

목 차

1. 스프링 프레임워크 개념 & 모듈 구성	3
1-1 스프링 프레임워크 개념	3
1-2 스프링 프레임워크 설치와 모듈	3
2. 사용시장분석_활용사례 (정부표준프레임워크 선정)	5
2-1 스프링 사용목적	5
2-2 실무에서 개발프레임워크 적용범위	7
2-3 정부 표준프레임워크	8
3. 개발 환경 및 개발 시 고려사항	11
3-1 개발에 필요한 프로그램 다운로드	11
3-2 개발환경 설정	15
4. 스프링 프레임워크 기술 학습	18
4-1 Spring DI(Dependency Injection)	18
4-2 Spring AOP(Aspect Oriented Programming)	43
4-3 Spring Database연동	54
4-4 Spring Transaction	61
4-5 Spring MVC	63
5. 스프링 프레임워크 활용 프로젝트	74
5-1 스프링 MVC패턴을 이용한 게시판 프로젝트구성	74

1. 스프링프레임워크 개념 & 모듈 구성

1-1. 스프링 프레임워크 개념

1 spring이란?

- spring framework는 Enterprise Application에서 필요로 하는 기능을 제공하는 오픈 소스 프레임워크다.
- spring framework는 평범한 자바객체(POJO : Plain Old Java Object)를 이용해서 단순하고, 테스트하기 쉬우며, 객체간의 결합이 느슨한 Enterprise Application 개발이 가능하도록 지원한다.

2 spring framework의 특징

- 스프링은 자바객체를 담고 있는 경량의 컨테이너다. 이들 자바객체의 생성 소멸과 같은 라이프사이클을 관리한다.
- 스프링은 의존성 주입(Dependency Injection)을 지원한다.
- 스프링은 관점지향 프로그래밍(Asspect Oriented Programming)을 지원한다.
- 스프링은 POJO(Plain Old Java Object)를 지원한다.
- 스프링은 트랜잭션 처리를 위한 일관된 방법을 제공하며, 설정파일을 통해 트랜잭션 관련정보를 선언적으로 정의할 수 있다.
- 스프링은 다양한 ORM(Object-Relation Mapping)툴과의 연동을 지원한다.
- 스프링은 Enterprise Application개발에 필요한 다양한 API를 지원한다.

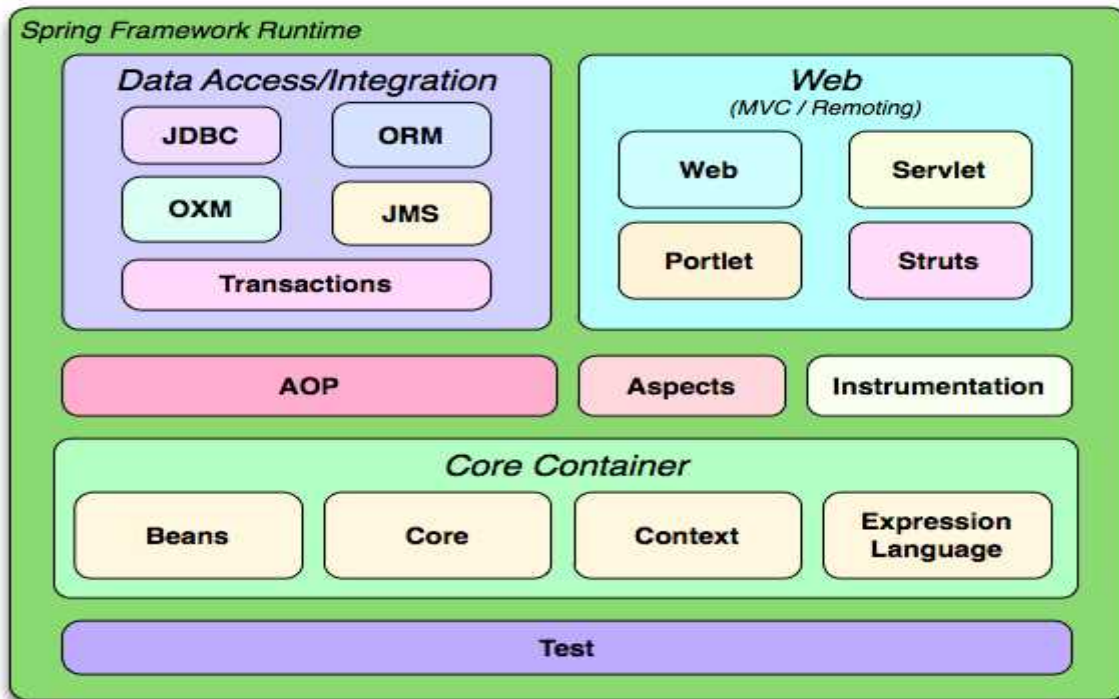
1-2. 스프링 프레임워크 설치와 모듈

1. 스프링3.0 프레임워크 설치

- <http://www.springsource.org/download>
- spring-framework-3.0.5.RELEASE-with-docs.zip (sha1) 46.8 MB : 스프링 프레임워크의 모듈과 소스코드, 레퍼런스 문서가 포함되어 있다.

■ spring-framework-3.0.5.RELEASE.zip (sha1) 24.8 MB :스프링 프레임워크에서 사용하는 의존하는 의존 jar파일 목록이 포함되어 있다.

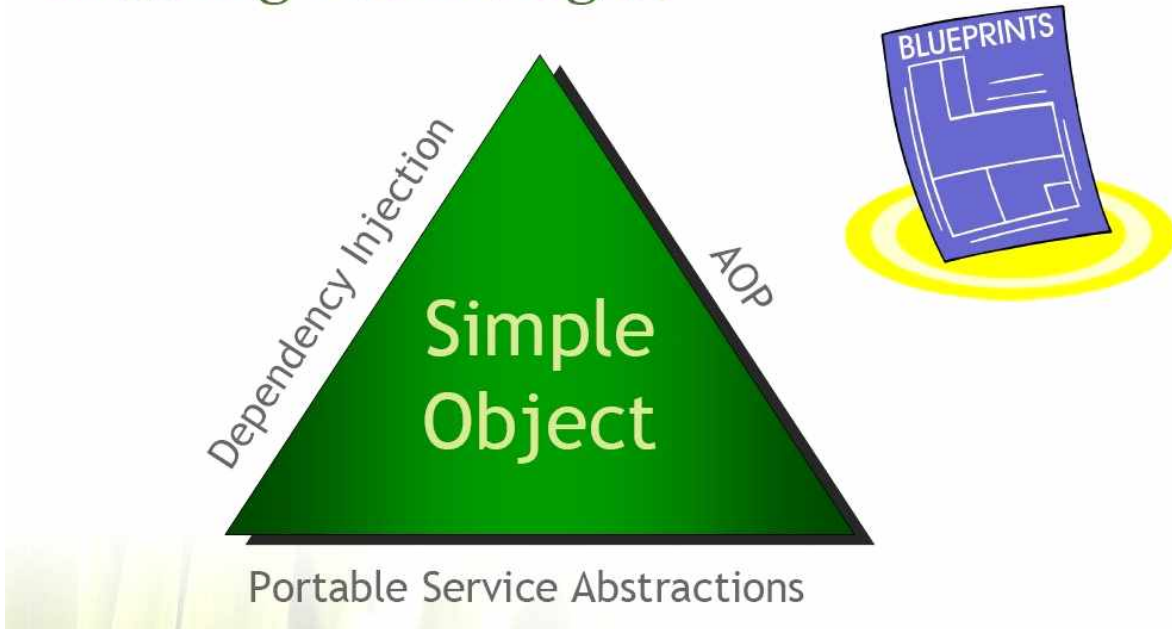
2. 스프링3.0 프레임워크의 모듈



- Spring Bean : BeanFactory인터페이스를 통해 구현된다.
- Spring Core : spring framework의 핵심기능인 의존성 주입기능을 지원하는 모듈이다.
- Spring Context : Spring Core에서 지원하는 기능 외의 추가적인 기능을 지원하고, jndi연결이나 ejb연계를 위한 Adapter를 제공하는 모듈이다.
- Spring Expression : 객체에 접근하고 객체를 조작하기 위한 표현언어를 제공하는 모듈이다.
- Spring AOP : 관점지향 프로그래밍을 지원하는 모듈이다.
- Spring aspect : AspectJ와의 통합을 제공하는 모듈이다.
- Spring Instrumentation : Instrumentation 지원 클래스를 제공한다.
- Spring Transactions : AOP를 이용한 선언적 트랜잭션 관리 및 코드를 이용한 트랜잭션관리 기능을 제공한다.
- Spring OXM : 객체와 XML사이의 매핑을 처리하기 위한 추상 레이어를 제공한다. JAXB, Castor, XMLBeans, JiBX, XStream과의 연동을 지원한다.
- Spring JMS : JMS의 메시지를 생성하고 수신하는 기능을 제공하는 모듈이다.
- Spring JDBC : JDBC프로그래밍을 위한 추상레이어를 제공한다. JDBC템플릿을 제공함으로써 간결한 코드로 JDBC프로그래밍을 할 수 있다.
- Spring ORM : ORM(Object-Relation Mapping)툴과의 연동을 지원하는 모듈이다.

- Spring Portlet : 포틀릿 환경에서 사용되는 MVC구현을 제공한다.
- Spring Struts : 스프링과 스트럿츠 연동을 제공하는 모듈이다.
- Spring Web : Web Application개발에 적합한 Web Application Context를 제공하며, Mulitpart 요청을 처리할 수 있다. Struts와 같은 다른 프레임워크와의 연동을 지원한다.
- Spring Servlet : 스프링 MVC를 제공한다. JSP, Velocity에 대한 뷰 연동을 지원한다.

Enabling Technologies



2. 사용시장분석_활용사례(정부표준프레임워크 선정)

2-1. 스프링 사용목적

스프링을 사용하는 이유는 관점에 따라 비교하는 대상에 따라서 너무나 상대적이고, EJB와 비교할 때와 스트럿츠나 HiveMind와 같은 프레임워크와 비교할 때의 장,단점이 틀리기 때문에 한가지로 말하기는 어려우나 2가지 경우를 예로 들어보면 다음과 같다.

1. Spring 프레임워크가 테스트하기 쉬운 구조로 코드를 개발하는 것이 가능하도록 지원한다는 것이다.

Spring 프레임워크가 기본적으로 제공하고 있는 AbstractTransactionalSpringContextTests,

AbstractTransactionalDataSourceSpringContextTests 클래스 때문만이 아니라 Spring 프레임워크의 핵심 기능인 Dependency Injection은 개발자들이 원하건 원하지 않건 테스트하기 쉬운 구조로 코딩하는 것이 가능하도록 지원하게 된다.

처음에는 단위테스트 코드를 작성하지 않더라도 추후에 단위 테스트 코드를 작성할 때 그 위력을 느낄 수 있다.

Spring 프레임워크가 지원하는 Setter Injection은 디자인 패턴의 Strategy Pattern을 근간으로 하고 있기 때문에 이 같은 효과를 가져올 수 있다. 개발 경력이 많은 자바 개발자라 할지라도 테스트 용이한 소스 코드를 구현하는 것이 생각만큼 쉽지 않다.

그러나 Spring 프레임워크를 이용할 경우 이 같은 효과를 자연스럽게 얻을 수 있게 된다.

초보 개발자라 할지라도 Spring 프레임워크 기반으로 개발하게 될 경우 처음에는 그 가치를 느끼지 못하겠지만 개발 경험이 쌓이면서 그 가치를 이해하게 되면서 점점 더 테스트 용이한 코드를 만들어 낼 수 있도록 유도한다는 측면에서 큰 점수를 주고 싶다.

2. 인스턴스의 생성과 클래스간의 의존관계에 대한 고민을 덜어주고 있다는 측면이다. 인스턴스의 생성에 대한 측면이 더 강하다고 생각된다. 지금까지 많은 자바 개발자들은 인스턴스를 Singleton으로 생성할지 Non Singleton으로 생성할지에 대한 고민을 거의 하지 않은 것이 사실이다. 개인적으로 인스턴스의 생성과 의존관계에 기본이 잘 되어 있는 애플리케이션의 경우 유지보수와 확장성이 좋다고 생각한다.

그러나 개발자들이 이 같은 효과를 얻기 위해서는 디자인 패턴까지 이해한 상황에서 설계가 진행되어야 한다. 그러나 국내 현실에서 이 같은 부분까지 고민하면서 개발하기란 쉽지 않은 것이 사실이다. Spring 프레임워크는 개발자가 의도하지 않더라도 인스턴스의 생성에서부터 의존관계를 관리하는 것이 가능하도록 지원해준다.

인스턴스의 생성에서는 Singleton 인스턴스 생성을 위한 static을 사용하지 않아도 됨으로 좀 더 OOP적인 개발이 가능하도록 유도한다. 또한 인터페이스의 사용을 유도함으로써 각 클래스, 컴포넌트간의 의존관계를 줄여주는 효과를 얻을 수 있다.

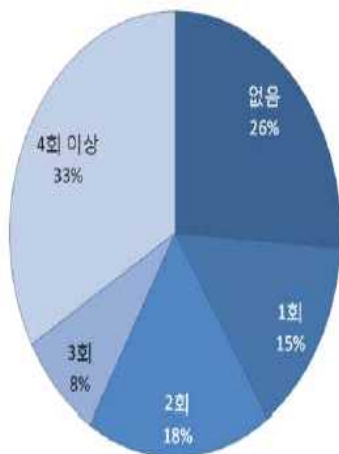
이 같은 모든 효과는 애플리케이션의 개발을 완료한 다음 유지보수업무를 진행할 때 큰 효과를 볼 수 있다.

물론 애플리케이션을 개발하는 도중에도 무수히 많은 유지보수성(기능의 변경) 업무가 발생하기 때문에 애플리케이션을 개발하는 도중에도 큰 효과를 얻을 수 있게 된다.

2-2. 실무에서 개발프레임워크 적용 범위

1. 행정안전부에서 개발프레임워크가 프로젝트 현장에서 어느 정도 적용되고 있는지 실태를 묻는 설문에서, **74%**가 적용 경험이 있는 것으로 나타났으며, 기업규모가 클수록 경험자 비중이 높은 경향이 있다.

