Framework: 반제품이다. 정말 특수한 형태라면 직접 만들어서 사용하겠지만 그게 아니라면 이미 잘 만들어져 있는 반제품을 사용할 것인지?

\* Spring Framework란?

1. 큰 어플리케이션을 구축할 수 있는 가벼운 솔루션이다.

2. 원하는 부분만 가져다 사용할 수 있도록 모듈화가 잘되어져 있다. (모듈 20개)

3. ioC 컨테이너이다.

4. 선언적으로 트랜잭션을 관리할 수 있다.

5. 완전한 기능을 갖춘 MVC Framework를 제공한다.

6. AOP지원

\* Spring 데이터 엑세스

spring-jdbc: 자바 JDBC프로그래밍을 쉽게 할 수 있도록 기능을 제공한다.

spring-tx: 선언적 트랜잭션 관리를 할 수 있는 기능을 제공한다.

\* Spring 웹

spring-web: 멀티 파트 파일 업로드, 서블릿 리스너 등 웹 지향 통합 기능을 제공한다. HTTP클라이언트와 Spring의 원격 지원을 위한 웹 관련 부분을 제공한다.

spring-webmvc: Web-Servlet모듈이라고 불리며, Spring MVC및 REST웹 서비스 구현을 포함한다.

\*컨테이너란?

-인스턴스의 생명주기를 관리한다.

-생성된 인스턴스들에게 추가적인 기능을 제공한다.

cf) Servlet또한 WAS역할인 Tomcat이 서블릿 컨테이너를 가지고 있기 때문에 프로그래머가 인스턴스를 생성을 따로 해줄 필요 없다. Tomcat이 실제로 메모리에 올리고 실행하는 것을 하는 것이다. Servlet 컨테이너는 동일한 서블릿에 해당하는 요청을 받으면, 또 메모리에 올리지 않고 기존에 메모리에 올라간 서블릿을 실행

\*IOC란?

-Inversion of Control: 제어의 역전

-서블릿 클래스는 개발자가 만들지만, 그 서블릿의 메소드를 알맞게 호출하는 것을 WAS이다.

\*DI란?

-DI란 Dependency Injection의 약자로, 의존성 주입이란 뜻

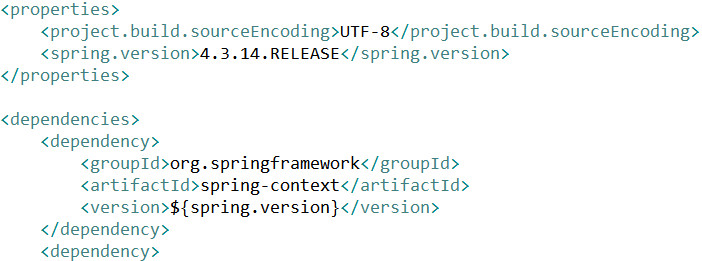
-DI는 클래스 사이의 의존 관계를 빈(Bean)설정 정보를 바탕으로 컨테이너가 자동으로 연결해주는 것을 말한다.

\*Spring에서 제공하는 IoC/DI 컨테이너

-BeanFactory: IoC/DI에 대한 기본 기능을 가지고 있다.

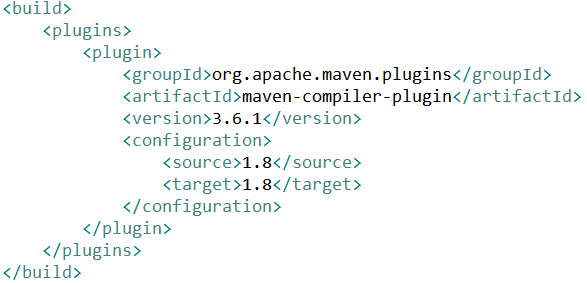
-ApplicationContext: BeanFactory의 모든 기능을 포함하며, 일반적으로 BeanFactory보다 추천된다. 트랜잭션처리, AOP등에 대한 처리를 할 수 있다.

\*pom.xml의 <properties></properties >에는 상수처럼 사용할 수 있도록 도와주는 것 작성



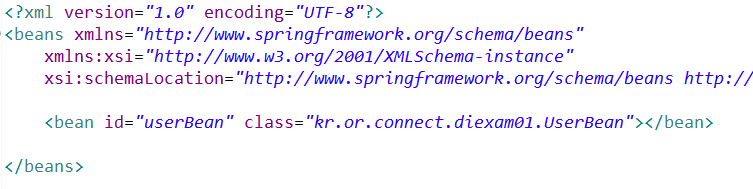
${spring.version}이 상수처리 하기 전에는 4.3.14.RELEASE였다.

JDK 버전을 1.8로 바꾸는 소스코드.



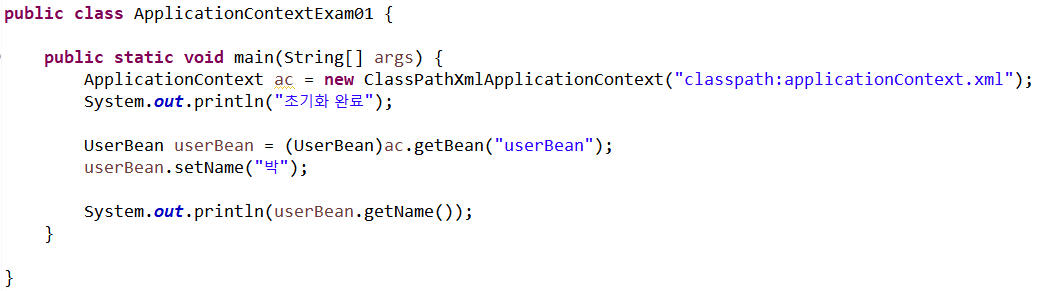
Src -> main -> 에 resources라는 폴더를 만들고 applicationContext.xml파일을 만들어서 Spring 컨테이너에게 줄 정보를 적어준다. Main폴더에 만든 resources폴더에서 생성한 xml파일은 자동으로 classpath로 지정이 된다.

<bean> </bean>이라는 엘리먼트에 몇 가지 속성을 부여한다.



Bean에는 class 경로와 그 클래스의 id를 설정해준다.

이제 kr.or.connect.diexam01패키지에 새로운 클래스를 만들어서 코드를 작성

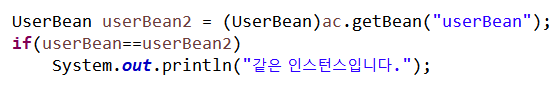


결과-> 초기화 완료

박

그런데 여기서 userBean2로 userBean을 불러와도 기존 userBean과 같다.

-> 하나의 객체만 메모리에 올라가는 싱글턴 패턴이다.



ApplicationConext ac = new ClassPathXmlApplicationContext(“~~~”);

실행 결과 xml에 정의된 모든 bean들의 객체가 메모리가 올라간다.

->이런 것들을 IOC제어의 역전이라고 한다.

새로운 객체를 만들고 싶다면

ApplicationConext ac = new ClassPathXmlApplicationContext(“~~~”);

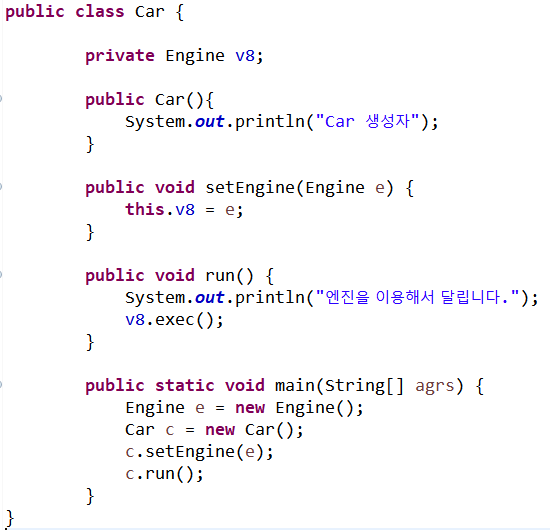
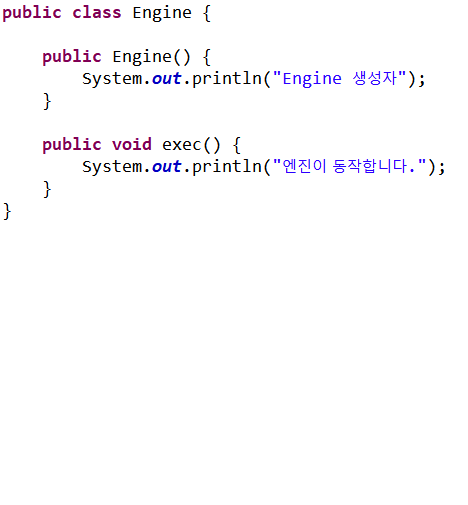
ApplicationConext ac2 = new ClassPathXmlApplicationContext(“~~~”);

와 같이 ApplicationContext를 다른 변수명으로 새로 할당한다.

\*의존성 주입

i)적용 안 했을 경우

Engine.java Car.java



출력->

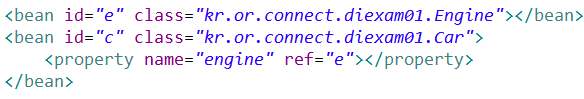
Engine 생성자

Car 생성자

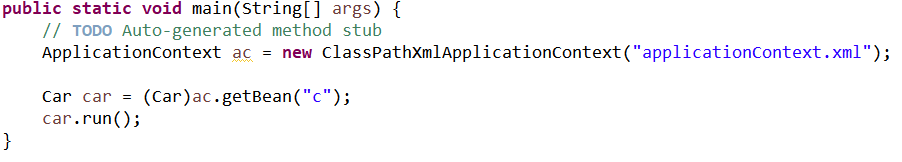
엔진을 이용해서 달립니다.

엔진이 동작합니다.

ii) 적용했을 경우



ApplicationContext.xml에 다음과 같이 추가. Ref필수다. 왜냐면 car객체안에 engine객체가 들어가기 때문이다.



