[\\192.168.0.58](file:///\\192.168.0.8),80

모델1

* Jsp만 이용
* Jsp+javabean 이용
* Model2 의 컨트롤러 개념모호함
* 개발속도는 빠르지만 ㅡㅡ직관적이다.
* 비즈니스 뷰 등 다갖추고있어 혼재되어있다. 모델엔드뷰 가구분이안되어있어.

모델2

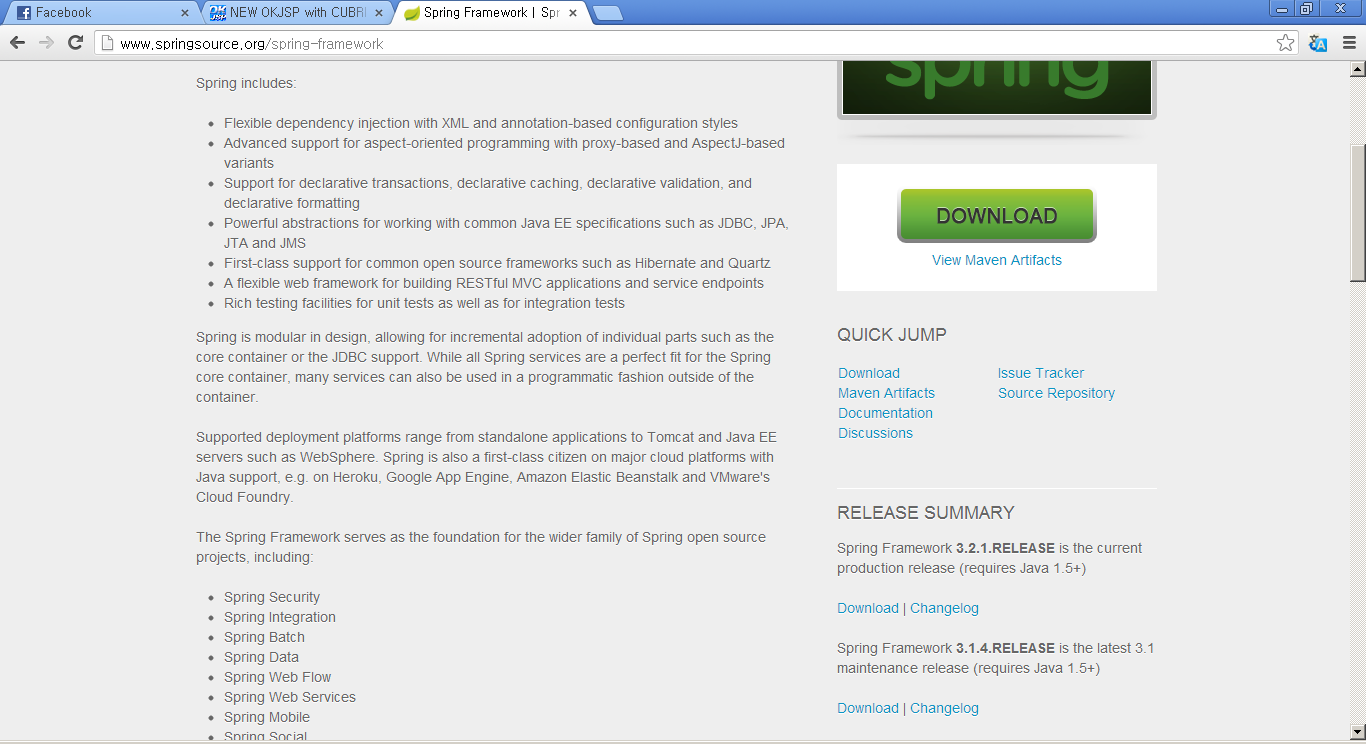
* 처음 요청받는 것은 컨트롤러가 받는다. MVC로.
* jsp는 뷰만신경써라
* 비즈니스 로직 프리젠테이션 분리로인해 어플리케이션 명료해진다
* 개발속도가 좀 늦다.

Spring 스프링

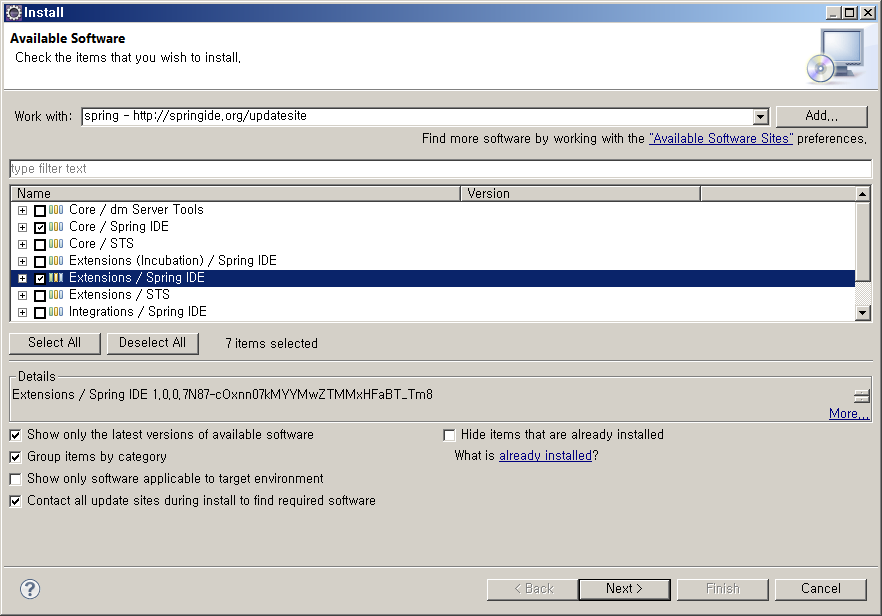
* spring은 봄을 뜻하는.
* [www.springfram]ework.org](http://www.springfram]ework.org)
* 경량 컨테이너(light weight container)

**Spring!** [**http://www.springsource.org/**](http://www.springsource.org/)

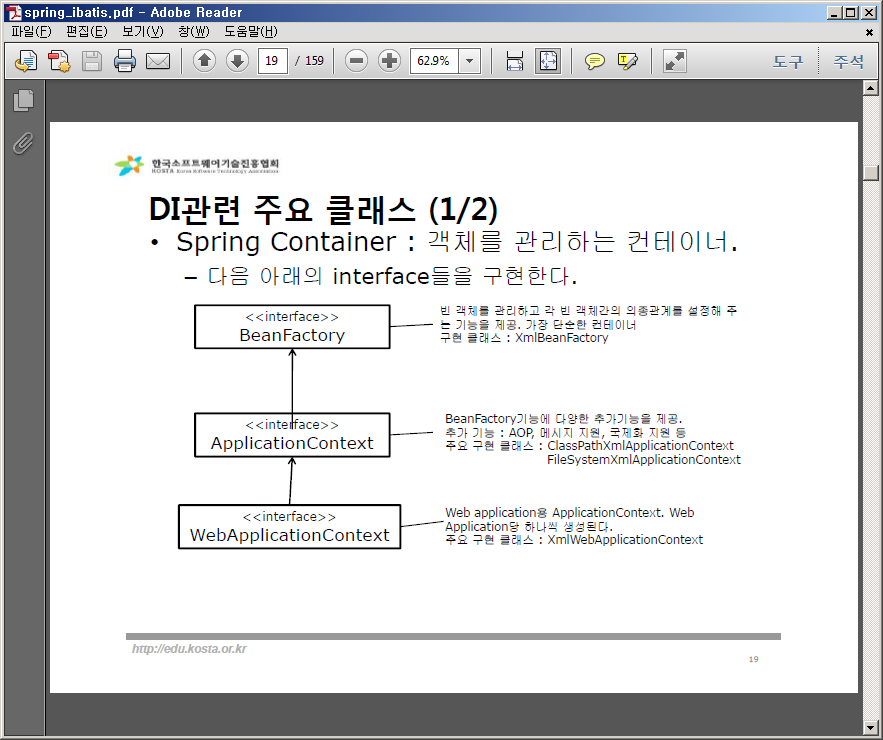
**http://www.springsource.org/download/community**

****

eclipse 셋팅http://springide.org/updatesite



DI(Dependency Injection,DI) 의존성 주입



DI를 통해 시스템에 있는 각 객체를 조정하는 외부 개체가 객체들에게 생성시에 의존관계를 주어짐.