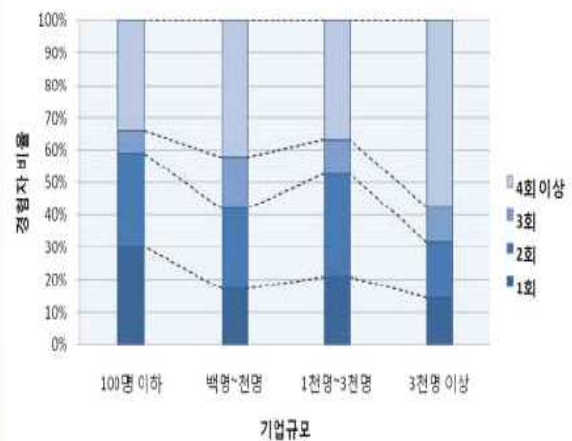
개발프레임워크 적용 경험 분석



※자료: 전자정부 개발프레임워크 표준화를 위한 인식도 조사(2008)

- 전체 응답자 483명 중 74%가 개발프레임워크 적용 경험이 있다고 응답하였으며, 그 중 33%는 4회 이상 경험하였다고 응답함
- IT 프로젝트에 개발프레임워크의 적용이 일반화되고 있음을 확인할 수 있음

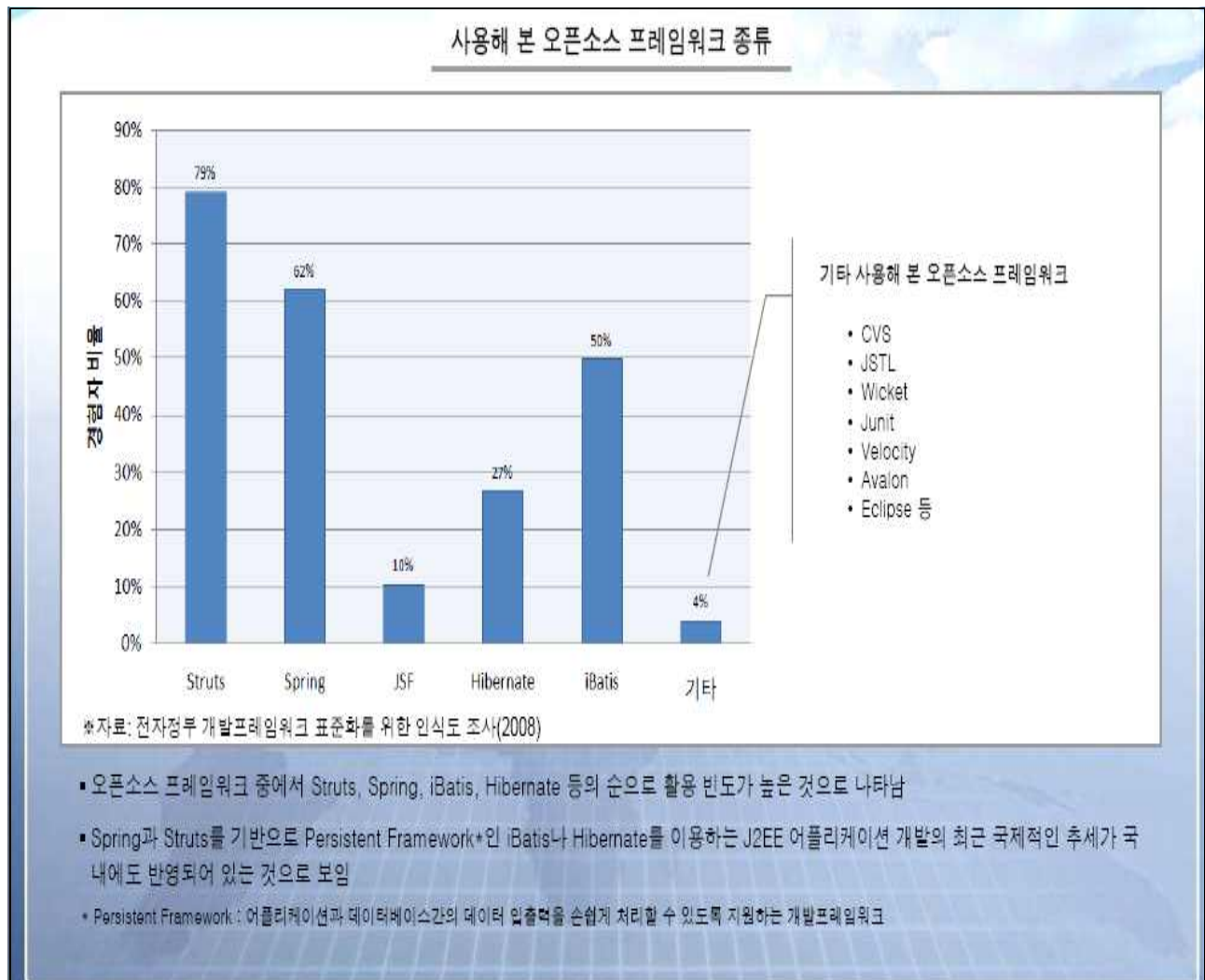
기업규모별 개발프레임워크 적용 경험 분석



※자료: 전자정부 개발프레임워크 표준화를 위한 인식도 조사(2008)

- 종업원수 3천명 이상의 대기업에 소속된 인력들의 개발프레임워크 적용 경험이 많은 것으로 나타남
- 대부분의 대기업들이 자사 개발프레임워크를 보유하고 있으며, 이로 인해 상대적으로 대기업 인력들의 개발프레임워크 적용 경험이 풍부한 것으로 보임

2. 보편적으로 많이 사용되는 오픈소스 프레임워크는 **Struts, Spring, iBatis, Hibernate** 등의 순인 것으로 조사됐다. 정부에서는 이를 바탕으로 표준프레임워크를 만들어 개발현장에서 사용할 있도록 제공하고 있다.



2-3. 정부표준프레임워크

1 목적

전자정부 표준 프레임워크는 응용SW의 구성기반이 되며 응용SW실행 시 필요한 기본 기능을 제공하는 환경이다. 전자정부 표준 프레임워크는 '전자정부 서비스의 품질향상 및 정보화 투자 효율성 향상'을 위해 개발 프레임워크 표준을 정립하고, 개발 프레임워크 표준 적용을 통한 응용 SW의 표준화 및 품질과 재사용성 향상을 목표로 한다.

2 실행환경 오픈소스 현황에서 Spring적용범위

개발프레임워크 실행환경 서비스는 오픈소스 소프트웨어에 기반하여 재활용하거나 확장하여

구현되었다. 일부 서비스는 선정 기준을 만족하는 오픈소스 소프트웨어가 선정되지 않았으며 자체 구현하였다.

<표준프레임워크에서 Spring적요범위>

서비스 그룹	서비스	오픈소스 소프트웨어	실행환경 확장 및 자체 개발	비고
화면 처리	Ajax Support	Ajax Tags		
	Internationalization	Apache Commons i18n		
	MVC	Spring MVC	Custom Tag 외 기능 확장	
	Security	Apache Commons Validator		
	UI Adaptor	선정되지 않음		UI Adaptor 연동 매뉴얼 제공
업무 처리	Process Control	Spring Web Flow		
	Exception Handling	Spring	Exception 기능 확장	
데이터 처리	Data Access	iBatis SQL Maps	Spring-iBatis 기능 확장	
	DataSource	Spring		
	ORM	Hibernate		
	Transaction	Spring		
연계 통합	Naming Service Support	Spring		
	Integration Service	선정되지 않음	표준 인터페이스 처리 기능 개발	
	Web Service Interface	CXF	Intergration Service 연계 기능 확장	
공통 기반	AOP	Spring		
	Cache	EHCACHE		
	Compress/Decompress	Apache Commons Compress		
	Encryption/Decryption	java simplified encryption (jasypt)	암호화 기능 확장	
	Excel	Apache POI	Excel 기능 확장	

File Handling	Jakarta Commons VFS	File Access 기능 확장	
File Upload/Download	Apache Commons FileUpload		
FTP	Apache Commons Net		
ID Generation	선정되지 않음	시스템 고유 ID 생성 기능 개발	
IoC Container	Spring		
Logging	Log4j		
Mail	Java Mail		
Marshalling/Unmarshalling	Castor		
Object Pooling	Apache Commons Pool		
Property	Spring	Property 기능 확장	
Resource	Spring		
Scheduling	Quartz		
Server Security	Spring Security	인증, 권한 관리 기능 확장	
String Util	Jakarta ORO	문자열 처리 기능 확장	
XML Manipulation	Apache Xerces 2 , JDOM	XML 처리 기능 확장	

3 기대효과


- 개발 생산성 향상: 공통적으로 필요한 기능을 제공함으로써 개발 중복을 최소화하고 기반 구조를 정의함으로써 개발자는 비즈니스 업무에 집중할 수 있도록 함으로써 개발 생산성을 향상시킨다.
- 전자정부 시스템의 재사용성 향상: 개발프레임워크 표준화를 통해 전자정부 표준 프레임워크에서 개발된 사업 컴포넌트를 타사업에서 사용함으로써 재사용성을 향상시킨다.
- 전자정부 상호운용성 향상: 전자정부 표준 프레임워크 사용 시스템간 연계 표준 인터페이스를 사용함으로써 상호 운용성을 향상합니다.
- 전자정부 응용 소프트웨어 표준화 효과: 화면처리/업무처리/데이터처리의 표준화된 개발 기반을 제공함으로써 개발 코드의 표준화를 유도한다..
- 오픈 소스 활성화: 오픈 소스에 기반한 표준 프레임워크를 정의함으로써 개발자들의 오픈 소스의 사용을 활성화 한다.
- 중소 소프트웨어 사업자의 산업경쟁력 강화: 전자정부 표준 프레임워크를 공유하고 프레임워크 기술 인력을 증가시킴으로써 중소 소프트웨어 사업자의 경쟁력을 강화한다.

3. 개발 환경 및 개발 시 고려사항

3-1. 개발에 필요한 프로그램 다운로드

1 JDK 1.6 다운로드

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>



Java Platform (JDK) JDK + JavaFX Bundle JDK + NetBeans Bundle JDK + Java EE Bundle

JDK JRE

Here are the Java SE downloads in detail.

Java Platform, Standard Edition		
Java SE 6 Update 23 This release includes performance improvements and bug fixes. Learn more ▶	Download JDK	Download JRE
What Java Do I Need? You must have a copy of the JRE (Java Runtime Environment) on your system to run Java applications and applets. To develop Java applications and applets, you need the JDK (Java Development Kit), which includes the JRE.	JDK 6 Docs <ul style="list-style-type: none">Installation InstructionsReadMeReleaseNotesOracle LicenseThird Party LicensesSupported System Configurations	JRE 6 Docs <ul style="list-style-type: none">Installation InstructionsReadMeReleaseNotesOracle LicenseThird Party LicensesSupported System Configurations

2 Tomcat 6.0 다운로드

<http://tomcat.apache.org> 에서 Core Zip파일을 다운받는다.

6.0.30

Please see the [README](#) file for packaging information. It explains what every distribution contains.

Binary Distributions

- Core:
 - [zip \(pgp, md5\)](#)
 - [tar.gz \(pgp, md5\)](#)
 - [32-bit Windows zip \(pgp, md5\)](#)
 - [64-bit Windows zip \(pgp, md5\)](#)
 - [64-bit Itanium Windows zip \(pgp, md5\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, md5\)](#)
- Deployer:
 - [zip \(pgp, md5\)](#)
 - [tar.gz \(pgp, md5\)](#)








Source Code Distributions

- [tar.gz \(pgp, md5\)](#)
- [zip \(pgp, md5\)](#)

3 eclipse 다운로드

<http://eclipse.org/downloads> 에서 Eclipse IDE for java EE Developers를 다운받는다.

Compare Packages Older Versions Eclipse Helios (3.6.1) Packages for Windows

	Eclipse IDE for Java Developers , 99 MB Downloaded 1,797,841 Times Details	 Windows 32 Bit Windows 64 Bit
	Eclipse IDE for Java EE Developers , 206 MB Downloaded 1,240,069 Times Details	 Windows 32 Bit Windows 64 Bit
	Eclipse Classic 3.6.1 , 170 MB Downloaded 520,504 Times Details Other Downloads	 Windows 32 Bit Windows 64 Bit
	Eclipse IDE for C/C++ Developers , 88 MB Downloaded 479,613 Times Details	 Windows 32 Bit Windows 64 Bit
	Eclipse for PHP Developers , 141 MB Downloaded 264,356 Times Details	 Windows 32 Bit Windows 64 Bit
	Eclipse IDE for JavaScript Web Developers , 108 MB Downloaded 97,664 Times Details	 Windows 32 Bit Windows 64 Bit

4 Spring 다운로드

<http://www.springsource.org/download> 에서 Spring Framework 3.0.5.RELEASE을 클릭한다.

Spring Downloads

The Spring projects are all available from the SpringSource [Download Center](#).

Get the latest Spring releases here

- Spring Framework **3.0.5.RELEASE** is the current production release (requires Java 1.5+)
 - [Download](#) | [Changelog](#)
- Spring Framework **2.5.6.SEC02** is the latest Spring 2.5.x release (compatible with Java 1.4+)
 - [Download](#) | [Changelog](#)
- Spring Framework **2.0.8** is the latest Spring 2.0.x release (compatible with Java 1.3+)
 - [Download](#) | [Changelog](#) | [Announcement](#)
- Spring Framework **sample projects** are available for checkout in the following Subversion (SVN) repository:
 - [Browse spring-samples repository](#)
- Spring Framework **nightly snapshots** are available for testing and development purposes
 - [Download](#)

아래에서 처럼 2개의 파일을 다운받는다.

Spring Community Downloads

- Spring Framework
 - Latest GA release: 3.0.5.RELEASE
 - spring-framework-3.0.5.RELEASE-with-docs.zip (sha1) 46.8 MB
 - spring-framework-3.0.5.RELEASE.zip (sha1) 24.8 MB
 - [More >>](#)

5 Spring IDE 다운로드

- <http://www.springframework.org/download> 에서 Download Center 클릭한다.

Spring Downloads

The Spring projects are all available from the SpringSource [Download Center](#).

Get the latest Spring releases here

- Spring Framework 3.0.5.RELEASE is the current production release (requires Java 1.5+)
 - [Download](#) | [Changelog](#)
- Spring Framework 2.5.6.SEC02 is the latest Spring 2.5.x release (compatible with Java 1.4+)
 - [Download](#) | [Changelog](#)
- Spring Framework 2.0.8 is the latest Spring 2.0.x release (compatible with Java 1.3+)
 - [Download](#) | [Changelog](#) | [Announcement](#)
- Spring Framework **sample projects** are available for checkout in the following Subversion (SVN) repository:
 - [Browse spring-samples repository](#)
- Spring Framework **nightly snapshots** are available for testing and development purposes
 - [Download](#)

■ <http://www.springsource.com/products/springsource-download-center>으로 이동을 하면 화면아래오 이동한후 Open Source Download Center을 클릭한다.