출력->

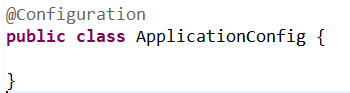
Engine 생성자

Car 생성자

엔진을 이용해서 달립니다.

엔진이 동작합니다.

출력결과 동일하다.



이 파일은 Config파일임을 알려주는 Annotation

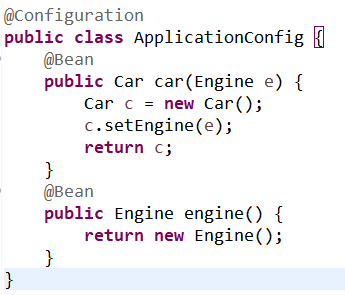
@Configuration-> Spring 설정 클래스이다.

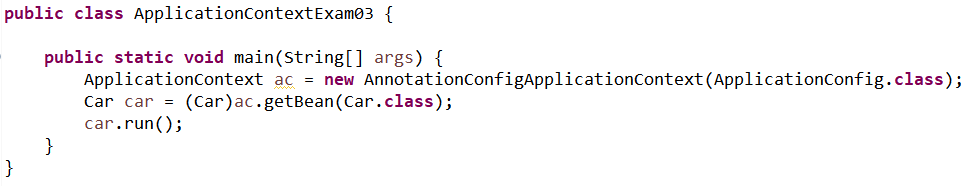
AnnotationConfigApplicationContext는 나중에 이런 자바 config클래스들을 읽어들여서 ioc와 di를 적용한다.

이때 설정 파일 중에 bean이라는 Annotation이 붙어 있는 메서드들을

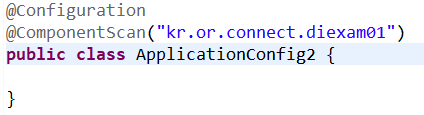
AnnotationConfigApplicationContext는 자동으로 실행해서 그 결과로 리턴해주는 객체들을 자동으로 싱글턴으로 관리해준다.

\* Config Annotation을 사용했을 경우

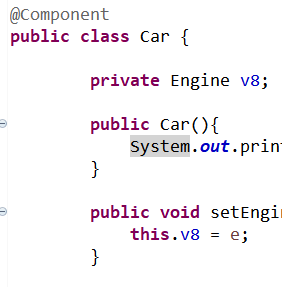




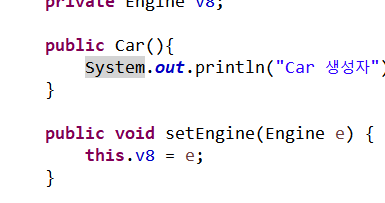
@ComponentScan(“패키지명”)=>패키지안에 있는 컨트롤러, 서비스, 리포지토리, 컴포넌트 이런 Annotation이 붙어있는 것들을 다 찾아서 bean으로 등록시켜줌



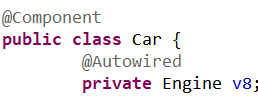
이제 클래스들을 찾아가서 @Component를 붙여준다. Engine클래스도 동일



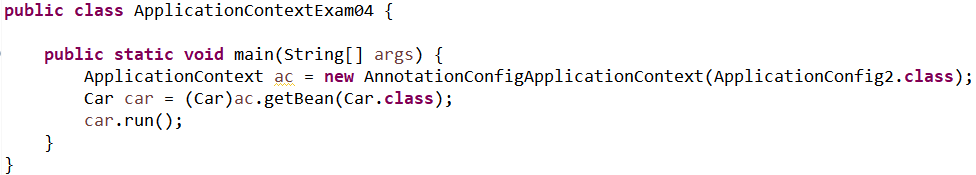
Car클래스안에 있는 setEngine을 지워준다.



그리고 나서 멤버변수 Engine v8; 위에다가 @AutoWired라는 Annotation은 자동으로 객체를 주입해달라는 의미이다.



ApplicationConfig2 클래스를 기준으로 Configuration한다.



**\*Spring JDBC**

-JDBC 프로그래밍을 보면 반복되는 개발 요소가 있다.

-이러한 반복적인 요소는 개발자를 지루하게 만든다.

-개발하기 지루한 JDBC의 모든 저수준 세부사항을 스프링 프레임워크가 처리해준다.

\*JdbcTemplate

-org.springframework.jdbc.core에서 가장 중요한 클래스

-리소스 생성, 해지를 처리해서 연결을 닫는 것을 잊어 발생하는 문제 등을 피할 수 있도록 한다.

-스테이트먼트의 생성과 실행을 처리한다.

-SQL조회, 업데이트, 저장 프로시저 호출, ResultSet반복호출 등을 실행한다.

\*DTO

-> DTO란 일반적으로 로직을 가지고 있지 않고 순수한 데이터 객체이다.

\*DAO

-> 데이터를 조회하거나 조작하는 기능을 전담하도록 만든 객체이다.

\*ConnectionPool이란?

-DB연결은 비용이 많이 든다.

-커넥션 풀은 미리 DB와 커넥션을 여러 개 맺어 둔다.

-커넥션이 필요하면 커넥션 풀에게 빌려서 사용한 후 반납한다.

\*DataSource

-커넥션을 얻고 반납하는 등의 작업을 수행

Maven 프로젝트 생성

Pom.xml에 추가하는 것.

1. <build></build> 부분에 jdk 1.8version을 설정

2. 의존성부분에 spring context,

3. spring jbdc

4. spring tx

5. mysql-connector

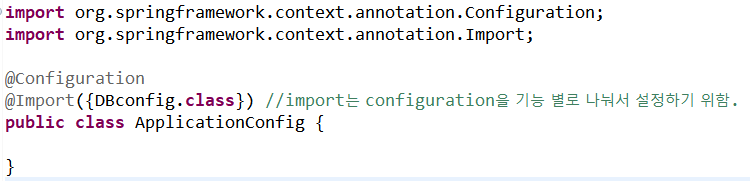
6. commons-dbcp2//DataSource의 종류 중에 하나이다.

Config 설정을 담을 새로운 패키지 하나 생성한다.

패키지 명: kr.or.connect.daoexam.config

Class는 ApplicationConfig.java, DBconfig.java 추가한다.

ApplicationConfig.java



DBconfig.java



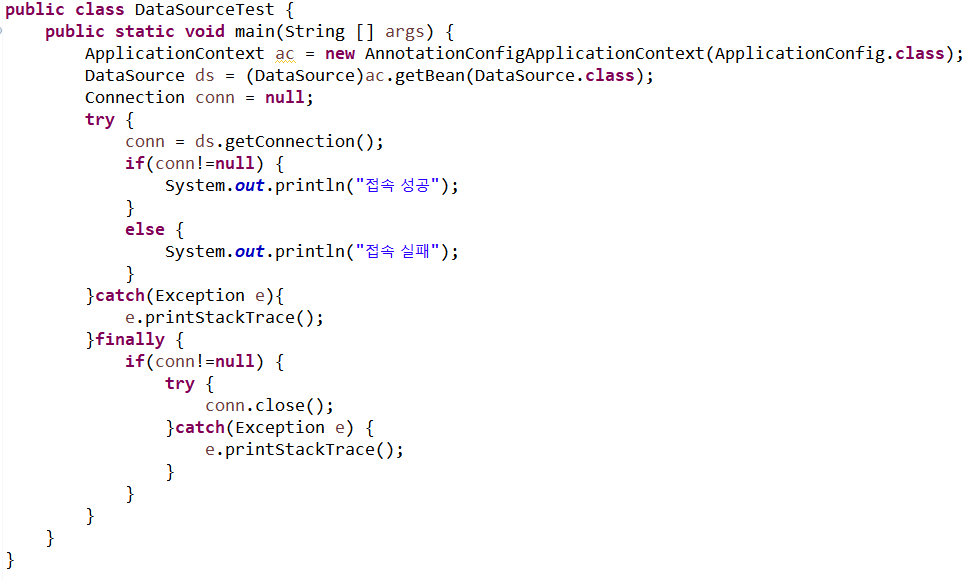
여기 까지 만들면 기본적인 구조는 완성!!!

이제 연결이 되는지 안되는지 테스트 해보겠다.

새로운 패키지에 클래스를 만들어준다.

패키지 명: kr.or.connect.daoexam.main

클래스 명: DataSourceTest.java



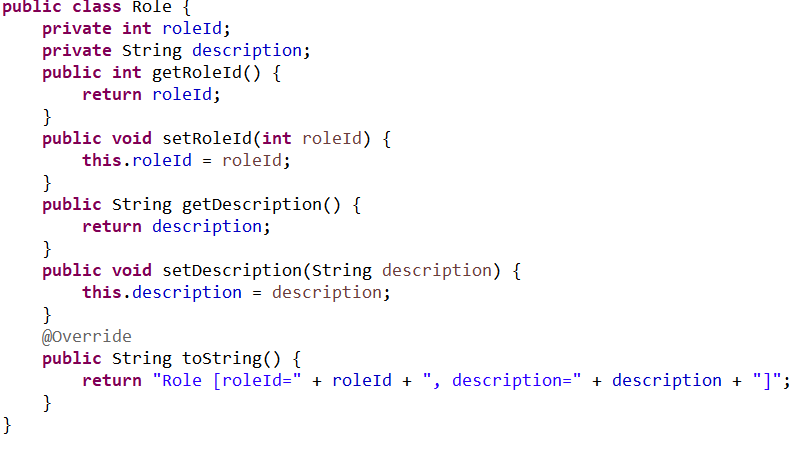
접속완료 성공!

Spring JDBC를 이용한 DAO작성 실습 시작

그러기 위해 DTO를 먼저 정의해줘야 한다.

DTO를 위한 새로운 패키지 생성시켜준다.