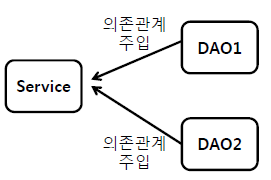
• 즉, 의존이 객체로 주입됨.

• 객체가 협업하는 객체의 참조를 어떻게 얻어낼 것인가라는 관점에서 책임성의 역행(inversion of responsibility)임.

느슨한결합

객체는 인터페이스에 의한 의존관계만을 알고 있으며, 이 의존관계는 구현 클래스에 대한 차이를 모르는채 서로 다른 구현으로 대체가

가능



XML에서 설정하면은 된다 즉 config 파일로 뭐든걸 제어한다

의존관계설정이라던지 xml에서 처리한다 Spring Container가 제공

Spring Container가 DI 조립기를 제공

– 스프링 설정파일을 통하여 객체간의 의존관계를 설정한

다.

– Spring Container가 제공하는 api를 이용해 객체를 사

용한다.

Spring 설정파일

• Application에서 사용할 Spring 자원들을 설정하는 파일

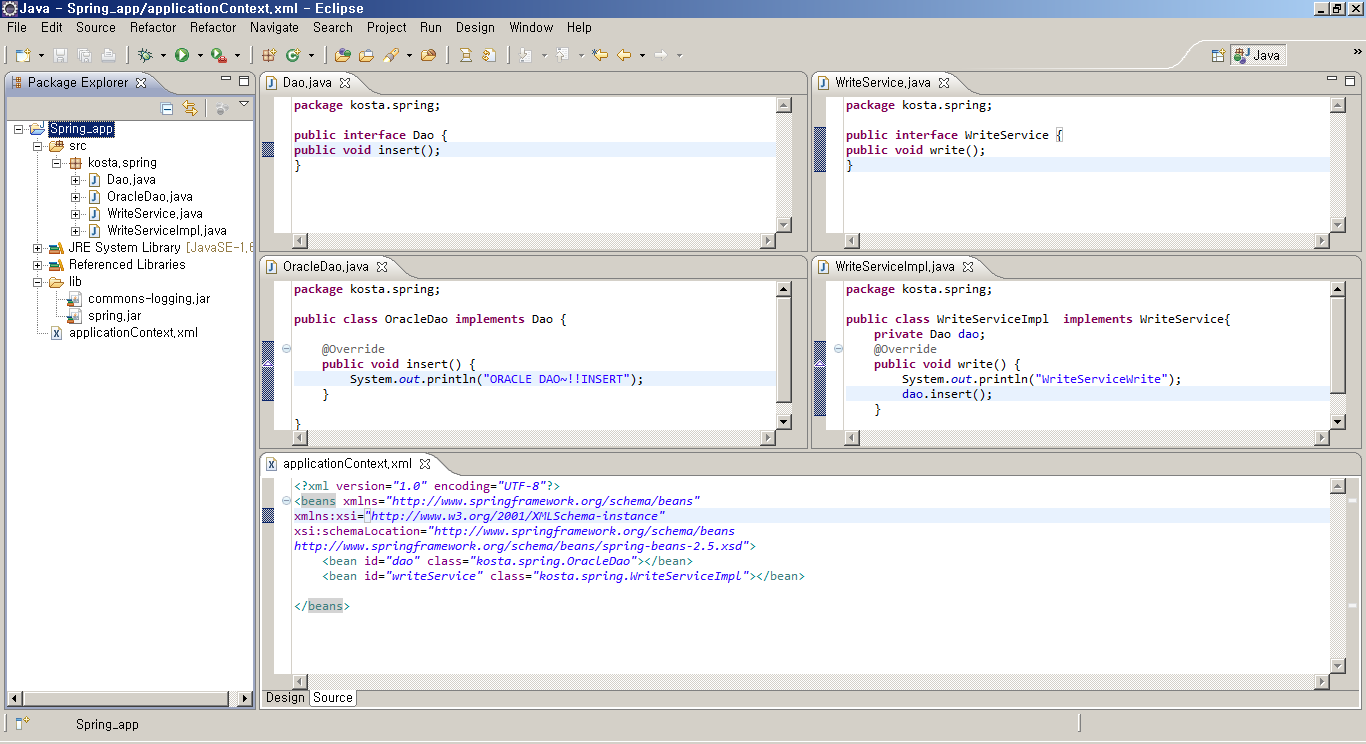
• Spring container는 설정파일에 설정된 내용을 읽어

Application에서 필요한 기능들을 제공한다.

• XML 기반으로 작성한다.

• Root tag는 <beans> 이다

• 파일명은 상관없다.



|  |
| --- |
| package kosta;  import kosta.spring.WriteService;  import org.springframework.beans.factory.BeanFactory;  import org.springframework.beans.factory.xml.XmlBeanFactory;  import org.springframework.core.io.ClassPathResource;  import org.springframework.core.io.Resource;  public class main {  public static void main(String[] args) {  //설정파일이 어디 있는지를 저장하는 객체  **Resource resource = new ClassPathResource("applicationContext.xml");**  //객체를 생성해주는 factory 객체  **BeanFactory factory = new XmlBeanFactory(resource);**  //설정파일에 설정한 <bean> 태그의 id/name을 통해 객체를 받아온다.  **WriteService service = (WriteService)factory.getBean("writeService");**  service.write();  }  } |

객체 또는 값을 생성자를 통해 주입 받는다.

• <constructor-arg> : <bean>의 하위태그로 설정한

bean 객체 또는 값을 생성자를 통해 주입하도록 설정

– 설정 방법 : <ref>,<value>와 같은 하위태그를 이용하여

설정, 속성을 이용해 설정

– 하위태그 이용

설정을 통한 객체 주입 – Constructor를 이용(1/4)

http://edu.kosta.or.kr

• <ref bean=“bean name”/> - 객체를 주입 시

• <value>값</value> - 문자(String), Primitive data 주입 시

– type 속성 : 값을 1차로 String으로 처리한다. 값의 타입을 명시해야 하

는 경우 사용. ex) <value type=“int”>10</value>

– 속성 이용

• ref=“bean 이름”