RabbitMQ™ DOWNLOAD

Messaging that Just Works
RabbitMQ is a robust and reliable open source messaging system for applications that is easy to use, optimized for cloud deployment, and supported on all major operating systems and developer platforms. [Download RabbitMQ](#)

SpringSource Community DOWNLOAD

Over 2,000,000 Java Developers Use Spring
The community downloads of Spring projects are designed for individual users looking to maintain and manage the Spring libraries themselves. [Open Source Download Center](#)

DOWNLOADS

- tc Server Trial
- Hyperic Trial
- SpringSource Tool Suite (FREE)
- SpringSource ERS Trial
- Buy Now

SPRING PROJECTS

- Spring Framework
- Spring Grails
- Spring Roo
- Spring Integration
- Spring Batch
- Spring Web

WEBINARS AND DEMOS

- Best Place to Build and Run Spring Apps
- Manage and Monitor Custom Apps
- Roo: Extreme Productivity in 10 Mins
- tc Server Product Demo
- Hyperic Product Demo

RESOURCES

- Whitepaper: Spring - Manager's Overview
- Case study: NPC Intl
- Case study: Associated Newspapers
- Case study: Univ. of Illinois
- Case study: Context

■ <http://www.springsource.com/download/community>으로 이동하면 spring-ide_2.3.0.200912170948-RELEASE.zip 파일을 다운받는다.

Spring Community Downloads

- Spring Framework
- Spring Data
- Spring Security
- Spring Web Flow
- Spring Web Services
- Spring Dynamic Modules
- Spring Integration
- Spring Batch
- Spring Batch Admin
- Spring.NET
- Spring AMQP
- Spring AMQP.NET
- Spring GemFire
- Spring GemFire for .NET
- Spring LDAP
- Spring Social
- Spring Android
- Spring IDE
 - Latest GA release: 2.3.0.200912170948-RELEASE
 - spring-ide_2.3.0.200912170948-RELEASE.zip (sha1) 27.8 MB
 - spring-ide_updatesite_2.3.0.200912170948-RELEASE.zip (sha1) 34.9 MB
 - More >>
- Spring BlazeDS Integration
- SpringSource Bundlor
- Spring Roo

3-2. 개발환경 설정

1 3.1에서 다운로드 받은 파일들을 모두 압축해제한다.

2 JDK1.6 설치

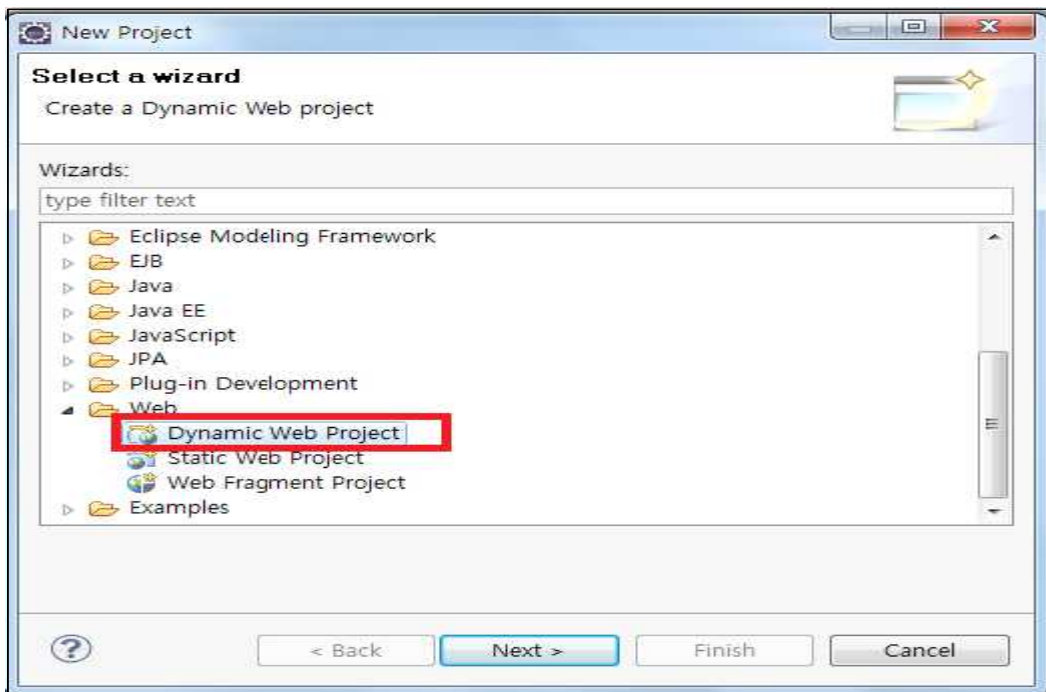
- 기본설치 다음 내컴퓨터->속성->고급->환경변수-> 시스템변수->path에
C:\Program Files\Java\jdk1.6.0_20\bin 추가한다.

3 이클립스에 Spring IDE 업데이트

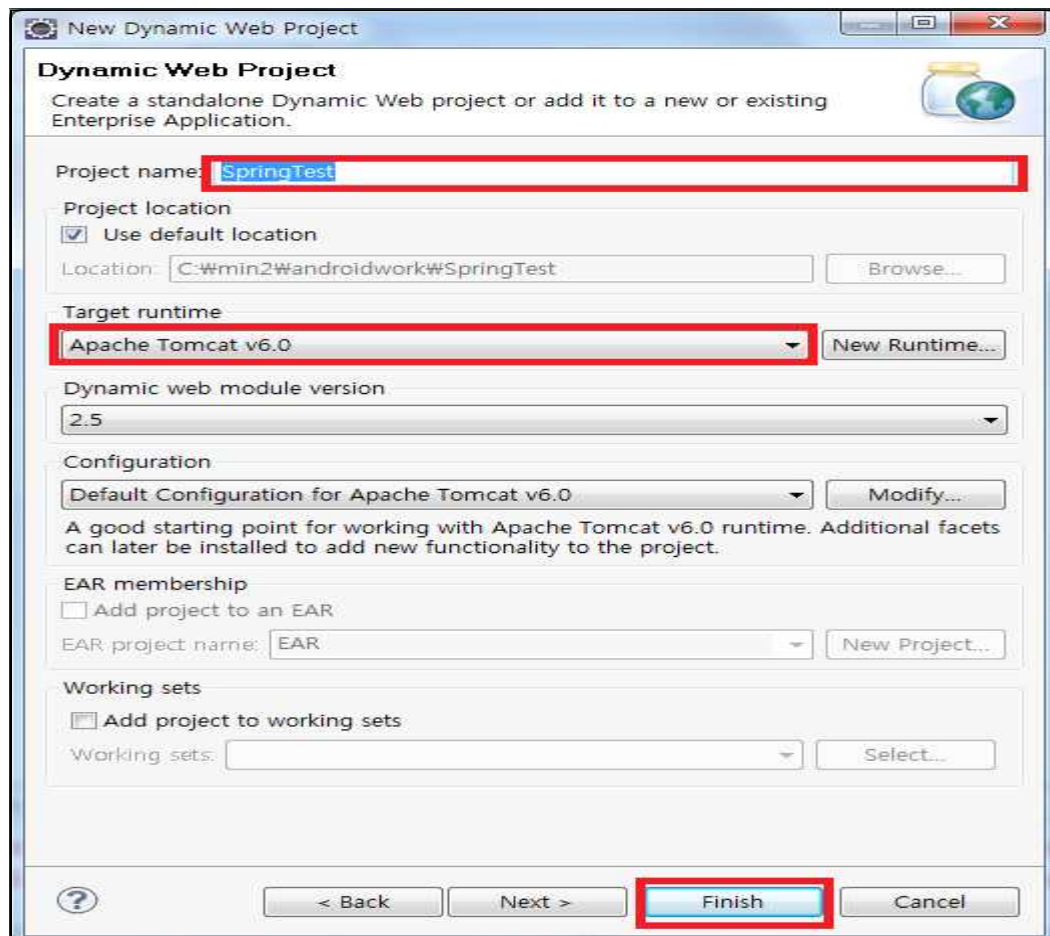
- Spring IDE에서 있는 features, plugins 두개의 폴더를 복사한 다음 이클립스 압축폴더에 붙여넣는다.

4 이클립스 실행

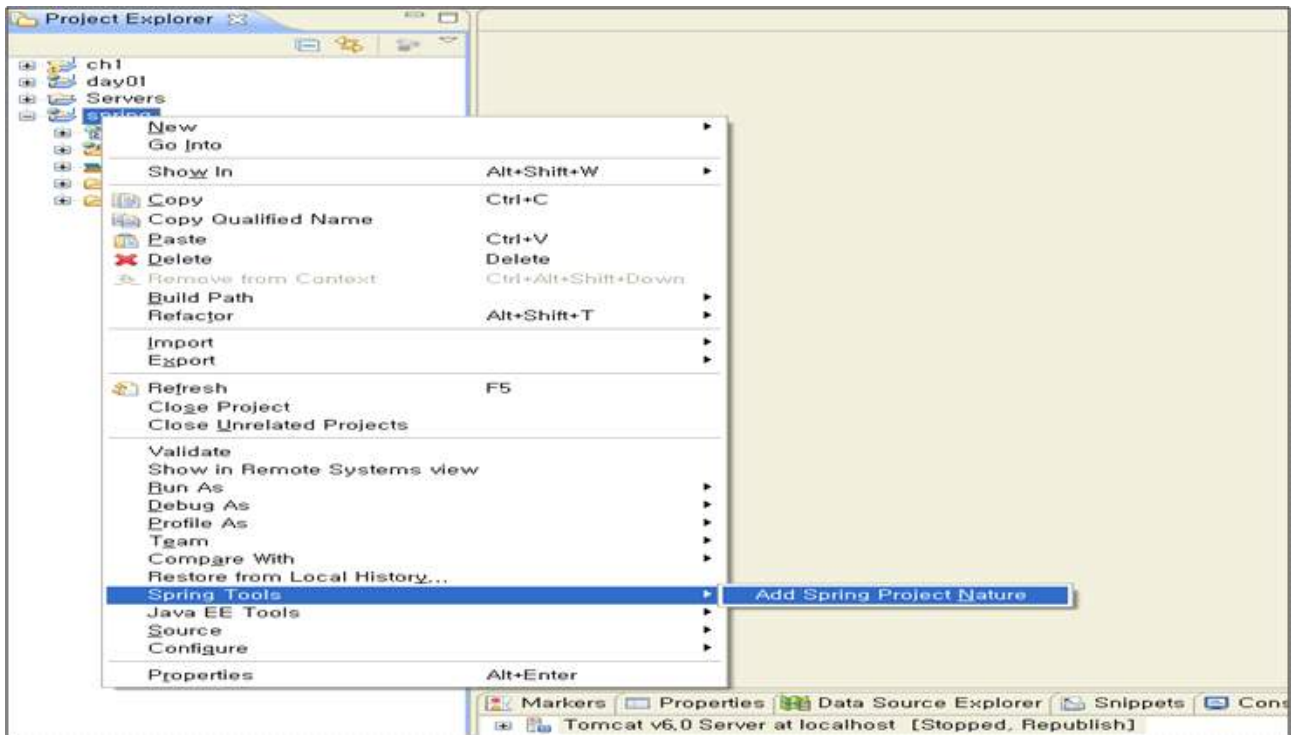
- 이클립스를 실행한 다음 메뉴 File->New->Project을 선택한다.
- 아래와 같은 화면이 나오면 Web->Dynamic Web Project을 선택한후 다음을 클릭한다.



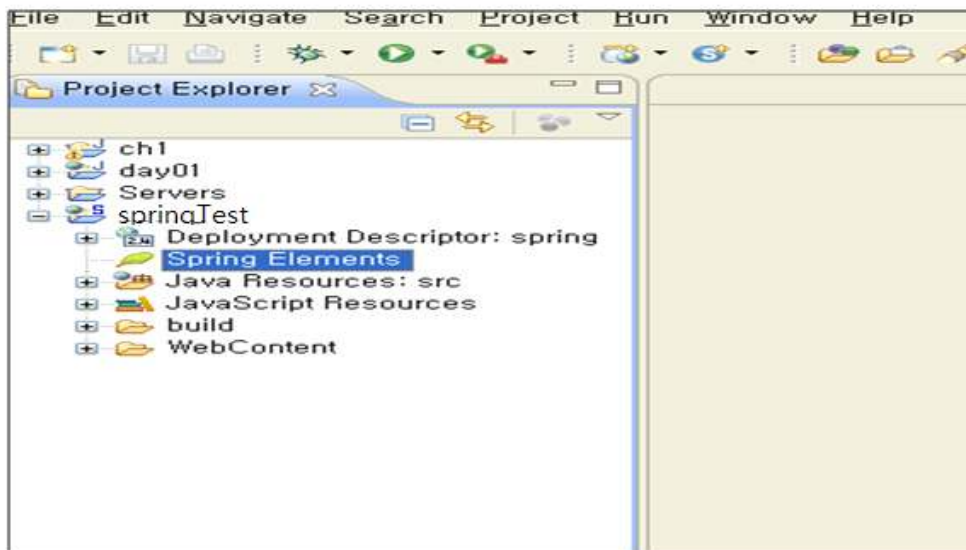
■ 아래와 같이 Project name, Target runtime 설정한 다음 finish을 클릭한다.



- springTest프로젝트 단축메뉴에서 Spring Tools -> Add Spring Project Nature을 클릭한다.



- 아래와 같이 Spring Elements가 나오면 스프링 작업환경이 설정되었다.



- 각 개발단계마다 필요한 스프링 모듈은 추후 추가하도록 한다.

4. 스프링 프레임워크 기술 학습

4-1. Spring DI(Dependency Injection)

1 spring_di관련 모듈

spring-beans-3.2.2.RELEASE.jar

spring-context-3.2.2.RELEASE.jar

spring-core-3.2.2.RELEASE.jar

spring-expression-3.2.2.RELEASE.jar

com.springsource.org.apache.commons.logging-1.1.1.jar

2 spring DI(Dependency Injection)란?

■ 의존성(Dependency)

■ 비즈니스 로직을 수행하기 위해서는 둘 이상의 클래스가 사용되는 데, 각 객체는 협업할 객체의 참조를 취득해야할 책임이 있는데, 이것이 의존성이다.