패지키 명: kr.or.connect.daoexam.dto



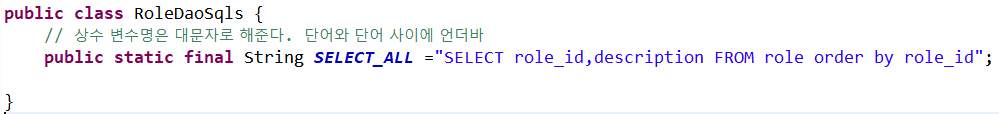
중요) 변수명 roleId (카멜 표기법)<->데이터베이스 칼럼명 role\_id

DAO를 위한 패키지도 생성해준다.

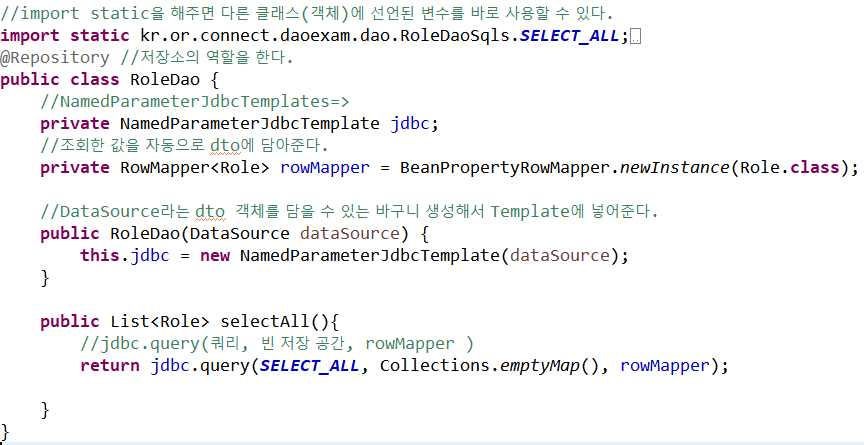
패키지 명: kr.or.connect.daoexam.dao

클래스는 dao의 sql문을 정의하기 위한 RoleDaoSqls.java, 로직들을 정의하기 위한 RoleDao.java를 만들어준다.

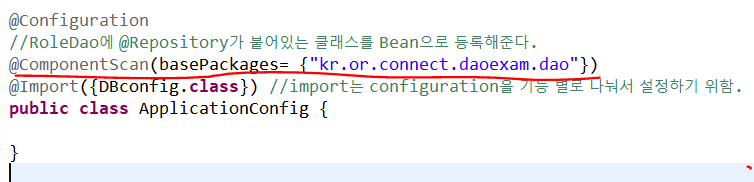
RoleDaoSqls.java에 모든 튜플을 조회하는 sql문을 상수형으로 정의해준다.



RoleDao.java에는

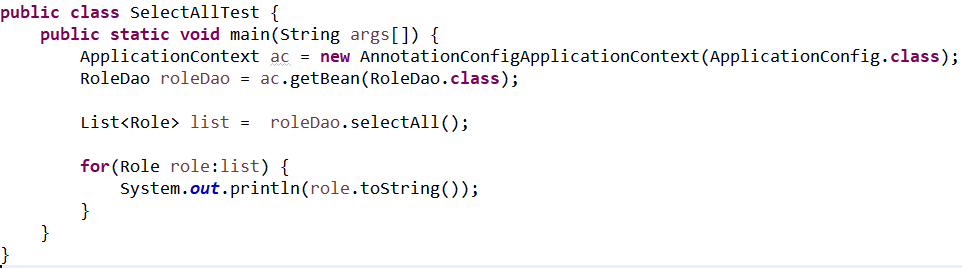


를 만들어주고 자동으로 Bean을 설정 하도록 ApplicationConfig.java에 ComponenetScan 줄 추가한다.



이제 kr.or.connet.daoexam.main 패키지에 SelectAllTest.java 파일을 추가한다.

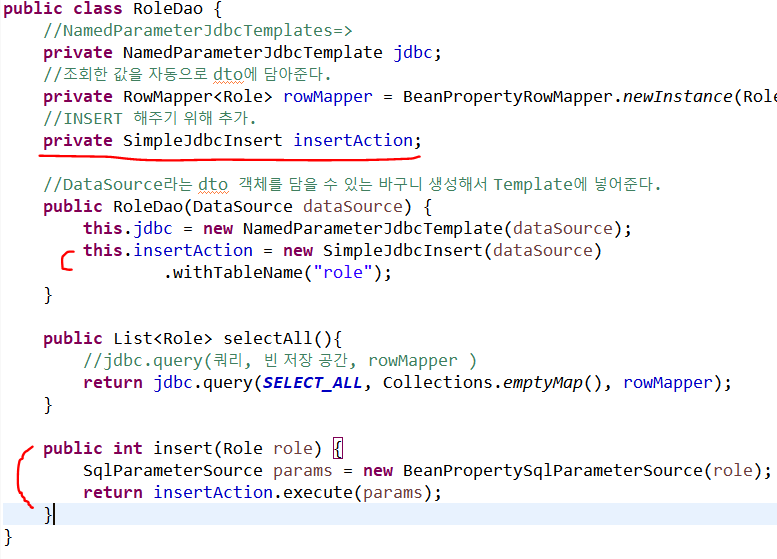
이 자바 파일에서 실제 모든 튜플을 조회를 할 것이다.



이제 다음 단계는 Insert와 Update문을 추가한다.

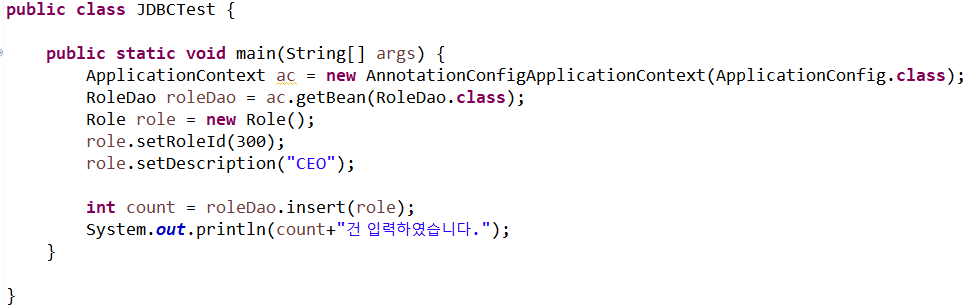
Insert를 하기 위해서는 SimpleJdbcInsert를 추가해줘야한다.

Insert는 따로 쿼리문이 사용되지 않으므로 RoleDao.java에서 바로 아래처럼 해줌.

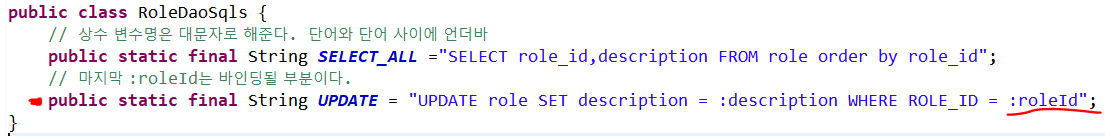


이를 테스트하기 위해 kr.or.connect.daoexam.main패키지 안에 JDBCTest.java

파일을 만들어준다.

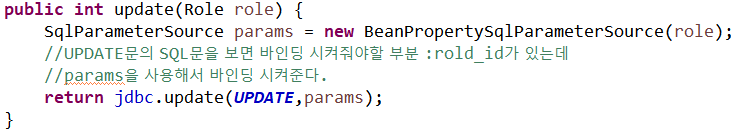


이제 Update를 만들어보자면 이것은 쿼리가 필요하므로 RoleDaoSqls.java파일에

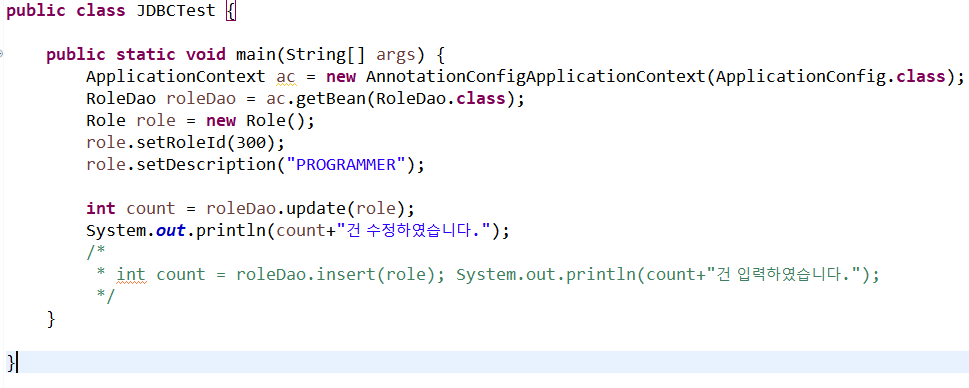


UPDATE변수를 추가해준다.

이제 RoleDao.java파일에서 update문을 실행하는 함수를 만들어준다.



JDBCTest.java 파일로 가서 role객체에 roleId= 300, description은 “PROGRAMMER”로 객체를 생성해서 update함수를 실행하면 roleId가 300인 description을 “PROGRAMMER”로 변경시켜준다.

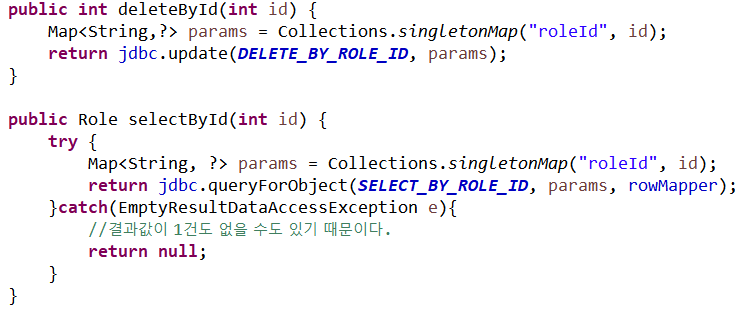


이제 1건 select해 오는 예제와 삭제하는 에제를 해보겠다.

이것은 쿼리가 필요하므로 RoleDaoSqls.java파일에 다음을 추가해준다.