• value=“값”

|  |  |
| --- | --- |
| package vo;  public class Person{  private String id,  private String name,  private int age;  public Person(String id){...} //1번 생성자  public Person(String id, String name){…} //2번 생성자  public Person(int age){…} //3번 생성자  } | 값을 주입 받을 객체  http://edu.kosta.or.kr  22  <bean id=“person” class=“vo.Person”>  <constructor-arg>  <value>abcde</value>  </constructor-arg>  </bean>  또는  <bean id=“person” class=“vo.Person”>  <constructor-arg value=“abc”/>  </bean>  <bean id=“person” class=“vo.Person”>  <constructor-arg>  <value>abcde</value>  </constructor-arg>  <constructor-arg>  <value>Hong Gil Dong</value>  </constructor-arg>  </bean>  또는  <bean id=“person” class=“vo.Person”>  <**constructor-arg value=“abc”**/> |
| <constructor-arg value=“Hong Gil Dong”/>  </bean>  ------------  <bean id=“person” class=“vo.Person”>  <constructor-arg>  <value type=“int”>30</value>  </constructor-arg>  </bean>  또는  <bean id=“person” class=“vo.Person”>  <constructor-arg value=“abc” type=“int”/>  <constructor-arg value=“Hong Gil Dong”/>  </bean> |

Bean 객체 주입

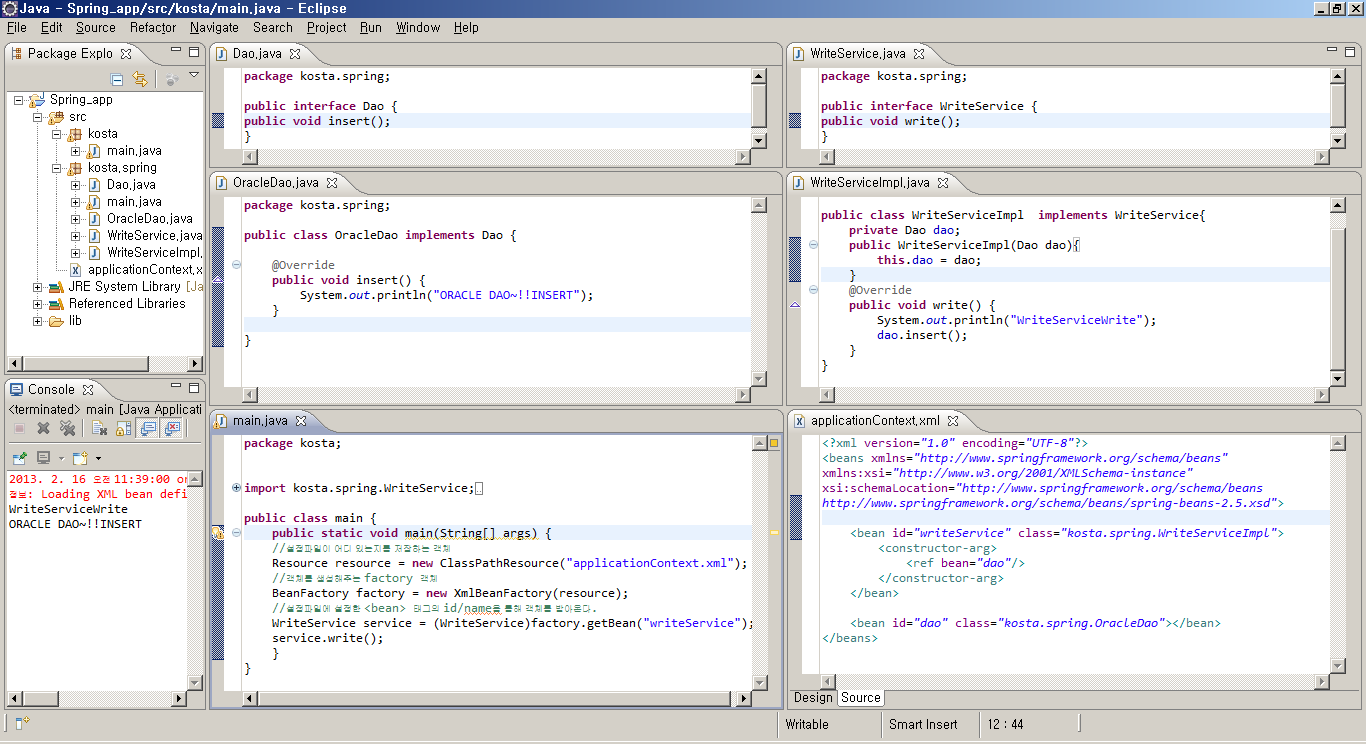
|  |  |
| --- | --- |
| public class BusinessService{  private Dao dao = null;  public BusinessService(Dao dao){  this.dao = dao;  }  } | <bean id=“dao” class=“spring.di.model.OracleDAO”/>  <bean id=“service” class=“spring.di.model.service.BusinessService”>  <constructor-arg>  <ref bean = “dao”/>  </constructor-arg>  </bean>  또는  <bean id=“service” class=“spring.di.model.service.BusinessService”>  <**constructor-arg ref=“dao”**>  </bean> |