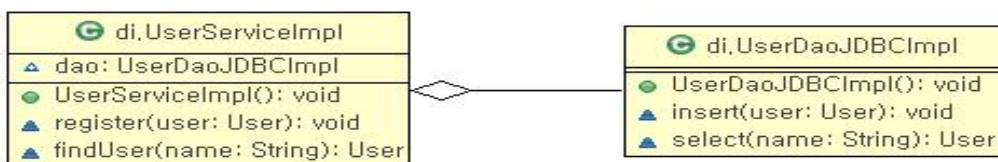
■ 객체간의 결합도가 높으면 테스트하기 어려운 코드가 만들어진다.

■ 의존성 주입 : 객체들은 객체의 생성 시점에 spring container로부터 의존성을 부여 받게 된다. 즉, 의존하는 객체를 주입받게 된다.

■ 클래스와 클래스간의 의존성

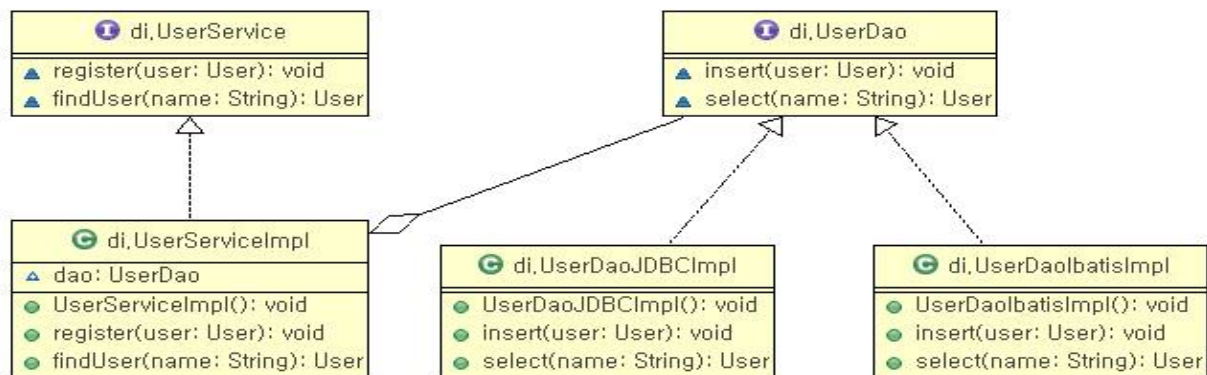
■ 클래스가 구현 클래스에 의존하는 경우

- 클래스와 클래스간의 결합도가 높다.



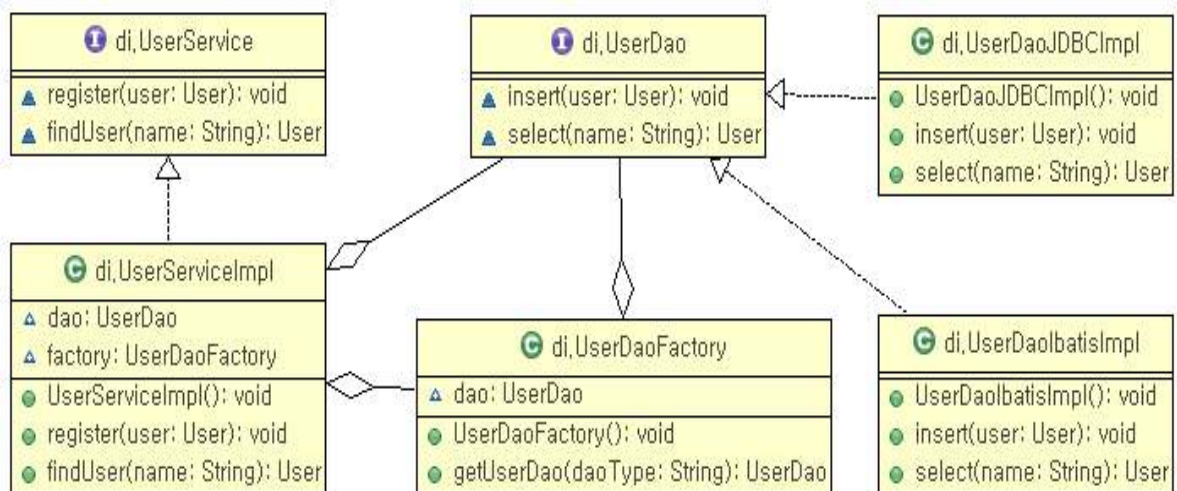
■ 클래스가 인터페이스에 의존하는 경우

- 클래스간의 결합도가 낮아진다.



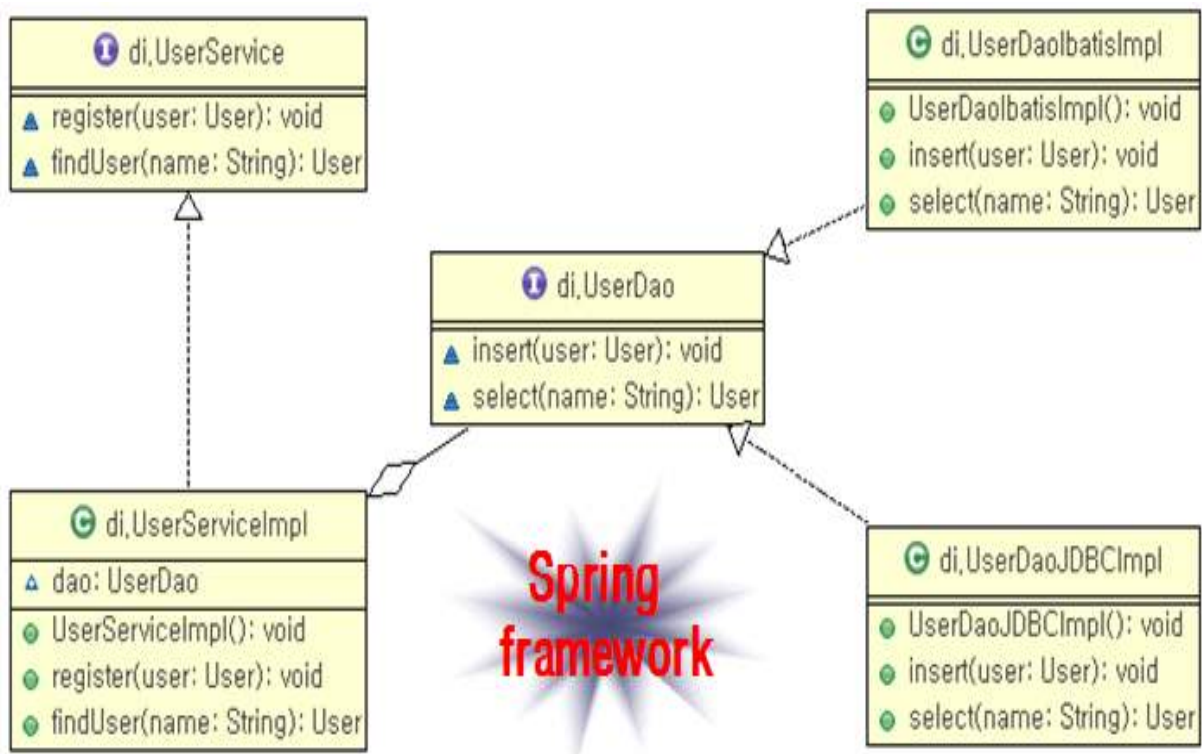
■ factory pattern을 사용한 경우

- 구현클래스를 변경하더라도 소스 코드의 변화 없음



■ spring DI 사용한 경우

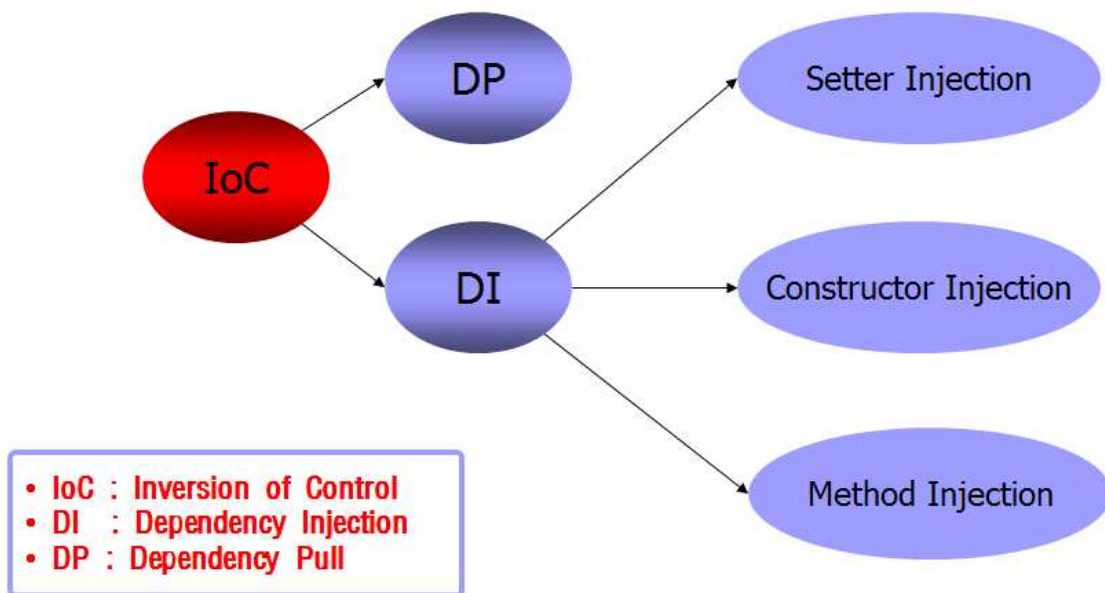
- spring container가 객체 사이의 의존관계를 조립한다.



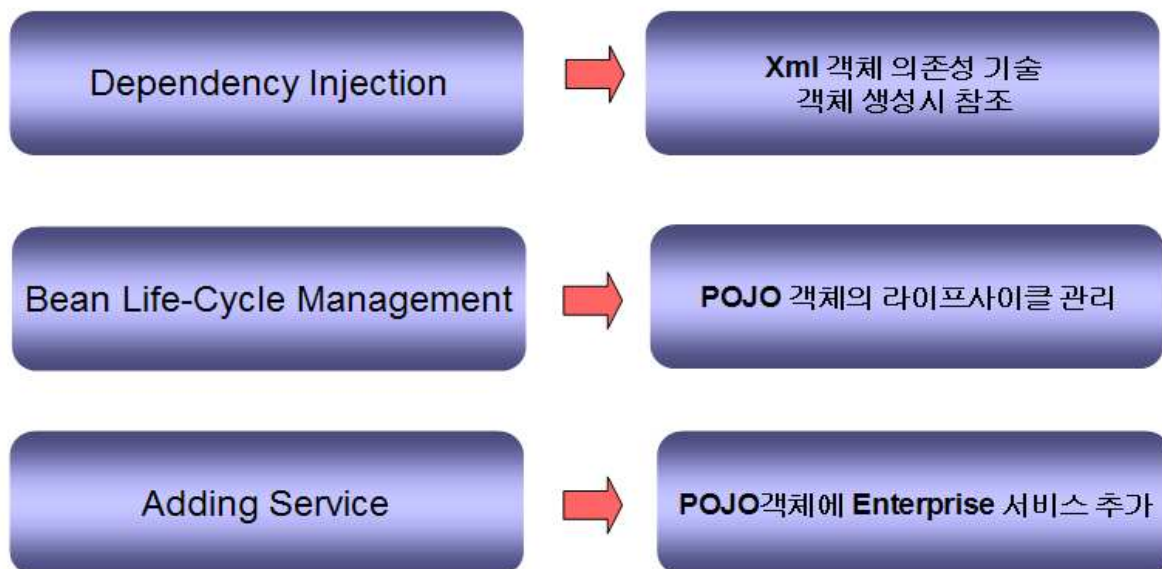
■ 제어역행(IoC : Inversion of Control)

각 객체는 협업할 객체의 참조를 취득해야하는 책임이 있는데, 제어역행은 의존하는 객체를 역행적으로 취득하는 것이다. IoC는 한 객체가 의존성을 가지는 다른 객체의 참조를 취득하는 방법에 대한 책임의 역행이라는 의미를 가지고 있다. IoC를 적용하면 객체들은 어떤 존재에 의해 객체를 생성할 때 의존성을 가지는 객체를 주입받게 된다.

※ 의존성을 가지는 객체를 참조하는 법



■ spring framework에서의 DI



■ spring 컨테이너

spring은 BeanFactory와 ApplicationContext 인터페이스를 구현한 컨테이너를 제공한다.

■ org.springframework.beans.factory.BeanFactory

Bean객체를 관리, Bean객체간의 의존관계 설정 등의 기능을 제공하는 단순한

형태의 컨테이너

- XmlBeanFactory

```
Resource resource = new FileSystemResource("beans.xml");
XmlBeanFactory factory = new XmlBeanFactory(resource);
```

■ org.springframework.context.ApplicationContext

BeanFactory의 기능 외에 자원처리 추상화, 국제화 지원, 이벤트 지원등의 추가 기능을 제공

- ClassPathXmlApplicationContext
- FileSystemXmlApplicationContext
- XmlWebApplicationContext

[예제-빈생성]

Dao.java

```
package part1;

public interface Dao {
    public void add();
    public void remove();
}
```

DaoImp.java

```
package part1;

public class DaoImp implements Dao {
    public DaoImp() {

        System.out.println("생성자");
    }

    @Override
    public void add() {

        System.out.println("add()");
    }

    @Override
    public void remove() {
```

```
        System.out.println("remove()");
    }
}
```

di.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">

<!-- 빈 선언 -->
<bean id="service" class="part1.Daolmp" />
</beans>
```

PartMain.java

```
package part1;

import org.springframework.beans.factory.xml.XmlBeanDefinitionStoreException;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import org.springframework.core.io.ClassPathResource;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;

public class PartMain {

    public static void main(String[] args) {
        //Dao dao=new Daolmp();
        // dao.add();
        //Dao dao2=new Daolmp();
        //dao2.add();

        //ApplicationContext은 xml을 읽어올때 빈이 생성
```



```

        //ApplicationContext context=
        //      new FileSystemXmlApplicationContext("src/part1/di.xml");

        //Dao service=(Dao)context.getBean("service");
        //service.add();

        //BeanFactory은 getBean()호출할때 빈 생성
        Resource resource=new FileSystemResource("src/part1/di.xml");
        XmlBeanFactory factory=new XmlBeanFactory(resource);
        Dao service=(Dao)factory.getBean("service");
    }
}

```

■ applicationContext.xml

■ 스프링에서는 컨테이너에 저장될 Bean객체와 각 Bean객체간의 연관관계를 xml 파일을 통해서 설정한다.

■ <bean> 태그는 스프링 컨테이너 안에서 생성해야 하는 자바 빈을 정의한다.

■ 빈 추가

빈을 위한 기본 설정에는 빈의 id와 클래스 전체 경로를 설정한다.