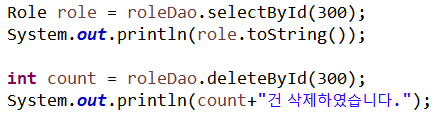
이제 쿼리문을 만들었으니 RoleDao.java파일에 가서 메서드를 작성해준다.



굳이 SqlParameterSource params = **new** BeanPropertySqlParameterSource(role);

를 안 쓴 이유는 굳이 객체를 생성할 필요 없이 id값만 매핑시켜주면 되기 때문.

JDBCTest.java 파일에 가서 테스트 해본다.



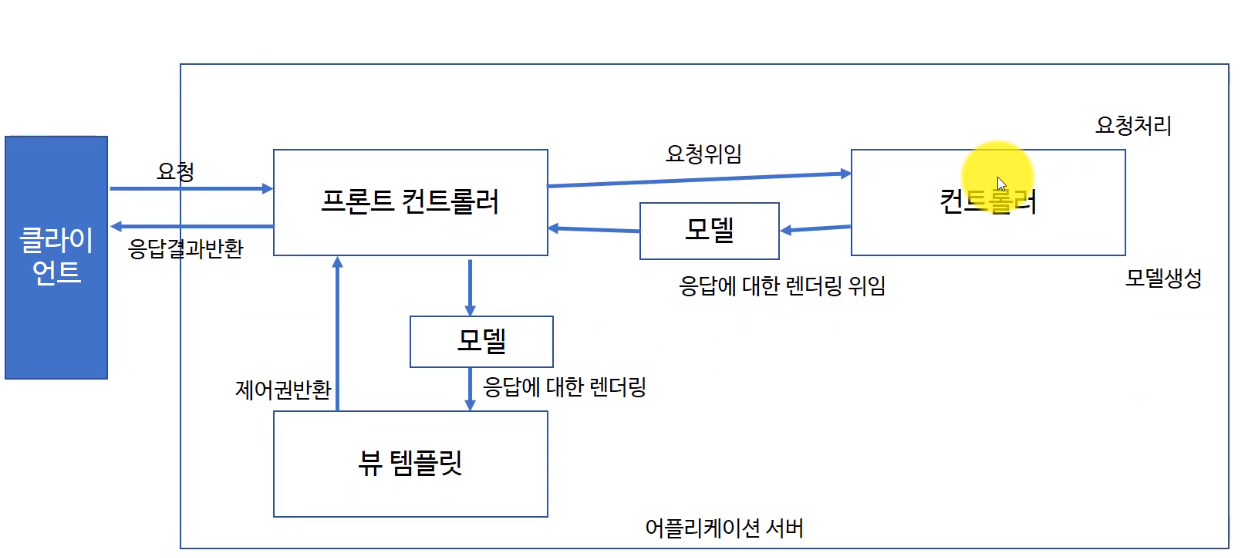
성공.

**\*Spring MVC**

Model: View가 렌더링하는데 필요한 데이터이다.

View: 웹 어플리케이션에서 뷰는 실제로 보이는 부분이며, 모델을 사용해 렌더링을 한다.

Controller: 컨트롤러는 사용자의 액션에 응답하는 컴포넌트이다. 컨트롤러는 모델을 업데이트하고, 다른 액션을 수행한다.

프론트 컨트롤러(=Servlet파일), 컨트롤러(컨트롤러 클래스, 핸들러 클래스)

컨트롤러는 Java bean 등을 이용해서 결과를 만들어내고 만들어진 결과를 모델에다 담아 보낸다.

Spring Framework의 모듈 중 Web Module 안에 Model2 MVC패턴이 구현되어져있다.

-생각해보기: 프론트 컨트롤러는 왜 사용하는 것일까?

장점이 어떤 것이 있을지 생각해보기 위해 프론트 컨트롤러가 없을 경우에 불편함을 생각해보았습니다.

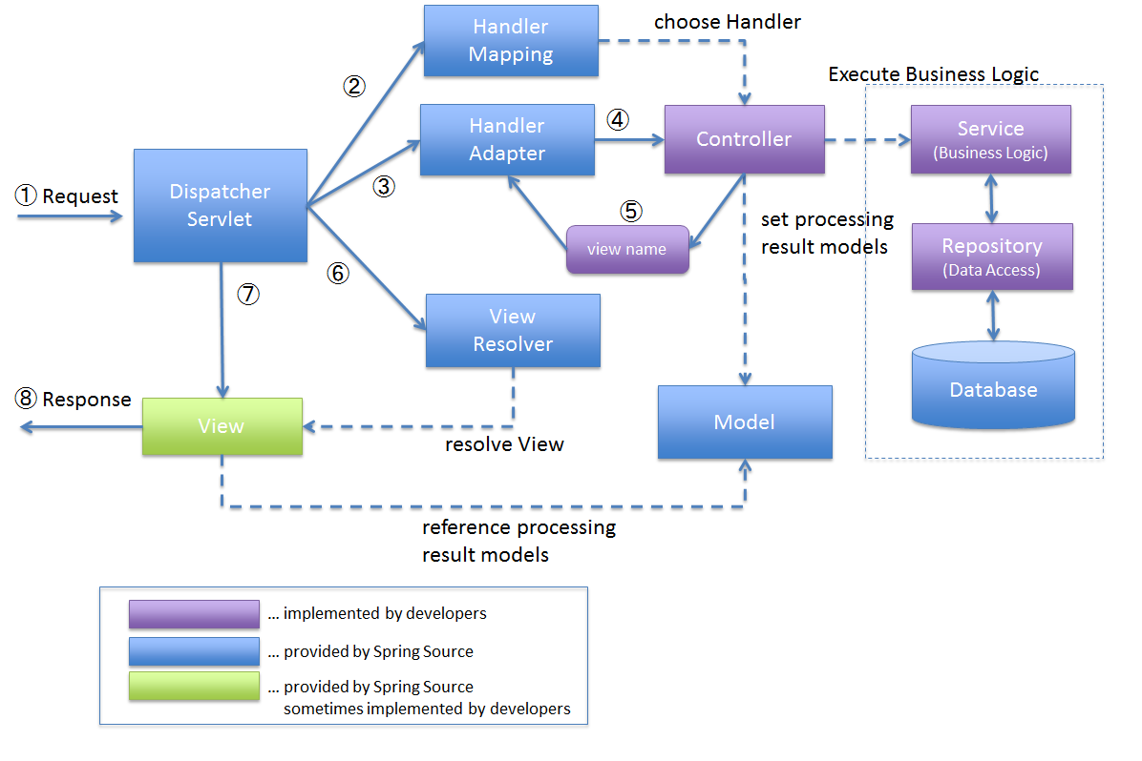
view에서 요청이 들어왔을 때 컨트롤러로 이동하게 되고 그 컨트롤에서 바로 로직을 처리하게 됐을 때, 1개의 요청 당 1개의 url을 가진 서블릿을 만들어야합니다. 이렇게 되면 유지 보수적인 측면과 코드 가독성, 개발의 효율이 떨어 질 수 밖에 없습니다.

따라서 프론트 컨트롤러라는 서블릿을 만들어 클라이언트의 요청을 한 곳으로 집중 시켜, 개발 및 유지보수에 효율을 극대화 시키는 방법을 사용하는 것입니다.

**\*Spring MVC 구성 요소와 기본 동작 흐름**

Database를 제외한 파란색 부분들은 spring mvc가 제공해준다.

개발자가 만들어야 하는 부분은 보라색, 녹색으로 되어있는 view는 개발자+스프링

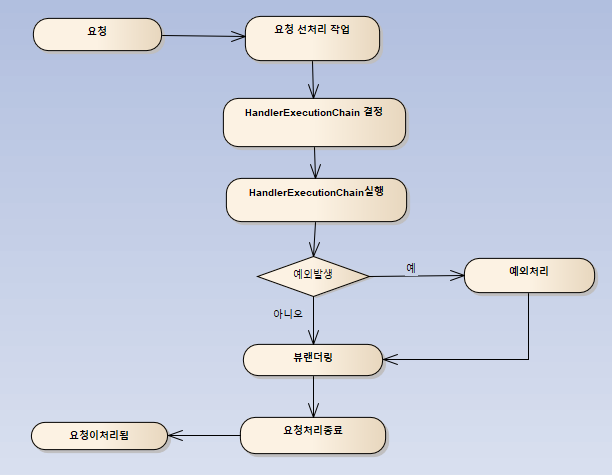


**\*Dispatcher Servlet**

-프론트 컨트롤러 역할을 한다.

-일을 처리 하지 않고 요청을 받고 넘겨주는 것 까지만 일을 한다.

-보통은 1개만 선언해서 사용한다.



**\* DispatcherServlet 내부 동작 흐름 상세 - 요청 선처리 작업시 사용된 컴포넌트**

1. org.springframework.web.servlet.LocaleResolver

지역 정보를 결정해주는 전략 오브젝트이다.

디폴트인 AcceptHeaderLocalResolver는 HTTP 헤더의 정보를 보고 지역정보를 설정해준다.

2. org.springframework.web.servlet.FlashMapManager

FlashMap객체를 조회(retrieve) & 저장을 위한 인터페이스

RedirectAttributes의 addFlashAttribute메소드를 이용해서 저장한다.

리다이렉트 후 조회를 하면 바로 정보는 삭제된다.

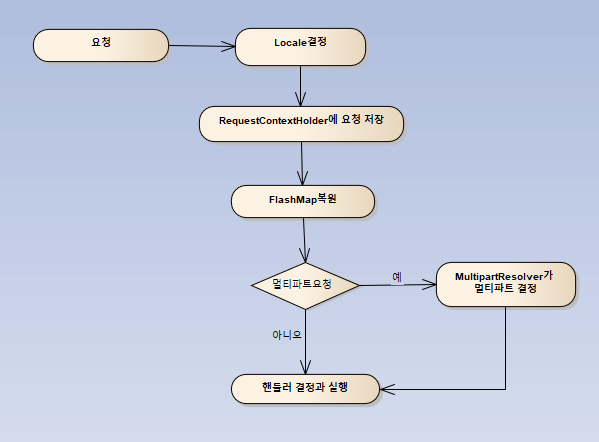
3. org.springframework.web.context.request.RequestContextHolder

일반 빈에서 HttpServletRequest, HttpServletResponse, HttpSession 등을 사용할 수 있도록 한다.