설정을 통한 객체 주입 – Constructor를 이용

값을 주입 받을 객체

http://edu.kosta.or.kr

24

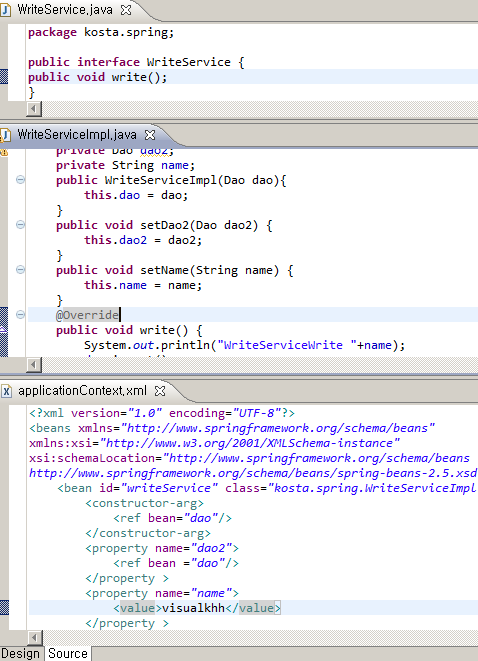


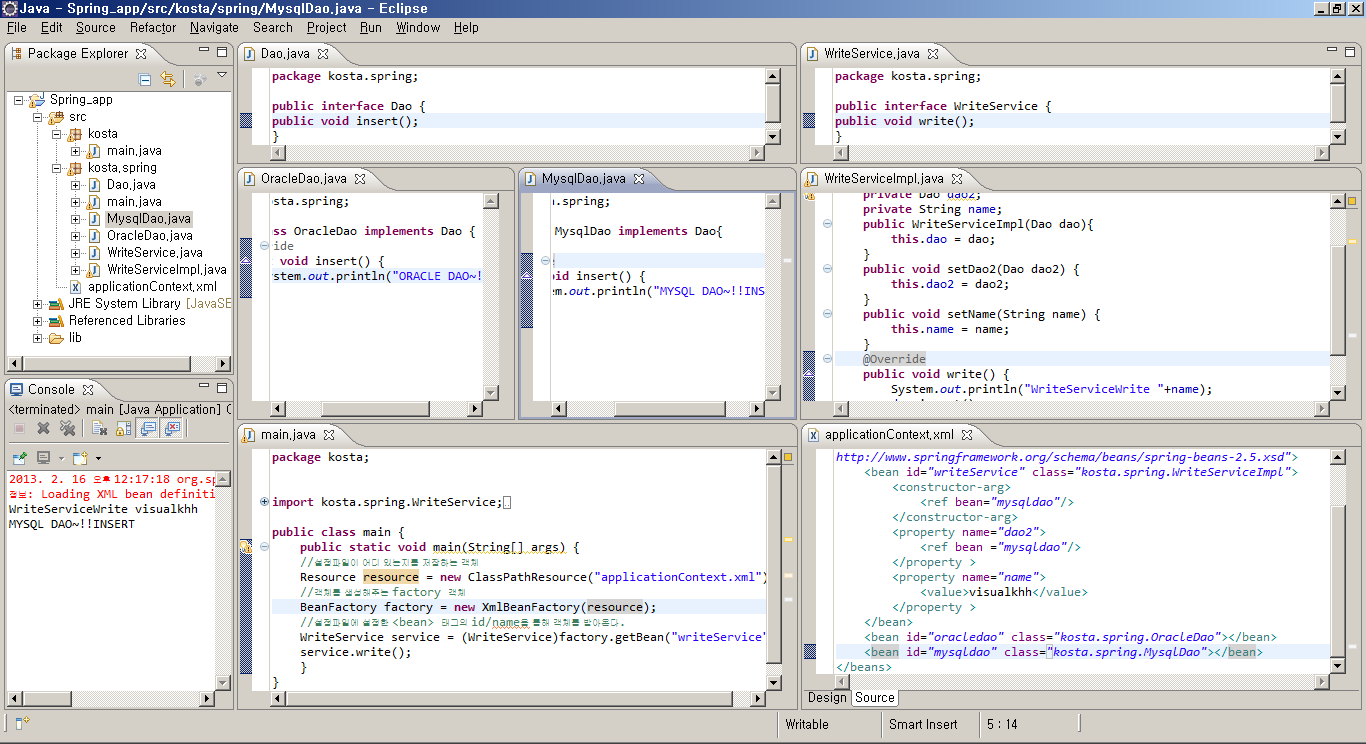
설정을 통한 객체 주입 – Property를 이용

|  |  |
| --- | --- |
| package spring.vo;  public class Person{  private String id,  private String name,  private int age;  public void setId(String id) {…}  public void setName(String name) {…}  public void setAge(int age) {…}  값을 주입 받을 객체 | Primitive Data Type 주입  http://edu.kosta.or.kr  27  <bean id=“person” class=“vo.Person”>  <property name="name">  <value>hong</value>  </property>  <property name="id" value="abcde"/>  <property name="age" value="20"/>  </bean> |

|  |  |
| --- | --- |
| public class BusinessService{  private Dao dao = null;  public setDao(Dao dao){…}  } | 값을 주입 받을 객체  <bean id=“dao” class=“spring.di.model.OracleDAO”/>  Bean 객체 주입  <bean id=“service” class=“spring.di.model.service.BusinessService”>  <property name=“dao”>  <ref bean =“dao”/>  </property >  </bean>  또는  <bean id=“service” class=“spring.di.model.service.BusinessService”>  < property name=“dao” ref=“dao”>  </bean> |

|  |  |
| --- | --- |
| public class BusinessService{  private Dao dao = null;  private int waitingTime = 0;  public setDao(Dao dao){…}  public setWaitingTime(int wt){…}  } | <?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-2.5.xsd"  **xmlns:p="http://www.springframework.org/schema/p"**  >  <bean id=“dao” class=“spring.di.model.OracleDAO”/>  <bean id="person" class="vo.Person"  **p:waitingTime=“20“**  **p:dao-ref=“dao**"/>  </beans> |





Collection 객체 주입하기

<property> 또는 <constructor-arg>의 하위 태그로 Collection

값을 설정하는 태그를 이용해 값 주입 설정

• 설정 태그

태그 Collection종류 설명

<list> java.util.List List 계열 컬렉션 값 목록 전달

<set> java.util.Set Set 계열 컬렉션 값 목록 전달

http://edu.kosta.or.kr

30

<map> java.util.map Map계열 컬렉션 에 keyvalue

의 값 목록 전달

<props> java.util.Properties Properties 에 key(String)-

value(String)의 값 목록 전달

• Collection에 값을 설정 하는 태그

• <ref> : <bean>으로 등록된 객체

• <value> : 기본데이터

• <bean> : 임의의 bean

• <list>,<map>,<props>, <set> : 컬랙션

• <null> : null

<list>

– List 계열 컬렉션이나 배열에 값들을 넣기.

– <ref>, <value> 태그를 이용해 값 설정

– <ref bean=“bean\_id”/> : bean 객체 list에 추가

– <value [type=“type”]>값</value> : 문자열(String),

Primitive 값 list에 추가

|  |  |
| --- | --- |
| public void setMyList(List list){…} | <bean id=“otherbean” class=“vo.OtherBean”/>  <bean id=“myBean” class=“vo.MyVO”>  <property name=“myList”>  <list>  <value>10</value> ->String으로 저장됨  <value type=“java.lang.Integer”>20</value> ->Integer  로 저장됨  <ref bean=“otherbean”/>  </list>  </property>  </bean> |

http://edu.kosta.or.kr

31

<map>

– Map계열의 Collection에 객체들을 넣기

• 속성 : key-type, value-type : key와 value의 타입을 고정시킬경우 사용

– <entry>를 이용해 key-value를 map에 등록

• 속성

– key, key-ref : key 설정

– value, value-ref : 값 설정

|  |  |
| --- | --- |
| public void setMyMap(Map map){…} | <bean id=“otherbean” class=“vo.OtherBean”/>  <bean id=“myBean” class=“vo.MyVO”>  <property name=“myMap”>  <map>  <entry key=“id” value=“abc”/>  <entry key=“other” value-ref=“otherbean”/>  </map>  </property>  </bean> |

<props>

– java.util.Properties 값(문자열)을 넣기

– <prop>를 이용해 key-value를 properties에 등록

• 속성

– key : key값 설정

• 값은 태그 사이에 넣는다. : <prop key=“id”>abcde</prop>

|  |  |
| --- | --- |
| public void setMySet(Set props)ㄴ…} | <bean id=“otherbean” class=“vo.OtherBean”/>  <bean id=“myBean” class=“vo.Bean”>  <property name=“mySet>  <set>  <value>10</value>  <value>20<value>  <ref bean=“otherbean”/>  </set>  </property>  </bean> |

Bean 객체의 생성 단위

• BeanFactory를 통해 Bean을 요청시 객체생성의

범위(단위)를 설정

• <bean> 의 scope 속성을 이용해 설정

– scope의 값

singleton 컨테이너는 하나의 빈 객체만 생성한다. - default

prototype 빈을 요청할 때 마다 생성한다.

request Http 요청마다 빈 객체 생성

session HttpSession 마다 빈 객체 생성

global-session 글로벌 http세션에 대해 빈 객체 생성- 포틀릿 관련

• request, session은 WebApplicationContext에서만 적용 가능

빈(bean) 생성 제어

빈(bean) 범위 지정

– singleton과 prototype

• <bean id=“dao” class=“dao.OracleDAO”

scope=“prototype”/>

• prototype은 Spring 어플리케이션 컨텍스트에서

getBean으로 빈(bean)을 사용시마다 새로운 인스턴스를

생성함.

• singleton은 Spring 어플리케이션 컨텍스트에서

getBean 으로 빈(bean)을 사용시 동일한 인스턴스를 생

성함.

Factory 메소드로부터 빈(bean) 생성

|  |  |
| --- | --- |
| public class OracleDAO{  private OracleDAO() {}  private static OracleDAO instance;  public static OracleDAO getInstance(){  if(instance==null)  instance = new OracleDAO();  return instance;  }}  Singleton 클래스는 static factory  메소드를 통해서 인스턴스 생성이 가  능하면 단 하나의 인스턴스만을 생성  함. | <bean id=“dao” class=“OracleDAO”  factory-method=“getInstance”/> |

AOP Aspect Oriented Programming

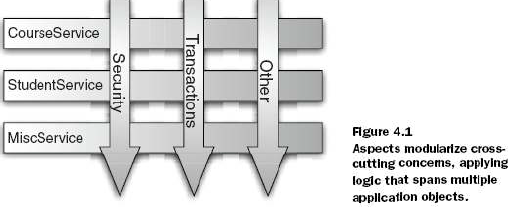
공통관심사들. 로깅이든지..섹션검사등.. 공통모듈등..