```

<?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="myBean" class="ex.bean.MyBean"/>

</beans>

```

```

<bean id="myBean" class="example.MyBean"/>

```

```
Resource resource = new FileSystemResource("applicationContext.xml");
XmlBeanFactory factory = new XmlBeanFactory(resource);
MyBean bean = (MyBean)factory.getBean("myBean");
```

■ 프로토타입과 싱글톤

- 스프링의 모든 빈은 기본적으로 싱글톤이다. 컨테이너가 배포하는 빈은 항상 동일한 인스턴스를 반환한다.
- 컨테이너로부터 매번 새로운 인스턴스를 제공받으려면 <bean>태그의 singleton속성을 false로 혹은 scope속성을 prototype로 정의한다.

```
<bean id="myBean" class="example.MyBean" singleton="false"/>
```

```
<bean id="myBean" class="example.MyBean" scope="prototype"/>
```

[예제-빈 객체의 범위 지정]

Service.java

```
package part2;

public interface Service {

    public void prn();

}
```

ServiceImp.java

```
package part2;

public class ServiceImp implements Service {

    public ServiceImp() {

        System.out.println("생성자 호출");

    }

}
```

```

@Override
    public void prn() {
        // TODO Auto-generated method stub

    }
}

```

di.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">

<!-- 빈 객체의 범위 지정 :scope속성 사용
  1 singleton : 스프링 컨테이너에 한개의 빈 객체만 존재한다(기본값)
  2 prototype : 빈을 사용할때 마다 객체를 생성한다.
  3 request : Http요청마다 빈 객체를 생성한다.
               WebApplicationContext에서만 적용된다.
  4 session : Http세션마다 빈 객체를 생성한다.
               WebApplicationContext에서만 적용된다.
-->

<bean id="svc" class="part2.ServiceImp" scope="prototype"/>
</beans>

```

ServiceTest.java

```

package part2;

import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;

```

```

public class ServiceTest {

    public static void main(String[] args) {

        Resource resource=new FileSystemResource("src/part2/di.xml");
        XmlBeanFactory factory=new XmlBeanFactory(resource);
        Service svc=(Service)factory.getBean("svc");
        svc=(Service)factory.getBean("svc");

        ApplicationContext context=
            new FileSystemXmlApplicationContext("src/part2/di.xml");
        context.getBean("svc");

        context=
            new FileSystemXmlApplicationContext("src/part2/di.xml");
        context.getBean("svc");

    }
}

```

■ 빈의 초기화와 소멸

- 스프링은 빈의 구성과해제를 위한 생명주기 메소드를 제공
- init-method는 빈이 객체화되는 즉시 호출될 메소드 정의
- destroy-method는 빈이 컨테이너로부터 제거되기 직전에

호출될 메소드를 정의

```

<bean id="myBean" class="example.MyBean"
    init-method="init" destroy-method="destroy"/>

```

```

public class MyBean{
    public void init(){
        //초기화 작업
    }
    public void destroy(){
        //해제작업
    }

    //business method
}

```

[예제-빈의 초기화와 소멸]

di2.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">
<!--
    빈 객체 초기화 및 소멸 메서드 지정
    init-method : 빈 객체가 생성된후 호출되는 메소드 지정
    destroy-method : 빈 객체가 소멸되기전 호출되는 메소드 지정
-->

<bean id="life" class="part2.ServiceImp"
       init-method="start" destroy-method="end" scope="prototype"/>
</beans>

```

ServiceTest2.java

```
package part2;

import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;
import org.springframework.core.io.FileSystemResource;
import org.springframework.core.io.Resource;

public class ServiceTest2{
    public static void main(String[] args) {
        Resource resource=new FileSystemResource("src/part2/di2.xml");
        XmlBeanFactory factory=new XmlBeanFactory(resource);
        Service life=(Service)factory.getBean("life");

        //빈을 강제적으로 소멸시킬때 호출하는 메소드
        factory.destroyBean("life",life);
    }
}
```

■ 빈 묶기(Bean Wiring)

스프링 컨테이너에서 각 객체간의 의존관계는 xml 파일을 통해서 명시할 수 있다. 객

```
public class ServiceImp{
    private BoardDao dao;
    public ServiceImp(BoardDao dao){
        this.dao = dao;
    }

    public void write(BoardVO vo){
        dao.insert(vo);
    }
}
```

체간의 의존 관계는 생성자와 setter 메소드 방식의 두 가지 방식으로 지정할 수 있다.

- 생성자 방식 : 의존하는 객체를 생성자를 통해서 전달받는다.

```
<bean id="boardDao" class="example.dao.BoardDaoImp"/>
```

```
<bean id="service" class="example.service.ServiceImp">
    <constructor-arg>
        <ref bean="boardDao"/>
    </constructor-arg>
</bean>
```

[예제-생성자 의존관계설정]

Service.java

```
package part3;

public interface Service {
    public void prn();
    public void call();
    public void moniter();
}
```

ServiceImp.java

```
package part3;

public class ServiceImp implements Service {

    private int num;
    private String name;
    private String str;

    public ServiceImp(){}

    public ServiceImp(int num) {
        this.num=num;
    }

    public ServiceImp(int num, String name){
        this.num=num;
        this.name=name;
    }
}
```



```

    public ServiceImp(String str){
        this.str=str;
    }

    @Override
    public void prn() {
        System.out.println("num="+num);
        System.out.println("name="+name);
    }

    @Override
    public void call() {
        System.out.println("str:"+str);
    }

    @Override
    public void moniter() {
        // TODO Auto-generated method stub
    }
}

```

di.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="str" class="java.lang.String">
        <constructor-arg value="spring"/>
    </bean>

    <!-- 생성자를 이용한 의존성관계설정 -생성자에 기본자료형의 값을 넘길때-->

    <!-- ServiceImp svc=new ServiceImp(10); -->
    <bean id="svc" class="part3.ServiceImp">
    <!--

```

```

    <constructor-arg>
      <value>10</value>
    </constructor-arg>
    <constructor-arg>
      <value>홍길동</value>
    </constructor-arg>
  -->

  <!--
  <constructor-arg value="10"/>
  <constructor-arg value="홍길동"/>
  -->
  <!-- 생성자를 이용한 의존성관계설정- 생성자를 호출할때 객체를 넘길때 -->
  <!--
  <constructor-arg>
    <ref bean="str"/>
  </constructor-arg>
  -->
  <constructor-arg ref="str"/>
</bean>

<bean id="svc2" class="part3.ServiceImp"/>

</beans>

```

ServiceTest.java

```

package part3;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class ServiceTest {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        ApplicationContext context=
            new FileSystemXmlApplicationContext("src/part3/di.xml");
        Service svc=(Service)context.getBean("svc");
        //svc.prn();
        svc.call();
    }
}

```

```

    }

}

```

- setter 메소드 방식: 의존하는 객체를 setter메소드를 통해서 전달 받는다.

```

public class ServiceImp{
    private BoardDao dao;
    public void setDao(BoardDao dao){
        this.dao = dao;
    }

    public void write(BoardVO vo){
        dao.insert(vo);
    }
}

```

```

<bean id="boardDao" class="example.dao.BoardDaoImp"/>

<bean id="service" class="example.service.ServiceImp">
    <property name="dao"><ref name="boardDao"/></property>
</bean>

```

[예제- setter메소드 의존관계설정]

Service.java

```

package part4;

public interface Service {
    public void prn();
    public void call();
    public void monitor();
}

```

```
}
```

ServiceImp.java

```
package part4;
```

```
public class ServiceImp implements Service {
```

```
    private int num;  
    private Integer data;  
    private String name;
```

```
    public void setName(String name){  
        this.name=name;  
    }
```

```
    public void setData(Integer data){  
        this.data=data;  
    }
```

```
    public void setNum(int num){  
        this.num=num;  
    }
```

```
    @Override  
    public void prn() {  
        System.out.println("num="+num);  
    }
```

```
    @Override  
    public void call() {  
        System.out.println("data="+data);  
    }
```

```
    @Override  
    public void monitor() {
```

```
        System.out.println("name="+name);
    }
}
```

di.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">

    <!-- 프로퍼티 설정방식을 이용한 의존성관계설정
        : setter메소드를 이용해서 값이나 객체 전달 -->

    <!--
        ServiceImp svc=new ServiceImp();
        svc.setNum(20);
    -->
    <bean id="svc" class="part4.ServiceImp">
    <!--
        <property name="num">
            <value>20</value>
        </property>
    -->
        <property name="num" value="20"/>
    </bean>

    <bean id="svc2" class="part4.ServiceImp">
        <property name="data" >
            <value type="java.lang.Integer">30</value>
        </property>
    </bean>

    <bean id="str" class="java.lang.String">
        <constructor-arg value="test"/>
    </bean>
```

```

<bean id="svc3" class="part4.ServiceImp">
  <property name="name" >
    <ref bean="str"/>
  </property>

  <!--
    <property name="name" ref="str" />
  -->
</bean>

```

```

</beans>

```

ServiceTest.java

```

package part4;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ServiceTest {

    public static void main(String[] args) {
        //ClassPathXmlApplicationContext 을 이용해서 xml자원을 읽어올때는
        //class다음부터 위치를 지정한다.
        ApplicationContext context=
            new ClassPathXmlApplicationContext("part4/di.xml");

        Service svc=(Service)context.getBean("svc");
        svc.prn();

        Service svc2=(Service)context.getBean("svc2");
        svc2.call();

        Service svc3=(Service)context.getBean("svc3");
        svc3.monitor();
    }
}

```

```

    }
}

```

[예제- XML네임스페이스를 setter메소드 의존관계설정]

di2.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!--
xml네임스페이스를 설정하기 전에 beans시작요소에 내용이 먼저 추가되어 있어야함
xmlns:p="http://www.springframework.org/schema/p"
-->
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:p="http://www.springframework.org/schema/p"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- XML네임스페이스를 이용한 프로퍼티 설정
p:프로퍼티명="값"
p:프로퍼티명-ref="객체"
-->
<bean id="imp" class="part3.ServiceImp" p:num="10"/>
</beans>

```

ServiceTest2.java

```

package part4;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ServiceTest2 {

    ApplicationContext context2=
        new ClassPathXmlApplicationContext("part4/di2.xml");

    Service imp=(Service)context2.getBean("imp");
    imp.prn();
}

```



```
}  
}
```

※ 컬렉션타입에 의존관계설정 : 프로퍼티타입이 java.util.List나 java.util.Map과 같은 컬렉션타입인 경우,스프링은 각 타입에 맞는 태그를 이용하여 값을 할당할 수 있다.

[예제-컬렉션 타입 프로퍼티 설정]

Service.java

```
package part5;  
  
public interface Service {  
  
    public void prn();  
    public void prn2();  
    public void prn3();  
    public void prn4();  
    public void prn5();  
}
```

ServiceImp.java

```
package part5;  
  
import java.util.Iterator;  
import java.util.List;  
import java.util.Map;  
import java.util.Properties;  
import java.util.Set;  
  
public class ServiceImp implements Service {  
    List<Integer> list;  
    List<String> data;  
    Map<String,String> mp;  
    Properties prop;  
    Set<Integer> st;
```

```
public ServiceImp(Set<Integer> st) {
    this.st=st;
}

public ServiceImp(Properties prop) {
    this.prop=prop;
}

public ServiceImp(Map<String,String> mp) {
    this.mp=mp;
}

    public ServiceImp(List<Integer> list) {
        System.out.println("생성자");
        this.list=list;
    }

    public void setSt(Set<Integer> st){
        this.st=st;
    }

    public void setProp(Properties prop){
        this.prop=prop;
    }

    public void setMp(Map<String,String> mp){
        this.mp=mp;
    }

    public void setData(List<String> data) {
        System.out.println("data 메소드");
        this.data = data;
    }

    @Override
    public void prn() {
        for(Integer it : list)
            System.out.println(it.intValue());
    }
```

```

@Override
public void prn2() {
    for(String str : data)
        System.out.println(str.toString());

}

@Override
public void prn3() {
    System.out.println(mp.get("지역"));
}

@Override
public void prn4() {
    System.out.println(prop.getProperty("ip"));
}

@Override
    public void prn5() {
        Iterator<Integer> it=st.iterator();
        while(it.hasNext()){
            System.out.println(it.next());
        }
    }
}
}

```

di.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">

<!-- 컬렉션 타입에 값이나 객체 전달 -->
<bean id="svc" class="part5.ServiceImp">

```

```

<constructor-arg>
  <list>
    <value >10</value>
    <value>20</value>
  </list>
</constructor-arg>

<property name="data">
  <list>
    <value>홍길동</value>
    <value>김동수</value>
  </list>
</property>
</bean>

<bean id="svc2" class="part5.ServiceImp">
<!-- Map mp=new Hashtable();
      mp.put("지역","서울"); -->
  <constructor-arg>
    <map>
      <entry>
        <key> <value>지역</value></key>
        <value>서울</value>
      </entry>
    </map>
  </constructor-arg>

  <property name="mp">
    <map>
      <entry>
        <key> <value>지역</value></key>
        <value>서울</value>
      </entry>
    </map>
  </property>

  <!--
  <property name="mp">
    <map>
      <entry key="지역" value="서울"/>

```

```
        </map>
    </property>
    -->
</bean>

<bean id="svc3" class="part5.ServiceImp">
    <constructor-arg>
        <props>
            <prop key="ip"> 127.0.0.1</prop>
            <prop key="domain">localhost</prop>
        </props>
    </constructor-arg>

    <property name="prop">
        <props>
            <prop key="ip"> 127.0.0.1</prop>
            <prop key="domain">localhost</prop>
        </props>
    </property>
</bean>

<bean id="svc4" class="part5.ServiceImp">
    <constructor-arg>
        <set>
            <value>100</value>
            <value>200</value>
        </set>
    </constructor-arg>

    <property name="st">
        <set>
            <value>100</value>
            <value>200</value>
        </set>
    </property>
</bean>

</beans>
```

ServiceTest.java

```
package part5;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ServiceTest {

    public static void main(String[] args) {
        ApplicationContext context=
            new ClassPathXmlApplicationContext("part5/di.xml");
        Service svc=(Service)context.getBean("svc");
        //svc.prn();
        //svc.prn2();