해당 객체를 일반 빈에서 사용하게 되면, Web에 종속적이 될 수 있다.

4. org.springframework.web.multipart.MultipartResolver

멀티파트 파일 업로드를 처리하는 전략



**\*DispatcherServlet 내부 동작 흐름 상세 - 요청 전달**

1. org.springframework.web.servlet.HandlerMapping

HandlerMapping구현체는 어떤 핸들러가 요청을 처리할지에 대한 정보를 알고 있다. 디폴트로 설정되는 있는 핸들러매핑은 BeanNameHandlerMapping과 DefaultAnnotationHandlerMapping 2가지가 설정되어 있다.

2. org.springframework.web.servlet.HandlerExecutionChain

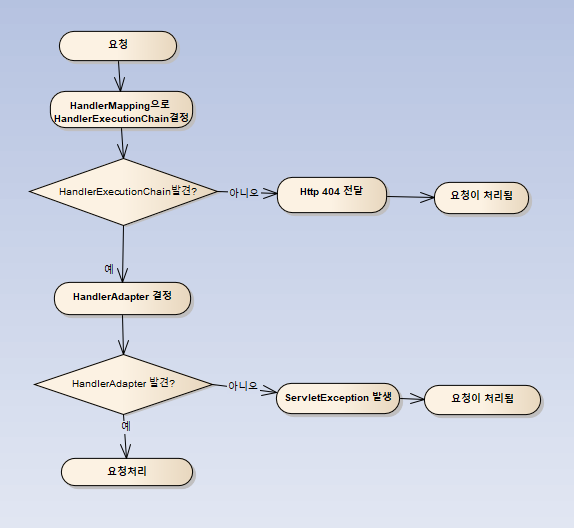
HandlerExecutionChain구현체는 실제로 호출된 핸들러에 대한 참조를 가지고 있다. 즉, 무엇이 실행되어야 될지 알고 있는 객체라고 말할 수 있으며, 핸들러 실행전과 실행후에 수행될 HandlerInterceptor도 참조하고 있다.

3. org.springframework.web.servlet.HandlerAdapter

실제 핸들러를 실행하는 역할을 담당한다. 핸들러 어댑터는 선택된 핸들러를 실행하는 방법과 응답을 ModelAndView로 변화하는 방법에 대해 알고 있다.

디폴트로 설정되어 있는 핸들러어댑터는 HttpRequestHandlerAdapter, SimpleControllerHandlerAdapter, AnnotationMethodHanlderAdapter 3가지이다.

@RequestMapping과 @Controller 애노테이션을 통해 정의되는 컨트롤러의 경우 DefaultAnnotationHandlerMapping에 의해 핸들러가 결정되고, 그에 대응되는 AnnotationMethodHandlerAdapter에 의해 호출이 일어난다.



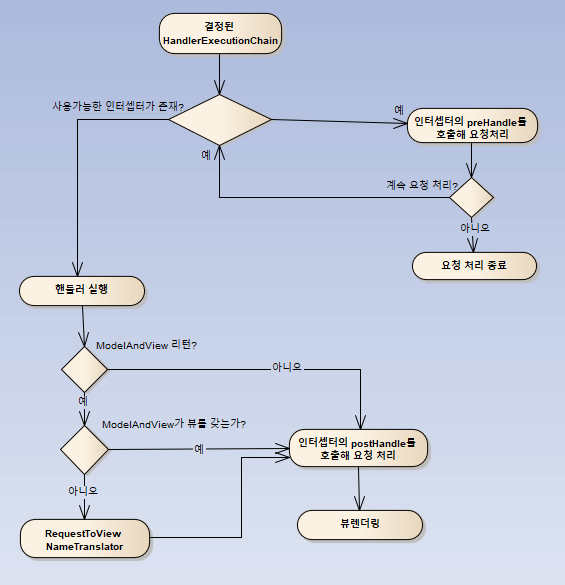
**\*DispatcherServlet 내부 동작 흐름 상세 – 요청 처리시 사용된 컴포넌트**

1. org.springframework.web.servlet.ModelAndView

ModelAndView는 Controller의 처리 결과를 보여줄 view와 view에서 사용할 값을 전달하는 클래스이다.

2. org.springframework.web.servlet.RequestToViewNameTranslator

컨트롤러에서 뷰 이름이나 뷰 오브젝트를 제공해주지 않았을 경우 URL과 같은 요청정보를 참고해서 자동으로 뷰 이름을 생성해주는 전략 오브젝트이다. 디폴트는 DefaultRequestToViewNameTranslator이다.

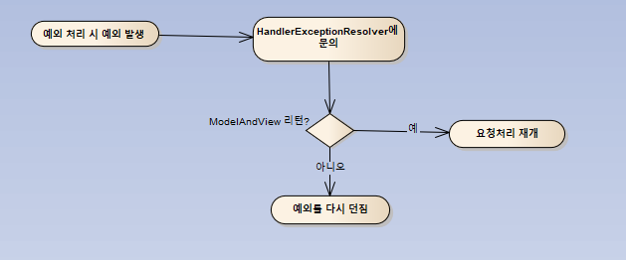


**\*DispatcherServlet 내부 동작 흐름 상세 – 예외 처리시 사용된 컴포넌트**

1. org.springframework.web.servlet.handlerexceptionresolver

기본적으로 DispatcherServlet이 DefaultHandlerExceptionResolver를 등록한다.

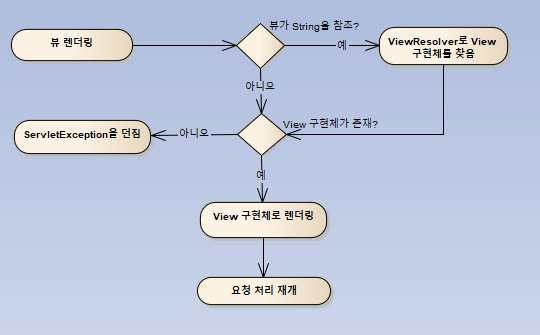
HandlerExceptionResolver는 예외가 던져졌을 때 어떤 핸들러를 실행할 것인지에 대한 정보를 제공한다.



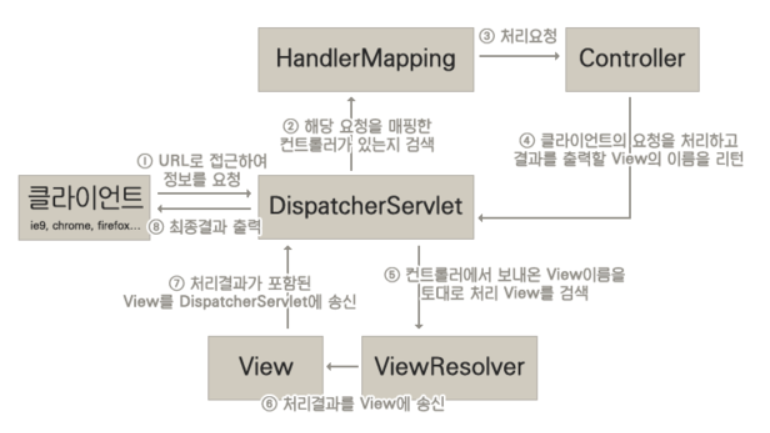
**\*DispatcherServlet 내부 동작 흐름 상세 – 뷰 렌더링 과정시 사용된 컴포넌트**

1. org.springframework.web.servlet.ViewResolver

컨트롤러가 리턴한 뷰 이름을 참고해서 적절한 뷰 오브젝트를 찾아주는 로직을 가진 전략 오프젝트이다. 뷰의 종류에 따라 적절한 뷰 리졸버를 추가로 설정해줄 수 있다.



이런 상세한 과정에 대해 조금 간략하게 정리해 보자면 다음과 같다.



이제 Spring MVC 웹 어플리케이션 실습 시작!!

Maven project 생성 -> 웹 어플리케이션을 만들 것이지 아키텍쳐 타입은 webapp을 선택해준다.

Src->main 폴더 안에 java라는 이름의 폴더를 하나 생성시켜준다.

그렇게되면 JavaResources에 src/java/main이 생성된다.

이제 jdk1.8 version을 사용하기 위해 pom.xml에 플러그인 추가한다.

그리고 jstl, jsp, servlet을 사용하기 위해서 라이브러리 추가.

Spring 사용하기 위해 spring-context 추가, Spring mvc 사용하기 위해서 webmvc 추가. Maven project update 해준다.

Webapp 같은 경우에는 Navigator에서 .settings에서 facet.core.xml부분 지정해줘야한다.

서블릿 버전이 2.3버전으로 되어잇는 것은 3.1로 수정해준다.

Eclipse 재시작 하고 project Facets에서 dynamic web module이 제대로 잘 바뀌어있는지 까지 확인한다.

기본 세팅 완료!

Spring mvc를 위한 몇 가지 기본적인 설정.