핵심 관심 사항(core concern) 과 공통 관심 사항 (cross-cutting concern)

• 기존 OOP 에서는 공통관심사항을 여러 모듈에서 용하는데 중복된 코드를 양산과 같은 한계가 존 - 이를 극복하기 위해 AOP 가 등장

http://edu.kosta.or.kr

• Aspect Oriented Programming은 문제를 해결하기 위한 핵심 관심 사항과 전체에 적용되는 공통관심 사항을 기준으로 프로그래밍함으로써 공통 듈을 손쉽게 적용할 수 있게 해준다.



|  |  |
| --- | --- |
| Aspect(what)뭘 | 여러 객체에서 공통으로 적용되는 공통관심 사항(ex:트랜잭션, 로깅, 보안) |
| JoinPoint | Aspect가 적용 될 수 있는 지점(ex:메소드, 필드) |
| Pointcut(who)누가 | 공통관심사항이 적용될 Joinpoint |
| Advice(when)언제 | 어느 시점(ex: 메소드 수행 전/후, 예외발생 후 등)에 어떤 공통 관심기능(Aspect)을 적용할지 정의한 것 |
| Weaving | 어떤 Advice를 어떤 Pointcut(핵심 사항)에 적용시킬 것  인지에 대한 설정 (Advisor) |
| 다합치면 어드바이저 가된다.ㅋㅋ | |

Spring에서 AOP 구현 방법

AOP 구현의 세가지 방법

– POJO Class를 이용한 AOP구현

– 스프링 API를 이용한 AOP구현

– 어노테이션(Annotation) 을 이용한 AOP 구현

XML 스키마 확장기법을 통해 설정파일을 작성한

다.

**• POJO** 기반 Advice 클래스 작성

**뭐든 객체들이 Aspect 될수있다.**

XML 스키마를 이용한 AOP 설정

– aop 네임스페이스와 XML 스키마 추가

AOP 설정 태그

1. <aop:config> : aop설정의 root 태그. – weaving들의 묶음.

2. <aop:aspect> : Aspect 설정 – 하나의 weaving에 대한 설정

3. <aop:pointcut> : Pointcut 설정

4. Advice 설정태그들

A. <aop:before> - 메소드 실행 전 실행될 Advice

http://edu.kosta.or.kr

B. <aop:after-returning> - 메소드 정상 실행 후 실행될 Advice

C. <aop:after-throwing> - 메소드에서 예외 발생시 실행될 Advice

D. <aop:after> - 메소드 정상 또는 예외 발생 상관없이 실행될 Advice – finally

E. <aop:around> - 모든 시점에서 적용시킬 수 있는 Advice 구현

|  |
| --- |
| <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:p="http://www.springframework.org/schema/p"  **xmlns:aop="http://www.springframework.org/schema/aop"**  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://edu.kosta.or.kr  http://www.springframework.org/schema/beans/spring-beans-2.5.xsd  **http://www.springframework.org/schema/aop**  **http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">**  </beans> |
| <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:p="http://www.springframework.org/schema/p"  xmlns:aop="http://www.springframework.org/schema/aop"  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-2.5.xsd  http://www.springframework.org/schema/aop  http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">  http://edu.kosta.or.kr  <bean id=“**writelog**” class=“org.kosta.spring.LogAspect”/>  **<aop:config>**  **<aop:pointcut id=“publicmethod”**  **expression=“execution(public \* org.kosta.spring..\*(..))”**  **/>**  **<aop:aspect id=“logingAspect” ref=“writelog”>**  **<aop:around pointcut-ref=“publicmethod” method=“logging”/>**  **</aop:aspect>**  **</aop:config>**  <bean id=“targetclass” class=“org.kosta.spring.TargetClass”/>  </beans> |

<aop:aspect> (4/5)

• 한 개의 Aspect (공통 관심기능)을 설정

• ref 속성을 통해 공통기능을 가지고 있는 bean을 연결한다.

• id는 이 태그의 식별자를 설정

• 자식 태그로 <aop:pointcut> advice관련 태그가 올수 있다.

|  |
| --- |
| <aop:config>  <aop:aspect id=“logingAspect” ref=“writelog”>  <aop:pointcut id=“publicmethod” expression=“execution(public \* public \* org.myspring..\*.\* (..))”/>  <aop:around pointcut-ref=“publicmethod” method=“logging”/>  </aop:aspect>  </aop:config> |

<aop:pointcut> (5/5)

• Pointcut(공통기능이 적용될 곳)을 지정하는 태그

• <aop:config>나 <aop:aspect>의 자식 태그

– <aop:config> 전역적으로 사용

– <aop:aspect> 내부에서 사용

• AspectJ 표현식을 통해 pointcut 지정

:

http://edu.kosta.or.kr

• 속성 – id : 식별자로 advice 태그에서 사용됨

– expression : pointcut 지정

<aop:pointcut id=“publicmethod” expression=“execution(public \*

org.myspring..\*.\*(..))”/>

|  |
| --- |
| <aop:config>  <aop:aspect id=“logingAspect” ref=“writelog”>  <aop:pointcut id=“publicmethod” expression=“execution(public \* org.myspring..\*.\*(..))”/>  <aop:around pointcut-ref=“publicmethod” method=“logging”/>  </aop:aspect>  </aop:config> |

AspectJ 표현식 (1/3)

• AspectJ에서 지원하는 패턴 표현식

• 스프링은 메서드 호출관련 명시자만 지원

• 명시자

명시자(제한자패턴? 리턴타입패턴 패키지패턴?이름패턴(파라미터패턴))

-?는 생략가능

http://edu.kosta.or.kr

– execution : 실행시킬 메소드 패턴을 직접 입력하는 경우

– within : 메소드가 아닌 특정 타입에 속하는 메서드들을 설정할

경우

– bean : 2.5버젼에 추가됨. 설정파일에 지정된 빈의 이름(name

속성)을 이용해 pointcut설정

AspectJ 표현식 (2/3)

• 표현

• 수식어 패턴에는 public, protected 또는 생략한다.

– \* : 1개의 모든 값을 표현

명시자(수식어패턴? 리턴타입패턴 패키지패턴? 클래스이름패턴.메소드이름패턴(파라미터패턴))-?는 생략가능

예) execution(public \* abc.def..\*Service.set\*(..)

http://edu.kosta.or.kr

• argument에서 쓰인 경우 : 1개의 argument

• package에 쓰인 경우 : 1개의 하위 package

– .. : 0개 이상

• argument에서 쓰인 경우 : 0개 이상의 argument

• package에 쓰인 경우 : 0개의 이상의 하위 package

– 위 예 설명

적용 하려는 메소드들의 패턴은 public 제한자를 가지며 리턴 타입에는모든 타입이 다 올 수 있다. 이름은 abc.def 패키지와 그 하위 패키지에

있는 모든 클래스 중 Service로 끝나는 클래스들 에서 set으로 시작하는메소드이며 argument는 0개 이상 오며 타입은 상관 없다.

|  |
| --- |
| execution(\* test.spring.\*.\*())  execution(public \* test.spring..\*.\*())  execution(public \* test.\*.\*.get\*(\*))  execution(String test.spring.MemberService.registMember(..))  execution(\* test.spring..\*Service.regist\*(..))  http://edu.kosta.or.kr  execution(public \* test.spring..\*Service.regist\*(String, ..))  within(test.spring.service.MemberService)  within(test.spring..MemberService)  within(test.spring.aop..\*)  bean(memberService)  bean(\*Service) |

public \* abc.def..\*Service.set\*(..)

|  |  |  |
| --- | --- | --- |
| public | \* | avc.def..\*Service.set\*(..) |
| 접근제한자 | 리턴값 | abc.def 패키지와 그 하위 패키지에  있는 모든 클래스 중 Service로 끝나는 클래스들 에서 set으로 시작하는메소드이며  argument는 0개 이상 오며 타입은 상관 없다 |

Advice 작성

• POJO 기반 Aspect클래스 작성

– 설정 파일의 advice 관련 태그에 맞게 작성한다.