        Service svc2=(Service)context.getBean("svc2");
        //svc2.prn3();

        Service svc3=(Service)context.getBean("svc3");
        //svc3.prn4();

        Service svc4=(Service)context.getBean("svc4");
        svc4.prn5();
    }
}
```

4-2. Spring AOP(Aspect Oriented Programming)

1 AOP관련모듈추가

com.springsource.org.aopalliance-1.0.0.jar

com.springsource.org.aspectj.tools-1.6.6.RELEASE.jar

spring-aop-3.2.2.RELEASE.jar

spring-aspects-3.2.2.RELEASE.jar

2 AOP(Aspect Oriented Programming)

■ 관점 지향 프로그래밍은 핵심로직을 구현하는 코드와 공통 기능 관련 로직(로그, 트랜잭션 관리, 보안 등)의 분리가 목적이다.

■ 다수의 컴포넌트에 공통 기능 관련 로직이 산재할 때의 이슈

- 시스템 전반적인 관심사를 구현한 코드가 여러 컴포넌트에 중복
- 변경 이슈가 발생할 경우 개별적인 코드를 수정
- 컴포넌트의 핵심기능이 기능적인 코드에 의해 오염됨

■ AOP는 어떤 클래스의 어떤 메소드가 실행되기 전·후에 다른 기능을 수행할 수 있는 코드를 캡슐화하는 것이다.(즉, 비기능적 요구사항이 핵심 애플리케이션 코드에 나타나지 않도록 캡슐화 하는 것)

3 AOP 용어

- Advice
 - What(무엇을)과 When(언제)이 결합된 것
 - 공통 관심 사항을 핵심 로직에 언제 적용할 것인가를 정의
 - 구현하고자 하는 공통 관심 사항의 기능
- Joinpoint
 - Where(어디에)의 의미
 - 공통 관심 사항을 적용할 수 있는 애플리케이션의 실행지점
 - Advice를 적용하는 지점
- Pointcut
 - Advice가 실제로 적용되는 Joinpoint

Advice, Joinpoint, Pointcut, Target



4 POJO 기반의 AOP 설정

- applicationContext.xml에 namespace 추가

applicationContext.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"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:p="http://www.springframework.org/schema/p"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
                           http://www.springframework.org/schema/beans/spring-beans.xsd
                           http://www.springframework.org/schema/aop
                           http://www.springframework.org/schema/aop/spring-aop.xsd">

</beans>
```

- 공통 관심 사항 구현(Logging)

LoggingAspect.java

```
package aop.pojo;

import org.aspectj.lang.JoinPoint;

public class LoggingAspect {

    public void logging(JoinPoint joinPoint){

        String name = joinPoint.getSignature().getName();
        System.out.println("::: signatureName " + name);

    }
}
```

- applicationContext.xml에 aop 설정추가

LoggingAspect.java

```
<bean id="log" class="aop.pojo.LoggingAspect"/>

<aop:config>
    <aop:aspect id="loggingAspect" ref="log">
        <aop:pointcut id="publicMethod" expression="execution(public**.*Imp.*(..))"/>
        <aop:before method="logging" pointcut-ref="publicMethod"/>
    </aop:aspect>
</aop:config>

<bean id="departmentService" class="aop.service.DepartmentServiceImp"/>
```

■ AOP 설정 태그

- <aop:aspect>
 - Aspect를 설정함
 - id : 유일한 aspect의 id, ref : aspect를 구현한 Bean 참조
 - <aop:pointcut>와 advice를 표현하는 태그를 포함하고 있다.
- <aop:pointcut>
 - advice를 적용할 pointcut을 정의한다.
 - id : pointcut를 구분하는 id값, expression : AspectJ 표현식
- Advice Tag

태그	설명
<aop:before>	메소드 실행전에 적용되는 advice를 정의한다.
<aop:after-returning>	메소드가 정상적으로 실행된 후에 적용되는 advice를 정의한다.
<aop:after-throwing>	메소드가 예외를 발생시킬 때 적용되는 advice를 정의한다. try-catch블록에서 catch 블록과 비슷하다.
<aop:after>	메소드가 정상적으로 실행되었는지 혹은 예외를 발생시키는지 여부에 상관없이 적용되는 advice를 정의한다. try-catch-finally에서 finally블록과 비슷하다.
<aop:arround>	메소드 호출 이전, 이후, 예외발생 등 모든 시점에 적용 가능한 advice를 정의한다.

■ Advice Tag의 속성값

- method : Advice 로직을 정의한 메소드 이름
- arg-names : Advice 메소드에 전달할 파라미터 이름
- pointcut-ref : 관련된 pointcut 이름
- returning : <aop:after-returning>에만 해당, 비즈니스 로직의 리턴값
- throwing : <aop:after-throwing>에만 해당, 메소드에 전달할 Exception 이

름

■ JoinPoint 클래스

- org.aspectj.lang.JoinPoint 임
- 대상 객체 및 호출되는 메소드에 대한 정보, 전달되는 파라미터에 대한 정보가 필요한 경우 사용한다.
- JointPoint의 주요 메소드

Return	Method	Description
Object	getTarget()	대상 객체
Object[]	getArgs()	파라미터 목록
Signature	getSignature()	메소드 정보
String	toLongString()	메소드 상세 정보
String	toShortString()	메소드 간략 정보

■ after returning advice

applicationContext.xml
<pre> <aop:config> <aop:aspect id="loginAspect" ref="logging"> <aop:pointcut id="publicMethod" expression="execution(public*..*Imp.*(..))"> <aop:after-returning method="loggingAfterReturing" pointcut-ref="publicMethod"> </aop:pointcut> </aop:aspect> </aop:config> </pre>

■ after returning advice

```

public void loggingAfterReturing(){
    // Aspect Logic
    ....
}

```

applicationContext.xml

```

<aop:config>
    <aop:aspect id="loginAspect" ref="logging">
        <aop:pointcut id="publicMethod" expression="execution(public*..*Imp.*(..))">
            <aop:after-returning method="loggingAfterReturing" pointcut-ref="publicMethod"
                returning="returnValue" />
        </aop:pointcut>
    </aop:aspect>
</aop:config>

```

```

public void loggingAfterReturing(Object returnValue){
    // Aspect Logic
    ....
}

```

```

public void loggingAfterReturing(JoinPoint joinPoint, Object returnValue){
    // Aspect Logic
    ....
}

```

■ after throwing advice

applicationContext.xml

```

<aop:config>
    <aop:aspect id="loginAspect" ref="logging">
        <aop:pointcut id="publicMethod" expression="execution(public*..*Imp.*(..))">
            <aop:after-throwing method="loggingException" pointcut-ref="publicMethod">
        </aop:pointcut>
    </aop:aspect>
</aop:config>

```

■ around advice

applicationContext.xml

```

public void loggingException(){
<aop:config>
    // Aspect Logic
    <aop:aspect id="loginAspect" ref="logging">
        .... <aop:pointcut id="publicMethod" expression="execution(public*..*Imp.*(..))">
    }
        <aop:after-returning method="loggingException" pointcut-ref="publicMethod">
    </aop:aspect>
</aop:config>

```

```

public UserObject profile(JoinPoint joinPoint)throws Throwable{
    // Aspect Logic
    ....
}

```

[예제-xml스키마를 이용한 AOP설정]

Service.java

```

package part1;

public interface Service {
    public void prn();
    public void first();
    public void second();
    public String third();
    public void fourth();
    public void fifth() ;
}

```

ServiceImp.javav-핵심관심사항

```

package part1;

public class ServiceImp implements Service {

    @Override
    public void prn() {

```

```

    }

    @Override
    public void first() {
        System.out.println("first()");
    }

    @Override
    public void second() {
        System.out.println("second()");
    }

    @Override
    public String third() {
        return "java";
    }

    @Override
    public void fourth() {
        System.out.println("fourth()");
    }

    @Override
    public void fifth() {
        int arr[]=new int[3];
        arr[4]=10;
    }
}

```

AopCommon.java -공통관심사항

```

package part1;

import org.aspectj.lang.ProceedingJoinPoint;

public class AopCommon {
    public void commMethod1(){

```

```

        System.out.println("선행 공통메소드()");
    }

    public void commMethod2(){
        System.out.println("후행 공통메소드()");
    }

    public void commMethod3(Object ret){
        System.out.println("반환값을 받은 후행 공통메소드():"+ret);
    }

    public void commMethod4(ProceedingJoinPoint joinPoint){
        System.out.println("target 실행전");
        try{
            joinPoint.proceed();
        }catch(Throwable e){e.getMessage();}
        System.out.println("target 실행후");
    }

    public void commMethod5(Exception thw){
        System.out.println(thw.getMessage());
    }
}

```

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

```

```

<!-- Aspect 빈 객체 선언 -->
<bean id="commAspect" class="part1.AopCommon" />

<!-- Target 빈 객체 선언 -->
<bean id="svc" class="part1.ServiceImp" />

<!-- Advice 정의 -->
<!-- AOP설정 : Target 빈에 Aspect을 빈 삽입 -->
<aop:config>
    <aop:pointcut expression="execution(* part1.ServiceImp.first(..))" id="aa"/>
    <aop:pointcut expression="execution(* part1.ServiceImp.second(..))" id="bb"/>
    <aop:pointcut expression="execution(* part1.ServiceImp.third(..))" id="cc"/>
    <aop:pointcut expression="execution(* part1.ServiceImp.fourth(..))" id="dd"/>
    <aop:pointcut expression="execution(* part1.ServiceImp.fifth(..))" id="ee"/>
    <aop:aspect ref="commAspect">
        <aop:before method="commMethod1" pointcut-ref="aa"/>
        <aop:after method="commMethod2" pointcut-ref="bb"/>
        <aop:after-returning method="commMethod3" pointcut-ref="cc" returning="ret"/>
        <aop:around method="commMethod4" pointcut-ref="dd" />
        <aop:after-throwing method="commMethod5" pointcut-ref="ee" throwing="thw"/>
    </aop:aspect>
</aop:config>
</beans>

```

ServiceTest.java

```

package part1;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.FileSystemXmlApplicationContext;

public class ServiceTest {
    public static void main(String[] args) {
        ApplicationContext context=
            new FileSystemXmlApplicationContext("src/part1/aop.xml");
        Service svc=(Service)context.getBean("svc");
        //svc.first();
        //svc.second();
        //svc.third();
    }
}

```

```
//svc.fourth();
    svc.fifth();
    }
}
```

[예제]-어노테이션을 이용한 AOP설정

AopCommon.java

```
package part2;
```

```
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.AfterThrowing;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
```

```
@Aspect
```

```
public class AopCommon {
```

```
    @Before("execution(* part2.ServiceImp.first(..)")
    public void commMethod1(){
        System.out.println("선행 공통메소드()");
    }
```

```
    @After("execution(* part2.ServiceImp.second(..)")
    public void commMethod2(){
        System.out.println("후행 공통메소드()");
    }
```

```
    @AfterReturning(pointcut="execution(* part2.ServiceImp.third(..)",returning="ret")
    public void commMethod3(Object ret){
        System.out.println("반환값을 받은 후행 공통메소드():"+ret);
    }
```

```
    @Around("execution(* part2.ServiceImp.fourth(..)")
    public void commMethod4(ProceedingJoinPoint joinPoint){
```

```
        System.out.println("target 실행전");
        try{
```



```

        joinPoint.proceed();
    }catch(Throwable e){e.getMessage();}
    System.out.println("target 실행후");
}

@AfterThrowing(pointcut="execution(* part2.ServiceImp.fifth(..)",throwing="thw")
public void commMethod5(Exception thw){
    System.out.println(thw.getMessage());
}
}

```

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

<!-- annotation을 이용한 AOP설정을 위해 아래와 같이 명시한다. -->
<aop:aspectj-autoproxy/>

<!-- Aspect 빈 객체 선언 -->
<bean id="commAspect" class="part2.AopCommon" />

<!-- Target 빈 객체 선언 -->
<bean id="svc" class="part2.ServiceImp" />

</beans>

```

4-3. Spring Database연동

1 Spring Database관련 모듈추가

mybatis-spring-1.2.0.jar
spring-jdbc-3.2.2.RELEASE.jar
spring-orm-3.2.2.RELEASE.jar
spring-tx-3.2.2.RELEASE.jar

2 Spring과 데이터베이스 연동

■ DataSource 설정하기

스프링은 DataSource(Connection Pool)를 통해서 Connection을 제공한다.

- DriverManager를 이용한 DataSource 설정
커넥션풀이나 jndi를 사용할 수 없는 경우 사용한다.

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl" />
    <property name="username" value="scott" />
    <property name="password" value="tiger" />
</bean>
```

- Apache DBCP를 이용한 DataSource 설정
commons-dbc.jar, commons-pool.jar lib 필요

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="oracle.jdbc.OracleDriver" />
    <property name="url" value="jdbc:oracle:thin:@localhost:1521:orcl" />
    <property name="username" value="scott" />
    <property name="password" value="tiger" />
</bean>
```

- JNDI를 이용한 DataSource 설정
META-INF폴더에 context.xml 생성, WAS의 connection pool 설정 기술

context.xml

```
<?xmlversion="1.0"encoding="UTF-8"?>
<Context>
    <Resource name="jdbc/myoracle" auth="Container"
              type="javax.sql.DataSource"
```

```

        driverClassName="oracle.jdbc.OracleDriver"
        url="jdbc:oracle:thin:@localhost:1521:orcl"
        username="scott" password="tiger"
        maxActive="20" maxIdle="10" maxWait="-1">
</Context>

```

```

<bean id="dataSource"
      class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="java:comp/env/jdbc/myoracle"/>
</bean>

```

■ spring JDBC 모듈

- 템플릿 클래스를 통한 데이터베이스 접근
- DaoSupport 클래스를 이용한 Dao 클래스 구현
- 의미있는 예외 클래스 제공

■ jdbc와 spring JDBC 비교

- 스프링은 데이터베이스 연동을 위한 템플릿 클래스를 제공한다.