1. DispatcherServlet을 FrontController로 설정. (2가지 방법이 존재)

a.Web.xml 파일에 설정

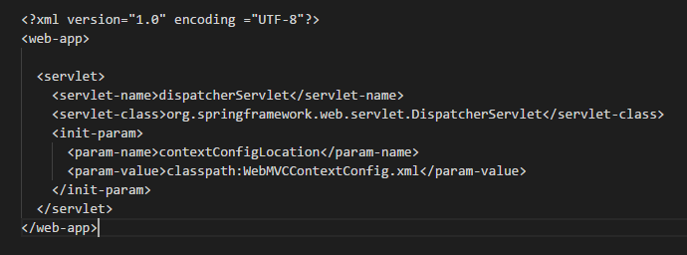
b.org.springframework.web.WebApplicationInitializer인터페이스를 구현해서 사용

(수업에선 a방법을 사용)

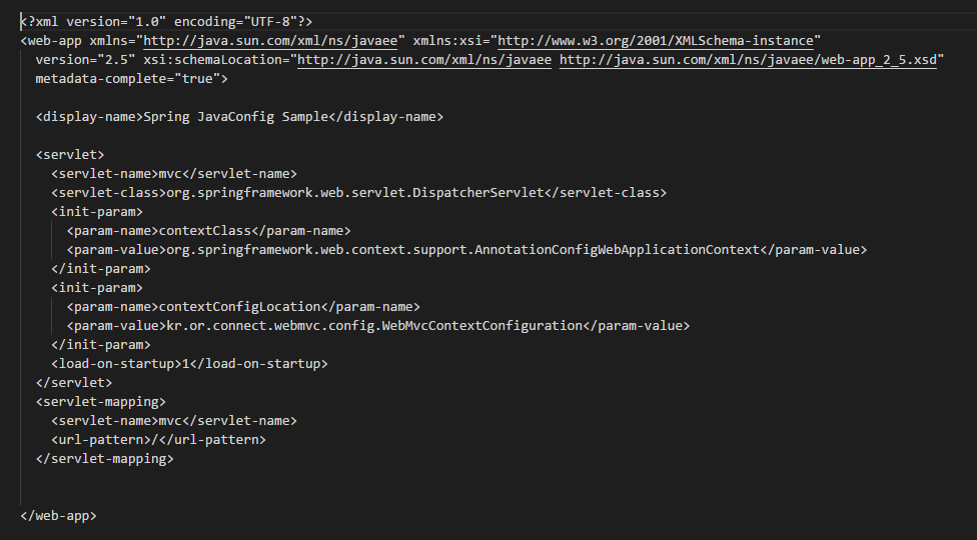
-> 자세히

Web.xml 파일에 설정

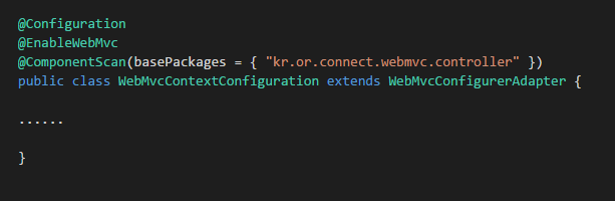
-xml spring설정 읽어들이도록 DispatcherServlet설정.



-Java config spring 설정 읽어들이도록 DispathcerServlet설정



2.Spring MVC 설정



1. @Configuration

-자바 config파일이라는 것을 알려주는 역할

2. @EnableWebMvc

-웹에 필요한 빈들을 대부분 자동으로 설정

3. @ComponentScan

-Spring MVC에서는 핸들러 즉, 컨트롤러를 찾기 위해서 사용.

-Controller, service, Repository, component애노테이션이 붙은 클래스를 찾아 스프링 컨테이너가 관리하게 된다.

4. 상속받은 WebMvcConfigurerAdapter

-@EnableWebMvc를 이용하면 기본적인 설정이 모두 자동으로 되지만, 기본 설정 이외의 설정이 필요할 경우 해당 클래스를 상속 받은 후, 메소들 오버라이딩하여 구현한다.

설정과 관련된 부분 끝!

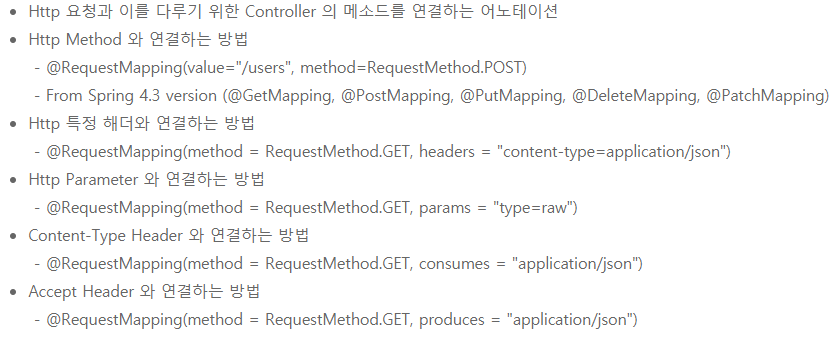
**이제 Controller(Handler)클래스 작성하기!**

-@Controller 애노테이션을 클래스 위에 붙인다.

-맵핑을 위해 @RequestMapping 애노테이션을 클래스나 메소드에서 사용한다.

@RequestMapping

-Http요청과 이를 다루기 위한 Controller의 메소드를 연결하는 어노테이션.



이제 실제 프로젝트에 작업을 시작해보겠다!

kr.or.connect.mvcexam.config 이라는 패키지를 하나 만들어준다.

해당 패키지안에 Config와 관련된 클래스 파일을 넣어주겠다.

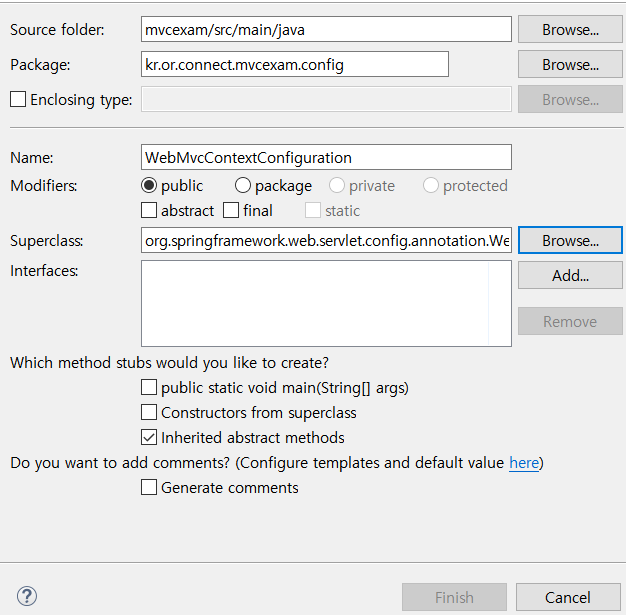
클래스명은 WebMvcContextConfiguration로 하고

WebMvcConfigurationAdapter를 상속받아준다.

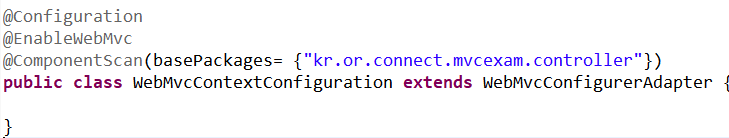
상속 받는 방법은 하단 이미지 참고

Browser를 누른 후 WebMvcConfigurerAdapter검색해서 확인.

상속 받는 이유-> @EnableWebMvc가 제공하는 기본적인 기능 이외에 더 기능을 직접 구현하여 추가하기 위해서이다.

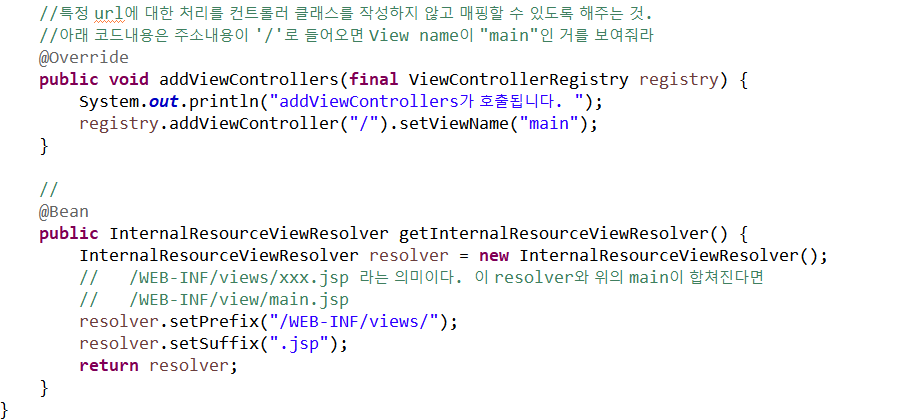


여기 까지 만들고 정리를 하자면 @어노테이션들을 붙여준다.



여기 까지 정리를 해보자면 지금 현재 이 클래스 자체가 DispatcherServlet이 실행될 때 읽어들이는 설정 파일이다.





이제 DispatcherServlet을 FrontController로 사용하기 위한 Web.xml 작성

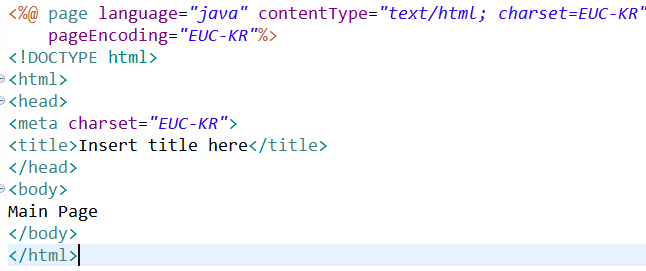
Web.xml은 src->mian->webapp->WEB-INF->web.xml에 있다.



이제 설정이 잘 되었는지 간단한 테스트를 시작!

WEB-INF에다가 views라는 폴더 만들고 main.jsp파일을 만든다.

이때, index.jsp를 삭제시켜준다. 왜냐하면 최상위에 index.jsp가 있다면 ‘/’라는 root경로가 들어왔을 때, 최상위 index.jsp를 자동으로 실행하기 때문이다.



프로젝트를 실해아면 Main.jsp 이 정상적으로 실행된다.

Controller 작성 실습 1/3

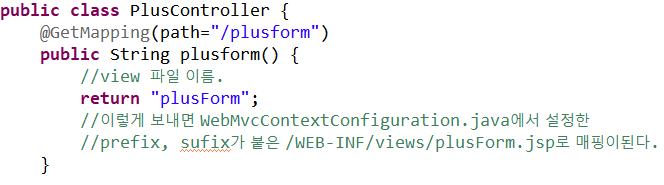
1. 웹 브라우저에서 http://localhost:8080/mvcexam/plusform 이라고 요청을 보 내면 서버는 웹 브라우저에게 2개의 값을 입력받을 수 있는 입력 창과 버튼이 있는 화면을 출력한다.