– <bean>으로 등록 하며 <aop:aspect> 의 ref 속성으

로 참조한다.

– 공통 기능 메소드 : advice 관련 태그들의 method 속

http://edu.kosta.or.kr

성의 값이 메소드의 이름이 된다.

속성

– pointcut-ref : <aop:pointcut>태그의 id명을 넣어 pointcut지정

– pointcut : 직접 pointcut을 설정 한다.

– method : Aspect bean에서 호출할 메소드명 지정

|  |
| --- |
| <bean id=“writelog” class=“org.kosta.spring.LogAspect”/>  <aop:config>(어드바이즈)  <aop:aspect id=“logingAspect” ref=“writelog”> (뭘)  <aop:pointcut id=“publicmethod” expression=“execution(public \*org.my.spring..\*.\*(..))”/>  (누가)  <aop:before pointcut-ref=“publicmethod” method=“logging”/>  (언제)  </aop:aspect>  </aop:config> |

POJO 기반 AOP구현 - Aspect클래스 작성 (1/4)

POJO 기반의 클래스로 작성한다.

– 클래스 명이나 메서드 명에 대한 제한은 없다.

– 메소드 구문은 호출되는 시점에 따라 달라 질 수 있다.

– 메소드의 이름은 advice 태그(<aop:before/>)에서method 속성의 값이 메소드 명이 된다.

• before 메소드

http://edu.kosta.or.kr

– 대상 객체의 메소드가 실행되기 전에 실행됨

– return type : void

– argument : 없거나 JoinPoint 객체를 받는다.

– ex) public void beforeLogging(JoinPoint jp){

}

After Returning Advice

– 대상객체의 메소드 실행이 정상적으로 끝난 뒤 실행됨

– return type : void

– argument :

• 없거나 JoinPoint 객체를 받는다. JoinPoint는 항상 첫 argument

로 사용된다.

http://edu.kosta.or.kr

• 대상 메소드에서 리턴되는 값을 argument로 받을 수 있다.

|  |
| --- |
| <aop:after-returning pointcut-ref=“publicmethod” method=“returnLogging” returning=“retValue”/>  public void returnLogging(Object retValue){  //대상객체에서 리턴되는 값을 받을 수는 있지만 수정할 수는 없다.  } |

After Throwing Advice

– 대상객체의 메소드 실행 중 예외가 발생한 경우 실행됨

– return type : void

– argument :

• 없거나 JoinPoint 객체를 받는다. JoinPoint는 항상 첫 argument

로 사용된다.

http://edu.kosta.or.kr

• 대상메소드에서 전달되는 예외객체를 argument로 받을 수 있다.

|  |
| --- |
| <aop:after-throwing pointcut-ref=“publicmethod” method=“returnLogging” throwing=“ex”/>  public void returnLogging(MyException ex){  //대상객체에서 리턴되는 값을 받을 수는 있지만 수정할 수는 없다.  } |

Around Advice

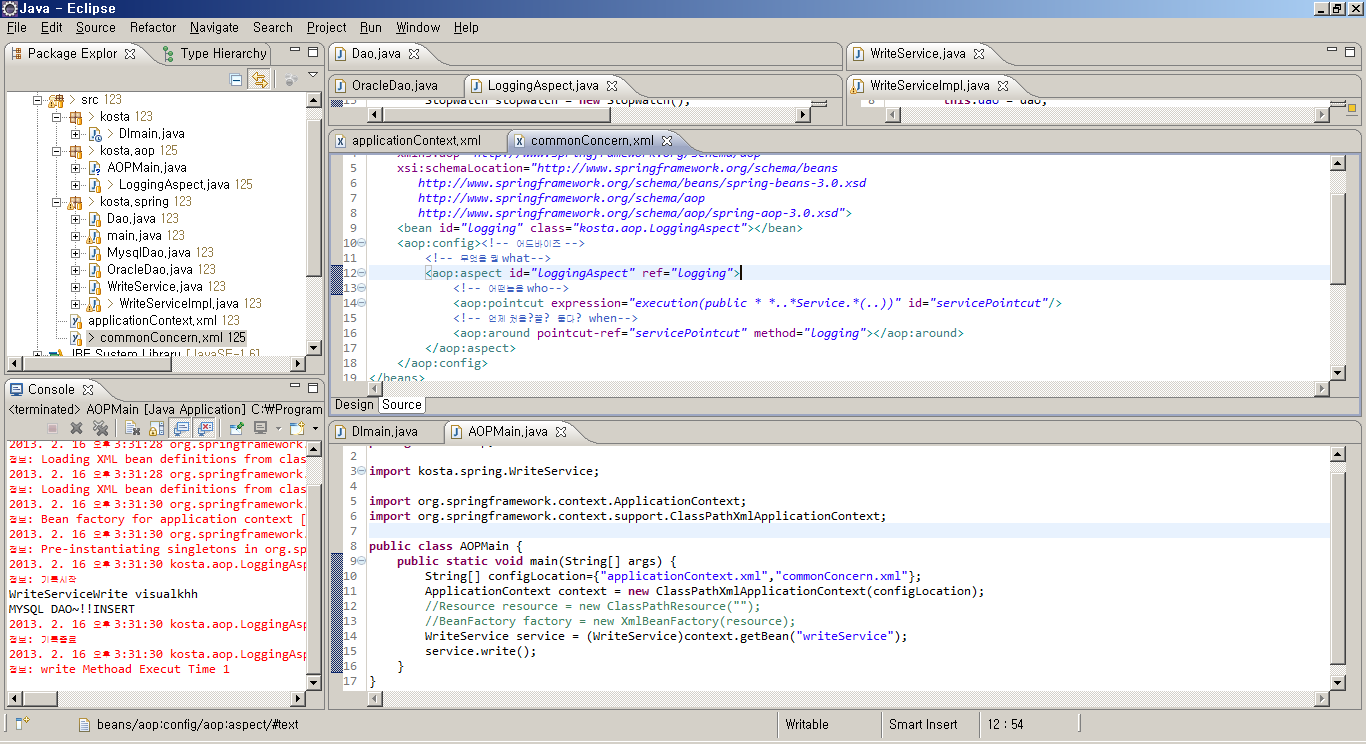
– 위의 네 가지 Advice를 다 구현 할 수 있는 Advice.