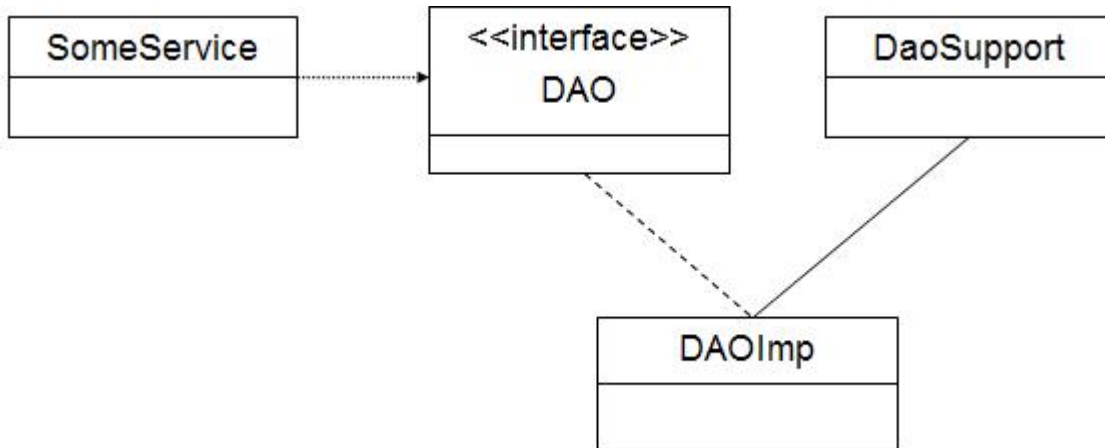
JDBC	구분	스프링
○	Connection 획득	X
○	Query	○
○	PreparedStatement	X
○	Parameter 선언	○
○	Parameter setting	X
○	Query 실행	○
○	ResultSet Handler	○
○	Transaction 처리	X
○	Connection 해제	X

- 개발자의 중복된 코드입력 작업을 줄여준다.
- JDBC, iBatis, Hibernate 등의 다양한 기술에 대해 템플릿 클래스를 제공한다.

■ DaoSupport 클래스

- Dao클래스에서 필요로 하는 기본 기능을 제공하는 클래스

- DataSource로부터 Connection획득, 템플릿 클래스 제공
- DaoSupport 클래스를 상속받아서 Dao 구현 클래스 작성



■ Spring jdbcDaoSupport 클래스

- JdbcTemplate 클래스로 제공하는 DaoSupport 클래스
 - JdbcTemplate 클래스는 CRUD작업을 위한 메소드를 제공한다.
 - JdbcDaoSupport 클래스를 상속받는 사용자 Dao클래스는 DataSource를 주입받아야 한다.
 - getJdbcTemplate() 메소드를 사용해서 JdbcTemplate 객체를 얻는다.

[예제-JJdbcDaoSupport를 이용한 DAO구현]

```

<bean id="dao" class="dao.NoticeDaoImp">
  <property name="dataSource" ref="dataSource" />
</bean>
  
```

```

public class NoticeDaoImp extends JdbcDaoSupport implements NoticeDao {

    @Override
    public void insert(Notice notice) {
        String sql = "insert into notice values (notice_seq.nextval, ?, ?, 0, sysdate)";

        Object[] params = new Object[] { notice.getTitle(), notice.getContent() };
    }
}
  
```

```

        this.getJdbcTemplate().update(sql, params);
    }

    @Override
    public Notice info(int no) {
        String sql = "select no, title, content, hit, regdate from notice where no = ?";

        return (Notice) getJdbcTemplate().queryForObject(sql, new Object[]{no},
            new RowMapper(){

                @Override
                public Object mapRow(ResultSet rs, int rownum) throws SQLException {
                    Notice notice = new Notice();

                    notice.setNo(rs.getInt("no"));
                    notice.setTitle(rs.getString("title"));
                    notice.setContent(rs.getString("content"));
                    notice.setHit(rs.getInt("hit"));
                    notice.setRegdate(rs.getDate("regdate"));

                    return notice;
                }
            });
    }

    @Override
    public List<Notice> list() {
        String sql = "select no, title, content, hit, regdate from notice";

        return getJdbcTemplate().query(sql, new RowMapper(){
            @Override
            public Object mapRow(ResultSet rs, int rownum) throws SQLException {

                Notice notice = new Notice();
                notice.setNo(rs.getInt("no"));
                notice.setTitle(rs.getString("title"));
                notice.setContent(rs.getString("content"));
                notice.setHit(rs.getInt("hit"));
                notice.setRegdate(rs.getDate("regdate"));
            }
        });
    }

```

```

        return notice;
    }
});
}

@Override
public void insert(Notice notice) {
    String sql = "insert into notice values (notice_seq.nextval, ?, ?, 0, sysdate)";

    Object[] params = new Object[]{notice.getTitle(), notice.getContent()};
    this.getJdbcTemplate().update(sql, params);
}

@Override
public void update(Notice notice) {
    String sql = "update notice set title = ?, content = ? where no = ?";

    Object[] params = new Object[]{notice.getTitle(), notice.getContent(), notice.getNo()};
    this.getJdbcTemplate().update(sql, params);
}
}

```

3 Spring과 ORM(Object-Relation Mapping) 연동

■ spring ORM 모듈은 iBatis, Hibernate, JPA와 같은 영속성 프레임워크와의 연동을 지원하는 라이브러리를 제공한다.

■ spring과 iBatis 연동

- 스프링은 SqlMapClient를 사용할 때 발생하는 중복 코드를 없애고, SQLException을 스프링에서 제공하는 UnChecked Exception로 변환해주는 SqlMapClientTemplate 클래스를 제공한다.

- 스프링은 SqlMapClientTemplate 클래스를 Dao 클래스에서 간편하게 사용할 수 있도록 하기 위해 SqlMapClientDaoSupport 클래스를 제공한다.

■ spring와 iBatis 연동 환경설정

- 스프링의 SqlMapClientFactoryBean을 이용해서 SqlMapClient 객체를 생성한다.
 - configLocation : iBatis 환경 설정 파일의 경로
 - dataSource : Connection Pool 객체를 지정한다.

```
<bean
  id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
  <property name="configuration" value="WEB-INF/SqlMapConfig.xml" />
  <property name="dataSource" ref="dataSource" />
</bean>
```

■ 스프링에서 제공하는 SqlMapClientDaoSupport을 상속받아서 Dao 구현 클래스를 작성한다.

- 구현 클래스 생성시 SqlMapClient 객체를 주입한다.
- getSqlMapClientTemplate() 메소드를 이용해서 템플릿클래스를 획득한다.

```
<bean
  id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
  <property name="configuration" value="WEB-INF/SqlMapConfig.xml" />
  <property name="dataSource" ref="dataSource" />
</bean>

<bean id="dao" class="dao.DaoImp">
  <property name="sqlMapClient" ref="sqlMapClient" />
</bean>
```

[예제- spring와 iBatis 연동]

SqlMapConfig.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTDSQLMapConfig2.0//EN"
"http://www.ibatis.com/dtd/sql-map-config-2.dtd">

<sqlMapConfig>

  <setting useStatementNamespaces="true" />

  <sqlMap resource="notice/config/notice.xml" />

</sqlMapConfig>
```

NoticDaoImp.java

```
import java.sql.SQLException;
import java.util.List;
import notice.vo.Notice;
import org.springframework.orm.ibatis.support.SqlMapClientDaoSupport;

public class NoticeDaoImpl extends SqlMapClientDaoSupport implements NoticeDao{

    public void delete(int no){
        getSqlMapClientTemplate().insert("Notice.delete", no);
    }

    public Notice info(int no){
        return (Notice)getSqlMapClientTemplate().queryForObject("Notice.info", no);
    }

    public void insert(Notice notice){
        getSqlMapClientTemplate().insert("Notice.insert", notice);
    }

    public List<Notice> list(){
        return getSqlMapClientTemplate().queryForList("Notice.list");
    }

    public void update(Notice notice){
        getSqlMapClientTemplate().insert("Notice.update", notice);
    }
}
```

4-4. Spring Transaction

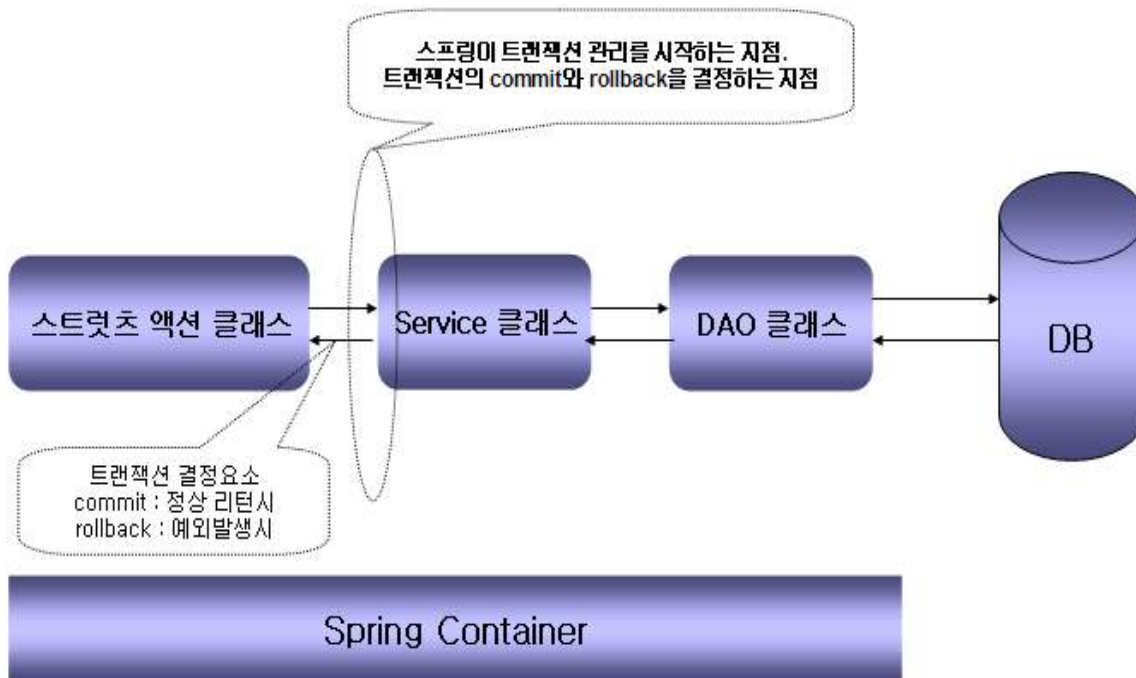
1 Spring에서의 Transaction 처리

■ 선언적 Transaction 지원

■ 스프링 트랜잭션관리 기능을 사용하면 데이터베이스를 연동하는 다양한 기술들(jdbc, iBatis, Hibernate, JPA, ...)에 대해서 일관된 방법으로 트랜잭션을 관리할 수 있다.

■ 실제 소스 코드에는 트랜잭션을 관리하기 위한 코드가 필요하지 않다.

■ 웹 어플리케이션에서의 트랜잭션 처리



■ JDBC 기반 트랜잭션 관리자 설정

■ JDBC나 iBatis와 같이 JDBC를 이용해서 데이터베이스 연동을 처리하는 경우, DataSourceTransactionManager를 트랜잭션 관리자로 사용한다.

```
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>
```

■ DataSourceTransactionManager는 dataSource프로퍼티를 사용해서 전달받은 DataSource로부터 Connection를 가져온 뒤, Connection의 commit(), rollback() 등의 메소드를 사용해서 트랜잭션을 관리한다.

■ 선언적 트랜잭션

■ 트랜잭션 처리를 코드에서 직접적으로 수행하지 않고, 설정 파일이나 어노테이션을 이용해서 트랜잭션의 범위, 롤백 규칙 등을 정의한다.

- <tx:advice> 태그를 이용한 선언적 트랜잭션 처리

```
<bean id="transactionManager"
      class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <property name="dataSource" ref="dataSource"/>
</bean>

<tx:adviceid="txAdvice" transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="get*" read-only="true"/>
    <tx:method name="add*" propagation="REQUIRED"/>
    <tx:method name="remove*" propagation="REQUIRED"/>
    <tx:method name="update*" propagation="REQUIRED"/>
  </tx:attributes>
</tx:advice>