2. 웹 브라우저에 2개의 값을 입력하고 버튼을 클릭하면http://localhost:8080/mvcexam/plus URL로 2개의 입력값이 POST방식으로 서버에게 전달한다. 서버는 2개의 값을 더한 후, 그 결과 값을 JSP에게 request scope으로 전달하여 출력한다.

우선 /plusform이라는 url이 실행되면 띄울 jsp 파일을 WEB-INF/views.plusForm.jsp로 만들어준다.

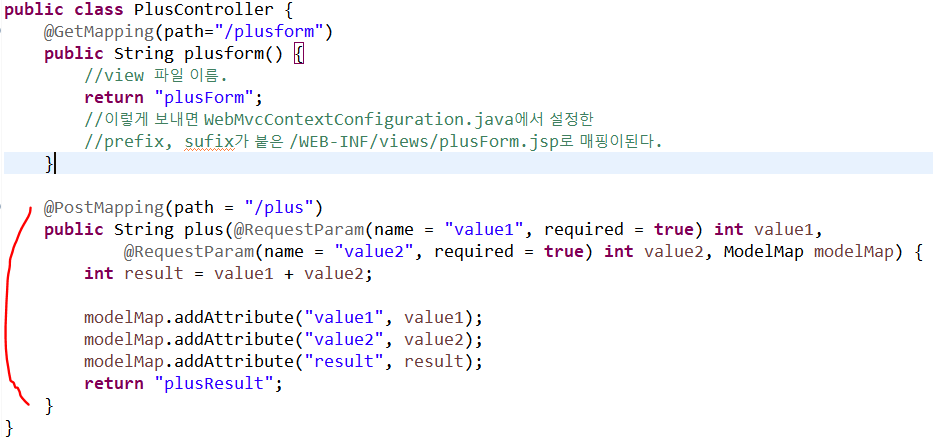


패키지 kr.or.connect.mvcexam.controller를 만들어주고 그 안에 PlusController.java 클래스를 만들어준다. 이 클래스에는 url 매핑에 따른 view파일 매핑 및 로직들이 들어갈 것이다.



url: <http://localhost:8080/mvcexam/plusform> 일때 정상으로 뜬다.

다음은 더하는 로직을 만들어줘야하므로 새로운 url매핑을 post 형식으로 만들어준다.

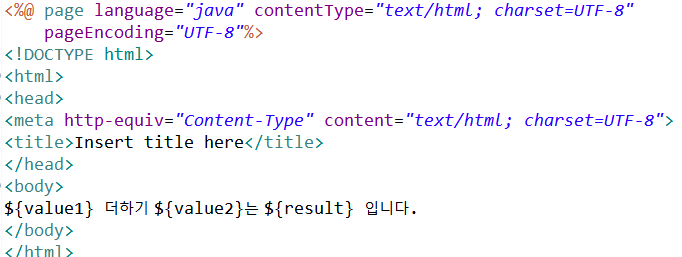


@RequestParam은 input타입의 name과 이름이 매핑된다.

ModelMap modelMap은 Spring이 제공하는 객체이다. 이 객체는 RequestScope에

자동으로 매핑된다.

이제 더하기 결과를 출력해줄 plusResult.jsp파일을 만들어주다.



plusform에 시작하여 테스트를해본다. 테스트 결과: 성공

Controller 작성 실습 2/3

1. http://localhost:8080/mvcexam/userform 으로 요청을 보내면 이름, email, 나이를 물어보는 폼이 보여진다.

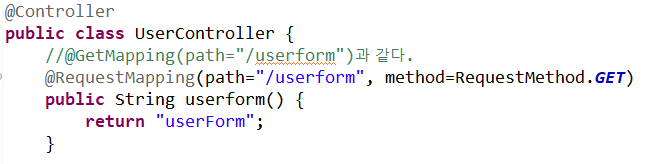
2. 폼에서 값을 입력하고 확인을 누르면 post방식으로 http://localhost:8080/mvcexam/regist 에 정보를 전달하게 된다.

3. regist에서는 입력받은 결과를 콘솔 화면에 출력한다.

userform url이 들어왔을 때, 보여지는 userForm.jsp페이지를 만든다.



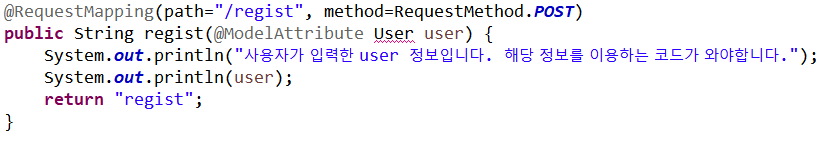
유저와 관련된 컨트롤러를 새로 만들어준다. controller패키지 안에 UserController.java 클래스를 생성한다.



userForm.jsp 매핑과 url을 매핑시켜준다. -> 성공

이제 사용자가 입력한 user정보를 받아와서 출력하는 로직부분을 만들어준다.

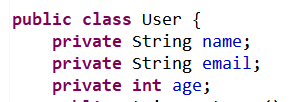
UserController.java 파일에 다음 코드를 추가한다.



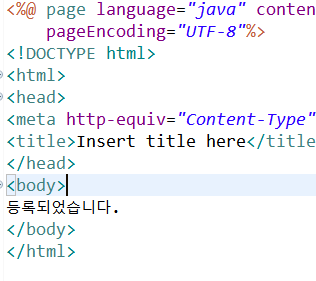
Input type으로 들어온 정보가 User 객체에 넣어져서 들어와지도록 코드를 작성했으니 User객체를 만들어주어야한다. => @ModelAttribute의 기능

(@RequestParam으로 하나하나 받을 수도 있긴하다.)

새로운 kr.or.connect.mvcexam.dto 패키지를 만들어주고 그 안에 User.java클래스를 만들어준다. 멤버변수는 다음과 같고 getter, setter메소드 자동 생성, toString() 메소드를 생성 해준다.



이제 regist.jsp라는 페이지 넘김용?을 하나 만들어준다. (어차피 중요한건 콘솔창에 출력되는 부분이다.)



Controller작성 실습 3/3

1. http://localhost:8080/mvcexam/goods/{id} 으로 요청을 보낸다.

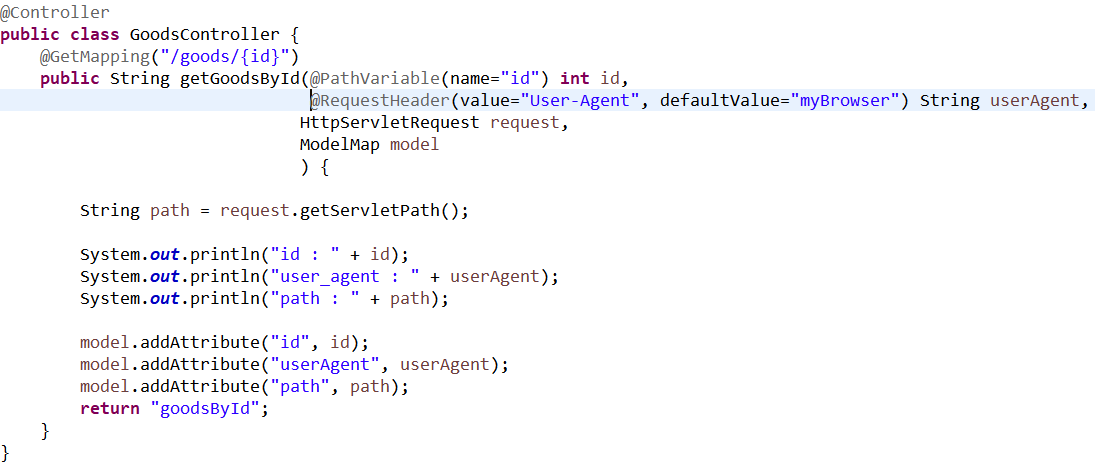
2. 서버는 id를 콘솔에 출력하고, 사용자의 브라우저 정보를 콘솔에 출력한다.

3. 서버는 HttpServletRequest를 이용해서 사용자가 요청한 PATH정보를 콘솔에 출력한다.

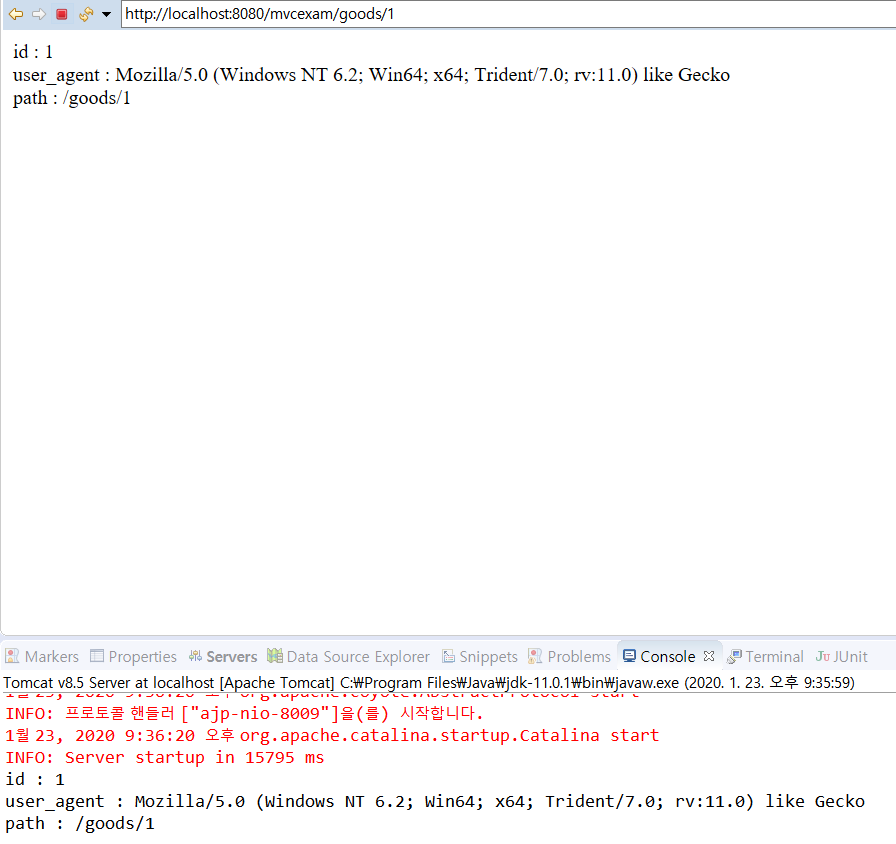
goods/{id}요청을 받을 때, 띄울 goodsById.jsp 페이지를 만들어준다.



goods 와 관련된 controller패지키에 GoodsController.java 클래스를 생성하고 관련 로직을 작성해준다.



