7 View 객체**

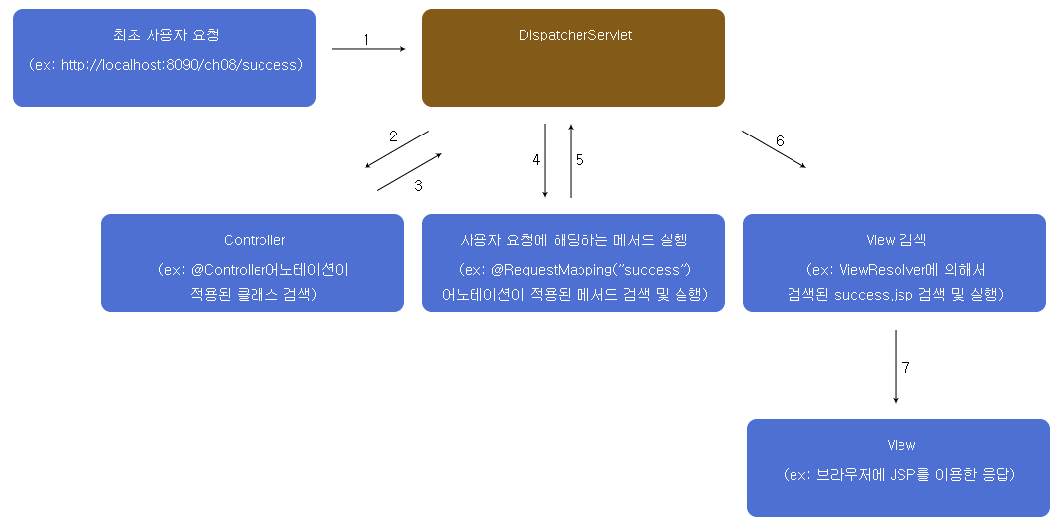
DispatcherServlet이 ViewResolver에게 위임하면,

스프링 설정파일에 InternalResourceViewResolver라는 Bean 객체를 생성한다.

Model 데이터 값과, 빈 객체에서 설정한 prefix, suffix 값을 합쳐서 적합한 View 파일로 매핑한다.



**13.8 전체적인 웹프로그래밍 구조**



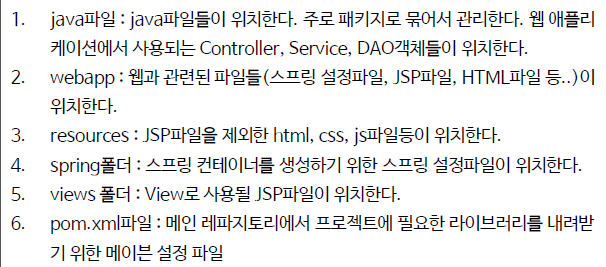
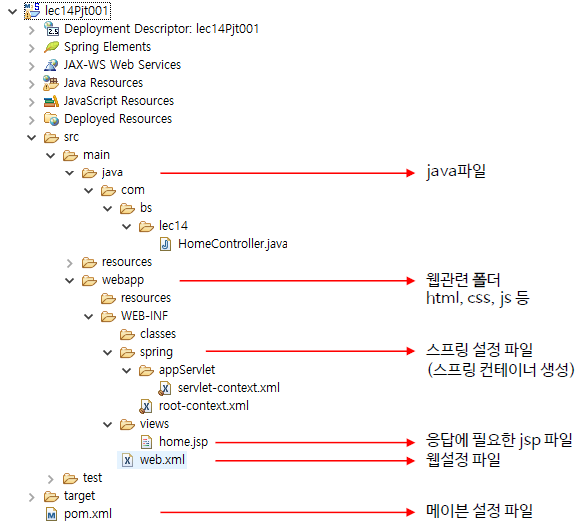
**14 스프링 MVC 웹 서비스**

웹 서버 (Tomcat) 연동 및 STS(Spring Tool Suit) 설치

**14.1 STS를 이용한 웹프로젝트 생성**

프로젝트를 생성할 땐 2단계 이상으로 생성해야한다. ex) com.xx.yy

**14.2 프로젝트 전체 구조**

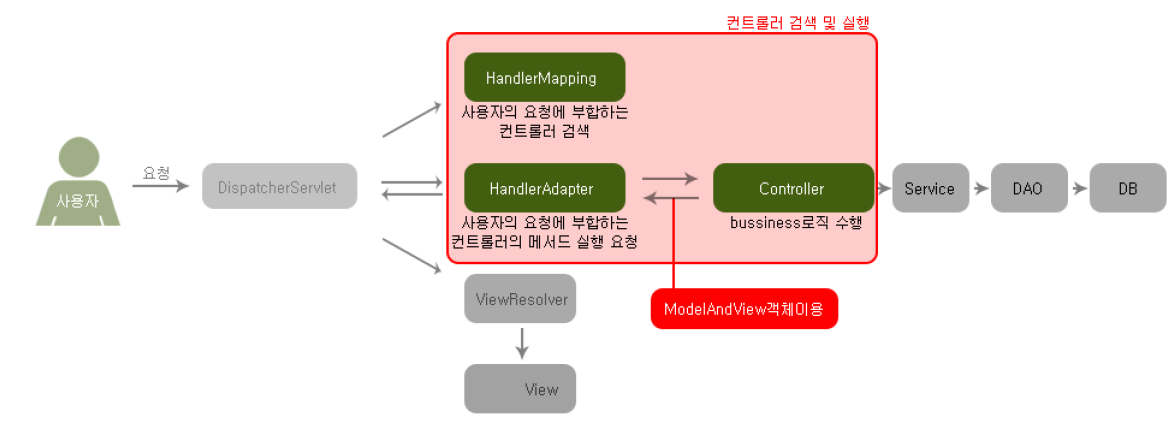


**14.3 web.xml**

최초 사용자의 요청이 발생하면 가장 먼저 DispatcherServlet이 요청을 받는다.

따라서 DispatcherSerlvet을 서블릿으로 등록해줘야한다. 그리고 모든 요청을 받기 위해 경로를 '/'로 설정한다.

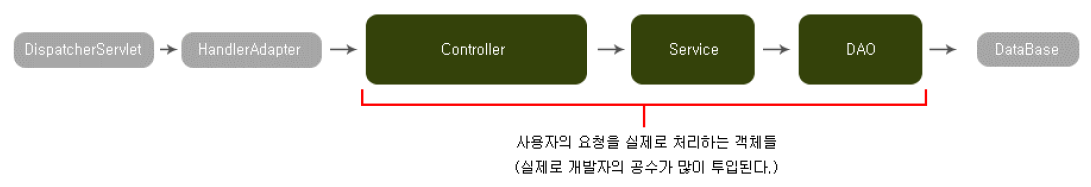
**14.4 DispatcherServlet**



**14.5 servlet-context.xml**

spring 설정 파일.

**14.6 Controller**

**14.7 View**

메소드 반환값이 "home"이라면 views/home.jsp를 사용자에게 응답한다.