– return type : Object

– argument

• org.aspectj.lang.ProceedingJoinPoint 를 반드시 첫 argument로 지정한다.

|  |
| --- |
| <aop:around pointcut-ref=“publicmethod” method=“returnLogging” />  public Object returnLogging(ProceedingJoinPoint joinPoint) throws Throwable{  //대상 객체의 메소드 호출 전 해야 할 전 처리 코드  try{  Object retValue = joinPoint.proceed(); //대상객체의 메소드 호출  //대상 객체 처리 이후 해야 할 후처리 코드  return retValue; //호출 한 곳으로 리턴 값 넘긴다. – 넘기기 전 수정 가능  }catch(Throwable e){  throw e; //예외 처리  }  } |



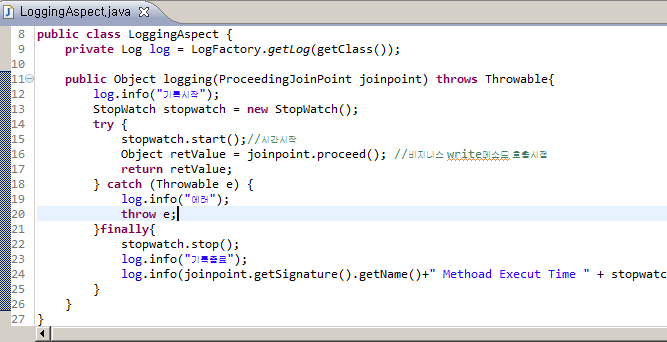
aspect 무엇을

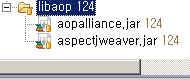
pointcut 어떤놈을

언제

1. around(비즈니스 전후가능하게(Object retValue = joinpoint.proceed(); //비지니스 write메소드 호출시점)
2. after
3. before

ApplicationContext를 써야된다





JoinPoint

• 대상객체에 대한 정보를 가지고 있는 객체로 Spring container로

부터 받는다.

• org.aspectj.lang 패키지에 있음

• 받듯이 Aspect 메소드의 첫 argument로 와야한다.

• 메소드들

http://edu.kosta.or.kr

Signature : 호출 되는 메소드에 대한 정보를 가진 객체

Object getTarget() : 대상객체를 리턴

Object[] getArgs() : 파라미터로 넘겨진 값들을 배열로 리턴. 넘어온 값이 없으면 빈 배열개체가 return 됨.

Signature getSignature () : 호출 되는 메소드의 정보

String getName() : 메소드 명

String toLongString() : 메서드 전체 syntax를 리턴

String toShorString() : 메소드를 축약해서 return – 기본은 메소드 이름만 리턴

@Aspect 어노테이션을 이용한 AOP

• @Aspect 어노테이션을 이용하여 Aspect 클래스에 직접

Advice 및 Pointcut등을 직접 설정

• 설정파일에 <aop:aspectj-autoproxy/> 를 추가 필요

• Aspect class를 <bean>으로 등록

• 어노테이션(Annotation)

– @Aspect : Aspect 클래스 선언

– @Before(“pointcut”)

http://edu.kosta.or.kr

– @AfterReturning(pointcut=“”, returning=“”)

– @AfterThrowing(pointcut=“”, throwing=“”)

– @After(“pointcut”)

– @Around(“pointcut”)

• Around를 제외한 나머지 메소드들은 첫 argument로

JoinPoint를 가질 수 있다.

• Around 메소드는 argument로 ProceedingJoinPoint를

가질 수 있다.

Spring MVC

어떤식으로 맵핑할껀데.이름?

URL과 컨트롤을 어떻게 맵핑할거임?

어떤뷰로 .. 타일즈나,jsp나

xml이나..뷰종류를 정한다

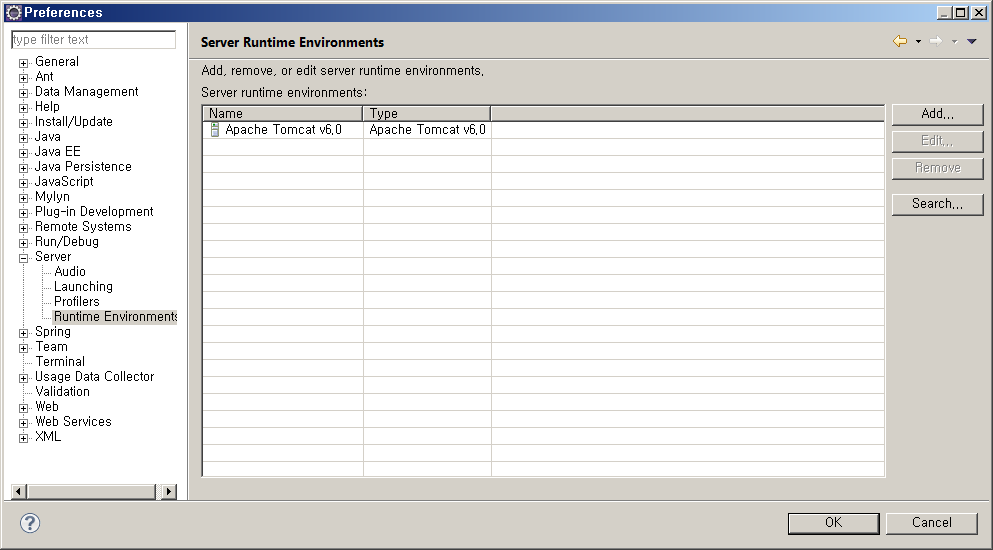
어떤뷰로 갈래? 어떤jsp로갈래?

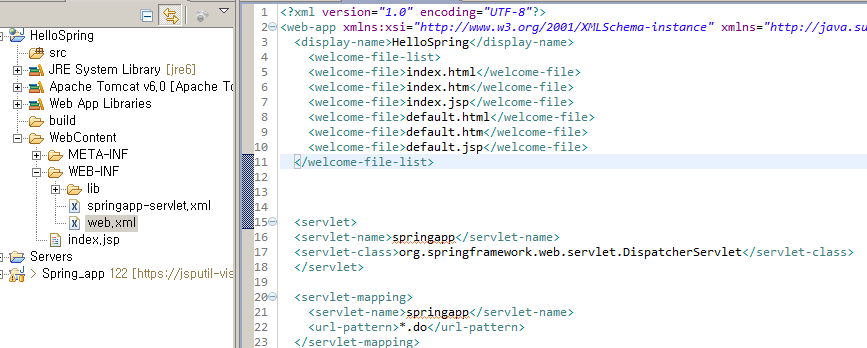
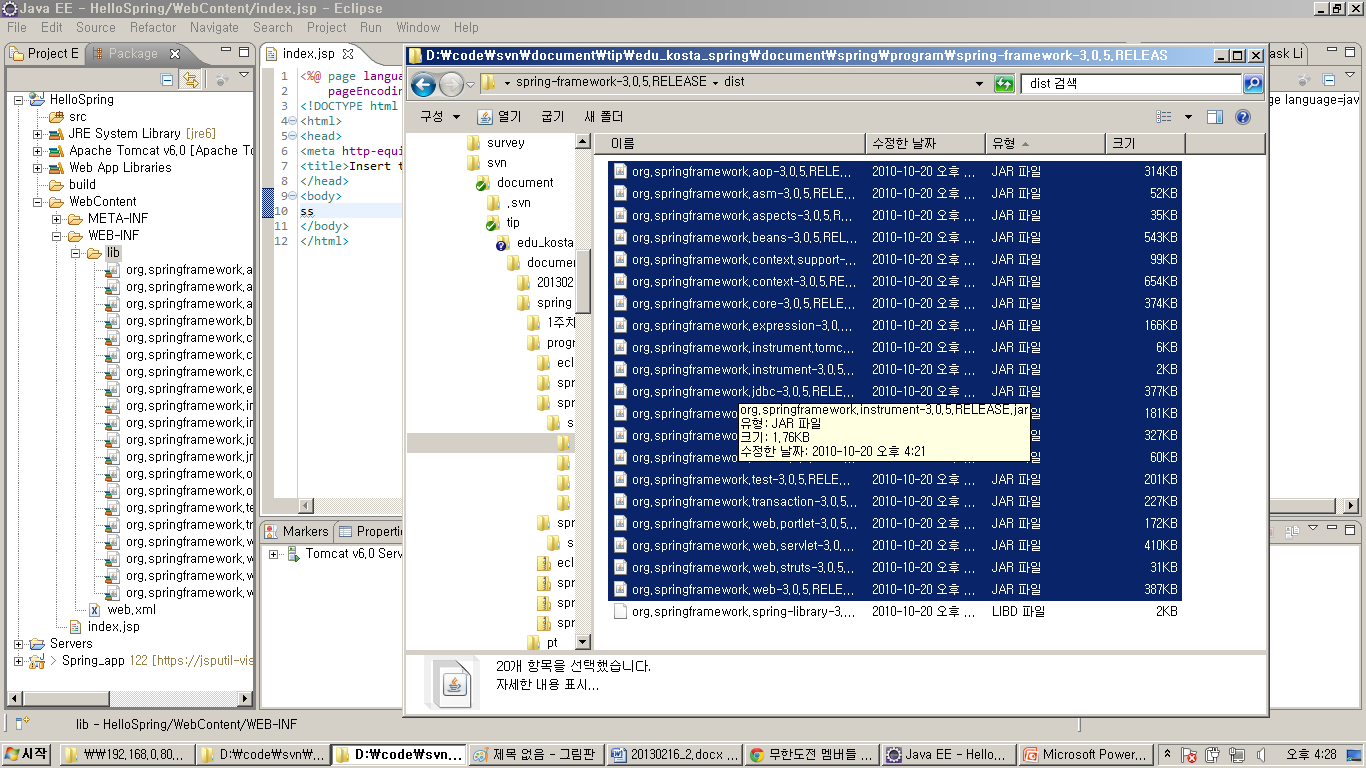
어떤 파라미터값으로 가져갈래