이제 테스트로 /goods/1 이라는 url에 접속하면 아래와 같은 화면이 뜬다.



**\*레이어드 아키텍처 란?**

Controller에서 중복되는 부분을 처리하려면?

-별도의 객체로 분리한다.

-볃로의 메소드로 분리한다.

컨트롤러 들이 중복적으로 호출되는 부분들은 서비스 객체로 구현하고 서비스 객체는 보통 업무와 관련된 메서드를 가지고 있고 그 메서드들을 비즈니스 메서드라고한다.

**서비스(Service)객체란?**

비지니스 로직(Business logic)을 수행하는 메소드를 가지고 있는 객체를 서비스 객체라고 합니다.

보통 하나의 비지니스 로직은 하나의 트랜잭션으로 동작합니다.

**트랜잭션(Transaction)이란?**

트랜잭션의 특징은 크게 4가지로 구분됩니다.

1. 원자성 (Atomicity)
2. 일관성 (Consistency)
3. 독립성 (Isolation)
4. 지속성 (Durability)

**JDBC 프로그래밍에서 트랜잭션 처리 방법**

DB에 연결된 후 Connection객체의 setAutoCommit메소드에 false를 파라미터로 지정합니다.

입력, 수정, 삭제 SQL이 실행을 한 후 모두 성공했을 경우 Connection이 가지고 있는 commit()메소드를 호출합니다.

**@EnableTransactionManagement**

Spring Java Config파일에서 트랜잭션을 활성화 할 때 사용하는 애노테이션입니다.

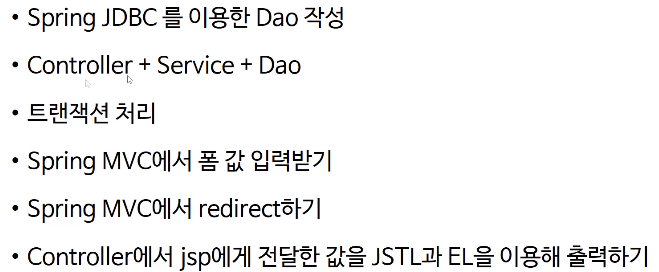
Java Config를 사용하게 되면 PlatformTransactionManager 구현체를 모두 찾아서 그 중에 하나를 매핑해 사용합니다.

특정 트랜잭션 메니저를 사용하고자 한다면 TransactionManagementConfigurer를 Java Config파일에서 구현하고 원하는 트랜잭션 메니저를 리턴하도록 합니다.

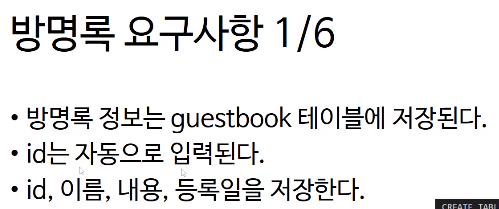
아니면, 특정 트랜잭션 메니저 객체를 생성시 @Primary 애노테이션을 지정합니다.

**\*레이어드 아키텍처 실습**

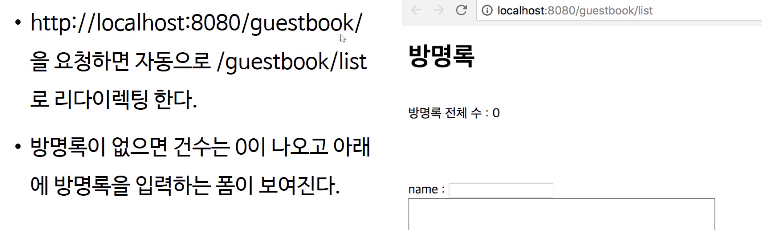
방명록 만들기 실습



첫번째 요구사항



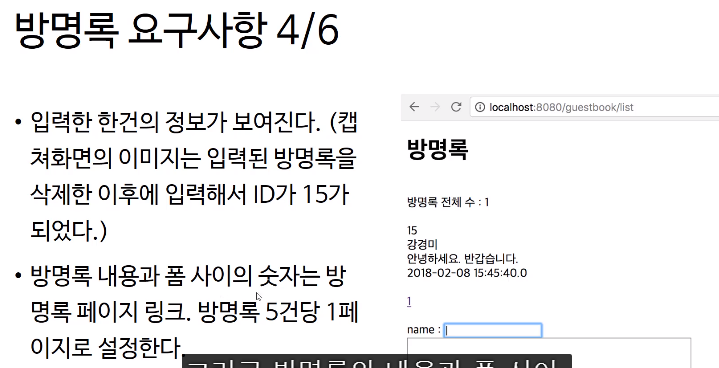
두번째 요구사항.



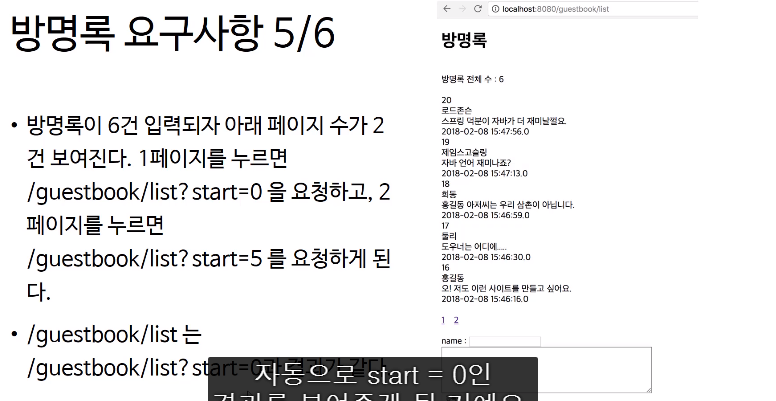
세번째 요구사항



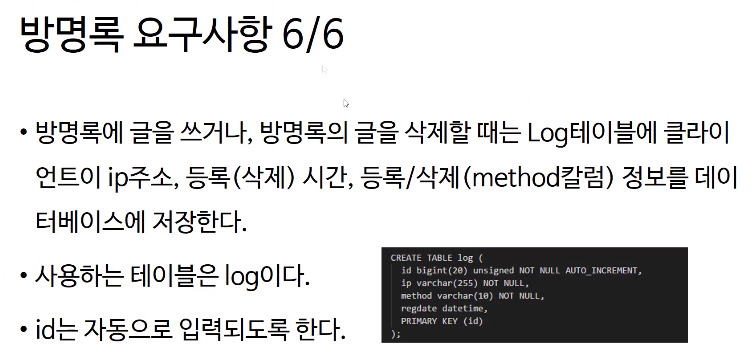
네번째 요구사항



다섯번째 요구사항



마지막 요구사항



총 5개의 설정을 해야한다.

1. 기본 설정: pom.xml

2. WebMvcContextConfiguration 설정

3. DBconfig 설정

4. ApplicationConfig 설정

5. web.xml 설정

패키지는 이렇게 구성된다.

1. dao : 여러 기능을 수행하는 함수들 개발(복합적인 기능 No, 개별 개별적 기능)

2. dto : 객체 정의.

3. config: Applicationconfig, DBconfig, WebMvcContext

4. service: dao에서 만든 함수들의 기능을 합쳐 실질적으로 이용하는 함수 선언

// 서비스 인터페이스

5.service.impl: service에서 선언한 함수들에 대한 직접적인 구현.

// 서비스 인터페이스에 대한 구현

6.service controller: url에 접속 요청에 따른 처리 및 응답, redirect (service메소드 사용)

참고)

1.main method에서 실행(테스트)하는 거면 run on server가 아니라 java application을 실행해야한다. 사실 main method가 쓰이는 것이 좋지 않음. junit을 사용해서 테스트 하는 방법을 익히는 것을 추천.

2. @Bean은 객체를 return시킬 경우 사용한다.

1. pom.xml 설정

의존성 부분

-spirng.version 상수선언.

-junit 테스트를 위함

-spring context

-spring web-mvc

-servlet, jstl, jsp

-spring jdbc

-spring tx

-mysql-connector

-commons-dbcp2 : data source를 위함

-jackson-core : json 사용을 위함.

-jackson-datatype

플러그인 부분

maven-compiler-plugin : jdk version

2. WebMvcContextConfiguration

WebMvcConfigurerAdapter을 상속받는다.

어노테이션은 Configuration, EnableWebMvc, ComponentScan을 사용.

컴포넌트 스캔으로 controller를 설정한다.

기본 /이 들어왔을 때, index로 가게한다.

img, js, css경로일 경우엔 따로 처리해준다.

/WEB-INF/views/ ~~~.jsp/ 경로 설정한다.

3. DBconfig

어노테이션은 Configuration, EnableTransactionManagement을 사용.

private String driverClassName, url, username, password 변수 선언 및 값 저장

dataSource()객체를 생성하는 것이 핵심..!

dao에서 dataSource 객체를 사용, @Repository선언과 javax.sq.DataSource때문이다.

**\*Rest Controller**

1. Rest Controller를 사용하려면 반드시 Jackson 라이브러리를 추가해야한다.