<aop:config>
  <aop:pointcut id="serviceOperation"
    expression="execution(* com.hanbit.service.*Service.*(..))"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceOperation"/>
</aop:config>
```

- <tx:advice> 태그는 트랜잭션을 적용할 때 사용될 Advisor를 생성한다.

id : 생성될 트랜잭션 Advisor의 식별 값을 지정.

transaction-manager : TransactionManager 빈을 지정.

- <tx:method>태그는 트랜잭션을 설정할 메소드 및 트랜잭션 속성을 지정한다.

■ <tx:advice> 태그는 Advisor만을 생성한다. 트랜잭션의 적용은 AOP를 통해서 이루어진다. <aop:config> 태그를 이용해서 <tx:advice>로 설정된 트랜잭션 Advisor를 적용하도록 설정한다.

4-5. Spring MVC

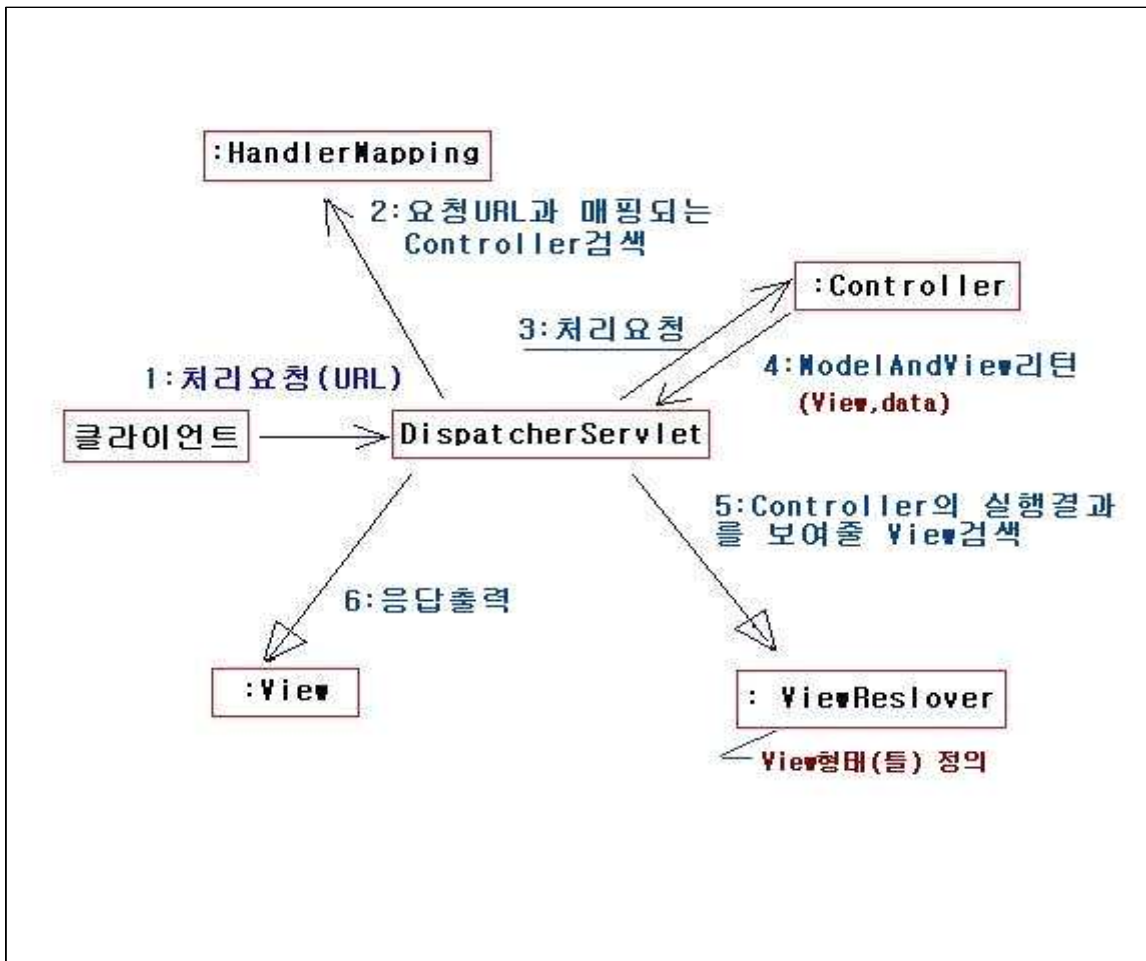
1 Spring MVC관련 모듈 추가

com.springsource.javax.servlet.jsp.jstl-1.1.2.jar

com.springsource.org.apache.taglibs.standard-1.1.2.jar
com.springsource.org.apache.commons.fileupload-1.2.0.jar
com.springsource.org.apache.commons.io-1.4.0.jar
spring-web-3.2.2.RELEASE.jar
spring-webmvc-3.2.2.RELEASE.jar

2 스프링 MVC의주요구성요소

- DispatcherServlet : 클라이언트의 요청을 전달 받는다. 컨트롤러에게 클라이언트의 요청을 전달하고, 컨트롤러가 리턴한 결과 값을 View에 전달하여 알맞은 응답을 생성하도록 한다.
- HandlerMapping : 클라이언트의 요청 URL을 어떤 컨트롤러가 처리할지를 결정한다.
- Controller : 클라이언트의 요청을 처리한 뒤, 그 결과를 DispatcherServlet에 알려준다. 스트럿츠의 Action과 동일한 역할을 수행한다.
- ModelAndView : 컨트롤러가 처리한 결과 정보 및 뷰 선택에 필요한 정보를 담는다.
- ViewResolver : 컨트롤러의 처리 결과를 생성할 뷰를 결정한다.
- View : 컨트롤러의 처리 결과 화면을 생성한다.



3 HandlerMapping

DispatcherServlet은 클라이언트의 요청이 들어오면 해당 요청을 처리할 컨트롤러를 구하기 위해 HandlerMapping을 이용한다.

- SimpleUrlHandlerMapping : 패턴과 컨트롤러 이름을 비교하여, URL이 패턴에 매칭될 경우 지정한 컨트롤러를 사용한다.
- BeanNameUrlHandlerMapping : URL과 일치하는 이름을 갖는 빈을 컨트롤러로 사용한다.
- ControllerClassNameHandlerMapping : URL과 매칭되는 클래스 이름을 갖는 빈을 컨트롤러로 사용한다.
- DefaultAnnotationHandlerMapping : @RequestMapping 어노테이션을 이용하여 요청을 처리할 컨트롤러를 구한다.

4 컨트롤러의 종류

스프링 3.0부터는 @Controller 어노테이션을 이용해서 컨트롤러 클래스를 구현하도록 권장하고 있으며 기존의 Controller 인터페이스와 클래스들은 고전클래스라 불린다. @MVC을 이용해서

구현하려면 컨트롤러 클래스를 구현하려면 @Controller 어노테이션과 @RequestMapping 어노테이션을 이용한다.

- 컨트롤러 클래스에 @Controller 어노테이션을 적용한다.
- 클라이언트의 요청을 처리할 메서드에 @RequestMapping 어노테이션을 적용한다.
- 설정 파일에 컨트롤러 클래스를 빈으로 등록한다.

5 ViewResolver

ViewResolver를 사용하여 결과를 출력할 View객체를 구하고, 구한 View객체를 이용하여 내용을 생성한다.

- InternalResourceViewResolver : 뷰 이름으로부터 JSP나 Tiles연동을 위한 View객체를 리턴한다.
- VelocityViewResolver : 뷰 이름으로부터 Velocity연동을 위한 View객체를 리턴한다.
- BeanNameViewResolver : 뷰 이름과 동일한 이름을 갖는 빈 객체를 View객체로 사용한다.
- ResourceBundleViewResolver : 뷰 이름과 View객체 간의 매핑 정보를 저장하기 위해 자원 파일을 사용한다.
- XmlViewResolver : 뷰 이름과 View객체간의 매핑 정보를 저장하기 위해 XML파일을 사용한다.

[예제-Controller클래스 메소드에서 String값반환]

web.xml
<pre><?xml version="1.0" encoding="UTF-8"?> <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" x m l n s = " h t t p : / / j a v a . s u n . c o m / x m l / n s / j a v a e e " xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5"> <display-name>spring_mvc</display-name> <!-- POST방식일때 한글처리 --> <filter> <filter-name>encodingFilter</filter-name> <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class> <init-param> <param-name>encoding</param-name></pre>

```

        <param-value>UTF-8</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

<servlet>
    <servlet-name>dispatchone</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>dispatchone</servlet-name>
    <url-pattern>*.htm</url-pattern>
</servlet-mapping>

```

WEB-INF/dispatchone.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">

    <!-- 1 컨트롤러 빈 선언 -->
    <bean id="hello" class="part2.HelloController" />
    <bean id="helloCommand" class="part2.HelloCommandController"/>

    <!-- 2 ViewResolver 빈 선언 -->
    <bean id="viewResolver"
          class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>

```

HelloVo.java

```

package part2;

public class HelloVO {
    private String id;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }
}

```

HelloController.java

```

package part2;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

//http://localhost:8090/spring_mvc/hello.htm

@Controller
public class HelloController {

    @RequestMapping("/hello.htm")
    public String search(){
        //반환이 String이면 이때 넘겨주는 값은 ("part2/hello) 뷰이다.
        return "part2/hello";
    }
}

```

WEB-INF/jsp/hello.jsp

```

<%@ page language="java" contentType="text/html; charset=EUC-KR"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=EUC-KR">

```

```
<title>Insert title here</title>
</head>
<body>
    spring mvc start~~~~~
</body>
</html>
```

[예제-Controller클래스 메소드에서 ModelAndView객체 반환]

HelloCommandController.java

```
package part2;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloCommandController {

    //helloCommand.htm으로 컨트롤러를 호출할때 파라미터를 넘겨준값은
    //search메소드에 인자로 정의된 HelloVo vo객체에 저장한다.
    //View에서 HelloVO vo에 있는 값을 출력할때는
    //클래스 첫글자를 소문자로 시작해서 사용한다.
    //${helloVO.id}

    // http://localhost:8090/spring_mvc/helloCommand.htm?id=young
    @RequestMapping("/helloCommand.htm")
    public String search(HelloVO vo){
        System.out.println("ssss"+vo.getId());
        return "part2/helloCommand";
    }

    //////////////////////////////////////

    /*
     * @ModelAttribute 어노테이션은 커맨드 객체의 모델 이름을 지정한다.
```



```

* @ModelAttribute("vo")로 지정해 놓았으면
* 뷰코드에서 객체의 값을 표현할때는 ${vo.id}로 접근하면된다
*/

```

```

//http://localhost:8090/spring_mvc/helloModel.htm?id=young
@RequestMapping("/helloModel.htm")
public String article(@ModelAttribute("vo") HelloVO vo){

    return "part2/helloModel";

}
}

```

WEB-INF/jsp/helloCommand.jsp

```

<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="EUC-KR"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>
    id:${vo.id}
    name:${name }
</body>
</html>

```

[예제-하나의 URI로 GET요청과 POST요청을 한개의 컨트롤러로 처리]

web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
    <display-name>spring_mvc</display-name>

```

```

<!-- POST방식일때 한글처리 -->
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
    </init-param>
</filter>

<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<servlet>
  <servlet-name>dispatchthree</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>dispatchthree</servlet-name>
  <url-pattern>*.hb</url-pattern>
</servlet-mapping>

```

dispatcherthree.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">

  <!-- 1 컨트롤러 빈 선언 -->
  <bean id="newcall" class="part3.NewCallController" />

  <!-- 2 ViewResolver 빈 선언 -->

```

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/" />
    <property name="suffix" value=".jsp" />
</bean>
</beans>
```

MemberVO.java

```
package part3;

public class MemberVO {
    private String id;
    private String name;

    public String getId() {
        return id;
    }
    public void setId(String id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

NewCcallController.java

```
package part3;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

/*
 * 하나의 URI로 GET요청과 POST요청을 한개의 컨트롤러로 처리할때 사용해주는 클래스이다.
 * 1 GET요청이 들어오면 form()메소드를 실행한다
 */
```

```

* 2 POST요청이 들어오면 submit()메소드를 실행한다.
* 3 GET요청이나 POST요청이 똑같은 URI을 사용한다.
* http://localhost:8090/spring_mvc/newcall.hb
*/

```

```
@Controller
```

```
//@RequestMapping("/newcall.hb")
```

```
public class NewCallController {
```

```
    @RequestMapping(value="/newcall.hb",method=RequestMethod.GET)
```

```
    //@RequestMapping(method=RequestMethod.GET)
```

```
    public String form(){
```

```
        return "part3/memForm";
```

```
    }
```

```
    @RequestMapping(value="/newcall.hb",method=RequestMethod.POST)
```

```
    //@RequestMapping(method=RequestMethod.POST)
```

```
    public String submit(MemberVO vo){
```

```
        return "part3/memSubmit";
```

```
    }
```

```
}
```

WEB-INF/jsp/memForm.jsp

```
<% @ page language="java" contentType="text/html; charset=UTF-8"
```

```
    pageEncoding="UTF-8"%>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<title>Insert title here</title>
```

```
</head>
```

```
<body>
```

```
<form name="frm" action="newcall.hb" method="POST">
```

```
아이디:<input type="text" name="id" size="20" /> <br/>
```

```
이름:<input type="text" name="name" size="20" /><br/>
```

```
<input type="submit" value="전송" />
```

```
</form>
```

```
</body>
```

```
</html>
```

memSubmi.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
```

```
    pageEncoding="UTF-8"%>
```

```
<!DOCTYPE    html    PUBLIC    "-//W3C//DTD    HTML    4.01    Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<title>Insert title here</title>
```

```
</head>
```

```
<body>
```

```
아이디:${memberVO.id} <br/>
```

```
이름:${memberVO.name}<br/>
```

```
</body>
```

```
</html>
```

5. 스프링 프레임워크 활용 프로젝트

5-1. 스프링 MVC패턴을 이용한 게시판 프로젝트 구성

1. DataSource 설정(JNDI를 이용한 설정방법)

톰캣서버(context.xml)에 <context> 태그 아래 다음을 등록한다.

```
<Resource auth="Container" driverClassName="oracle.jdbc.driver.OracleDriver"
maxActive="100" maxIdle="30" maxWait="10000" name="jdbc/oracle"
password="1234" type="javax.sql.DataSource" url="jdbc:oracle:thin:@localhost:1521:XE
username="hr" />
```

스프링설정(xxxxx-servlet.xml)

```
<bean id="springDataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
<property name="jndiName" value="jdbc/oracle"/>
<property name="resourceRef" value="true"/>
</bean>
java:comp/env/jdbc/oracle 를 JNDI에서 lookup 한다.
```

2. web.xml 설정

```
<servlet>
<servlet-name>spring</servlet-name>
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>spring</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
```

3. JdbcTemplate 클래스 설정(JDBC를 이용해 데이터에 접근한다.)

스프링설정(xxxxx-servlet.xml)

```
<bean id="springJdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
<property name="dataSource" ref="springDataSource"/>
</bean>
```

설정파일내부에서 참조하는 ref 값은 bean id 값이다.

#목록#

1. 리스트페이지 만들기

```
<% @ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<% @ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">

<% @page import="java.util.List"%>

<% @page import="board.BoardVO"%>

<% @page import="board.PageVO"%><html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<table width="80%">

<tr>

<td>

<a href="write.htm">글쓰기</a>

</td>

</tr>

</table>

<table border="1" width="80%">

<tr>

<th>번호</th><th>제목</th><th>조회수</th><th>ip</th>

</tr>

<c:forEach var="list" items="${list}">
```


2. DAO 생성

@Override

```
public int getCount() {  
    int i=0;  
    i=(Integer)getSqlMapClientTemplate().queryForObject("cnt");  
    return i;  
}
```

@Override

```
public List<BoardVO> getAll(PageVO pv) {  
    List<BoardVO> vo=null;  
    try{  
        vo=getSqlMapClientTemplate().queryForList("all", pv);  
    }catch(Exception e){  
        if(vo==null) vo=new ArrayList<BoardVO>();  
    }  
    return vo;  
}
```

3. Controller 만들기(비즈니스 로직)

```
package board;
```

```
import java.util.List;

import org.springframework.stereotype.Controller;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.ModelAndView;
```

@Controller

```
public class ListController {

    private BoardDaoImp dao;

    List<BoardVO> list;

    PageVO pv;

    int currentPage=1; //현재 페이지번호

    int blockCount=3; //한페이지당 보여줄 레코드 수

    int blockPage=2; //한 블록당 보여줄 페이지 수


    public void setDao(BoardDaoImp dao) {

        this.dao = dao;

    }


    @RequestMapping("/list.htm")

    protected ModelAndView listMethod(PageVO pv){

        int str=pv.getCurrentPage();
```

```
currentPage=(str==0 ? 1 :str);

int totalCount=dao.getCount();

if(totalCount>=0){

    pv=new PageVO(currentPage,totalCount,blockCount,blockPage);

    list=dao.getAll(pv);

}

ModelAndView mv=new ModelAndView();

mv.addObject("currentPage",currentPage);

mv.addObject("list",list);

mv.addObject("pv",pv);

mv.setViewName("list");

return mv;

}
```

```
}
```