어떤비지니스 처리하자~

* **요청 처리 순서** 
  1. DispatcherServlet이 요청을 수신
     + 단일 front controller servlet
     + 요청을 수신하여 처리를 다른 컴포넌트에 위임
     + 어느 컨트롤러에 요청을 전송할지 결정
  2. DispatcherServlet은 Handler Mapping에 어느컨트롤러를 사용할 것인지 문의
     + URL과 매핑
  3. DispatcherServlet은 요청을 컨트롤러에게 전송하고 컨트롤러는 요청을   
     처리한 후 결과 리턴
     + 비즈니스 로직 수행 후 결과 정보(Model)가 생성되어 JSP와 같은 뷰에서 사용됨
  4. ModelAndView 오브젝트에 수행결과가 포함되어 DispatcherServlet에 리턴
  5. ModelAndView는 실제 JSP정보를 갖고 있지 않으며, ViewResolver가 논리적 이름을 실제 JSP이름으로 변환
  6. View는 결과정보를 사용하여 화면을 표현함

서버시작





어노테이션으로 작성 3.0부터는 거의다 어노테이션으로바뀐다

* **컨트롤러 클래스 작성**
* **@Controller**
* **public class HelloController {**
* **@RequestMapping(“hello.do")**
* **public String sayHello(Model model) {**
* model.addAttribute(“message”, “안녕하세요”);
* **return “hello";**
* }

**컨텍스트 설정파일([servletName]-servlet.xml)에 컨트롤러 설정 (dispatcher-servlet.xml )**

<?xml version=*"1.0" encoding="UTF-8"?>*

<beans xmlns=*"http://www.springframework.org/schema/beans"*

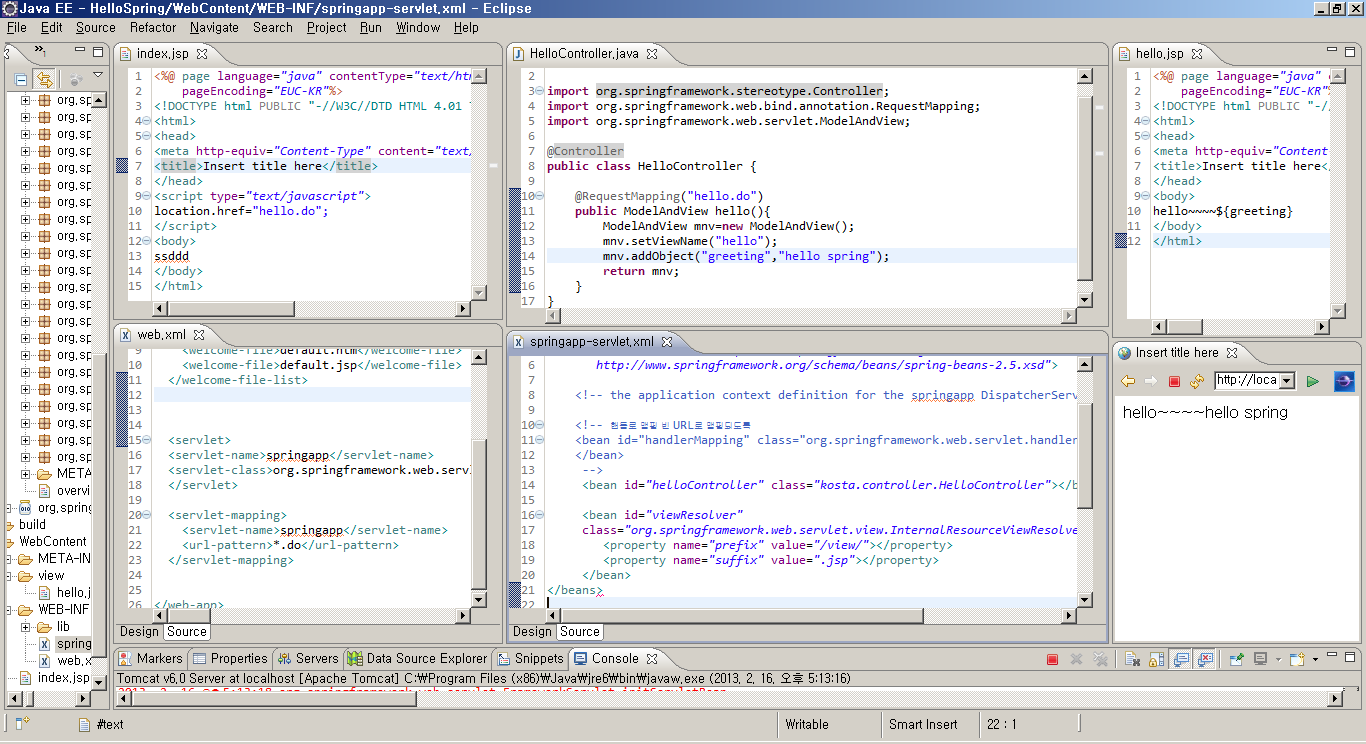
xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*" http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">*

**<bean id=*“helloController" class="com.kosta. HelloController“*/>**

</beans>

* **@Controller 와 @RequestMapping 선언** 
  + 메소드 단위의 매핑이 가능.
  + DefaultAnnotationHandlerMapping와 AnnotationHanderAdapter 를 사용함.
    - **스프링 3.0의 기본 설정이므로 별도의 추가 없이 사용 가능.**
    - **하지만 동시에 여러 개사용할때는 적어줘야합니다.**



/view/hello.jsp

기본3.0은어노테이션 맵핑핸들러라 가능