Spring Framework

1. Spring Framework

- Spring Framework 개요
- Spring Framework 특징
- Spring Framework 장점
- Spring Framework 모듈

Spring Framework 개요(1/2)

일반 자바 객체의 생성 -> 사용 -> 소멸 자체를 스프링 프레임웍에게 위임(xml, 어노테이션 설정)

servlet / jsp은 web container

Enterprise 개발을 쉽고, 편리하게
개발할 수 있도록 지원해주는
오픈소스 프레임워크로
경량급 애플리케이션 프레임워크라고도 함

java bean =DTO=VO

스프링에서의 자바 빈 = 스프링 프레임웍인 관리하는 모든 자바 객체들
설정파일 - <bean> tag로 사용

Spring Framework 개요(2/2)

- Spring 탄생 배경
  - 창시자 : Road Johnson
  - Spring의 모태
    : "Expert One-on-One J2EE Development without EJB" 라는 책을 통해 제시 했던 예제중 EJB를 사용하지 않고 Enterprise Application 전 계층에 등장하는 기술과 애플리케이션의 전 영역에 대한 효과적인 설계와 개발 기법 소개
  - 2003년 오픈 소스로 시작된 프로젝트

- springsource
  - Spring 창시자인 로드존슨을 주축으로 Spring을 관리하는 전문 기업

Spring Framework 특징(1/10)

Entrerprise Application에서 필요로 하는 기능 제공

경량(lightweight) 애플리케이션 컨테이너

Dependency Injection[DI] 지원

Aspect Oriented Programming[AOP] 지원

Plane Old Java Object[POJO] 지원 –일반 자바 클래스들
Servlet 상속받은 사용자정의 Servlet클래스?POJO 아님

트랜잭션 처리를 위한 일관된 방법 제공

영속성과 관련된 다양한 API 지원 및 연동 지원

Spring Framework 특징(2/10)

Enterprise Application에서 필요로 하는 기능 제공

- Spring은 단순한 툴과 기본적인 개발 환경만으로도 Enterprise 개발에서 필요로 하는 주요한 기능을 갖춘 애플리케이션 개발에 적합

- 예 : 고비용을 요하는 WAS(Web Application Server)를 사용하지 않고도, WAS에서 지원해주는 트랜잭션 관리 및 보안, 객체 pooling과 같은 Enterprise 개발의 고급 기술들도 Sprig Framework를 통해서 단순한 작업 및 설정만으로 동일한 수준의 기능들을 사용할 수 있게 됨

Spring Framework 특징(3/10)

경량(lightweight) 애플리케이션 컨테이너

- EJB
  - 분산형 객체 지향 자바 애플리케이션을 개발하고 보급하기 위한 컴포넌트 아키텍처로 썬마이크로시스템즈에서 **Enterprise 시스템** 개발용으로 제시한 스펙
  - EJB를 대표하는 기존의 많은 기술들은 무겁고 복잡
  - 코드에 불필요하게 등장하던 서버 환경에 의존적인 부분들이 다수 존재

- Spring
  - 서버에 의존적인 부분들이 제거됨
  - 단순한 툴과 기본적인 개발 환경만으로도 Enterprise 개발에서 필요로 하는 주요한 기능을 갖춘 애플리케이션 개발에 적합
  - EJB 적용 애플리케이션 개발에 비해 훨씬 빠르고 간편하게 개발할 수 있기 때문에 생산성과 품질 면에서 볼때 '경량급' 프레임워크라고 부르게 된 것

Spring Framework 특징-경량(Lightweight) 컨테이너(4/10)

경량(lightweight) 애플리케이션 컨테이너

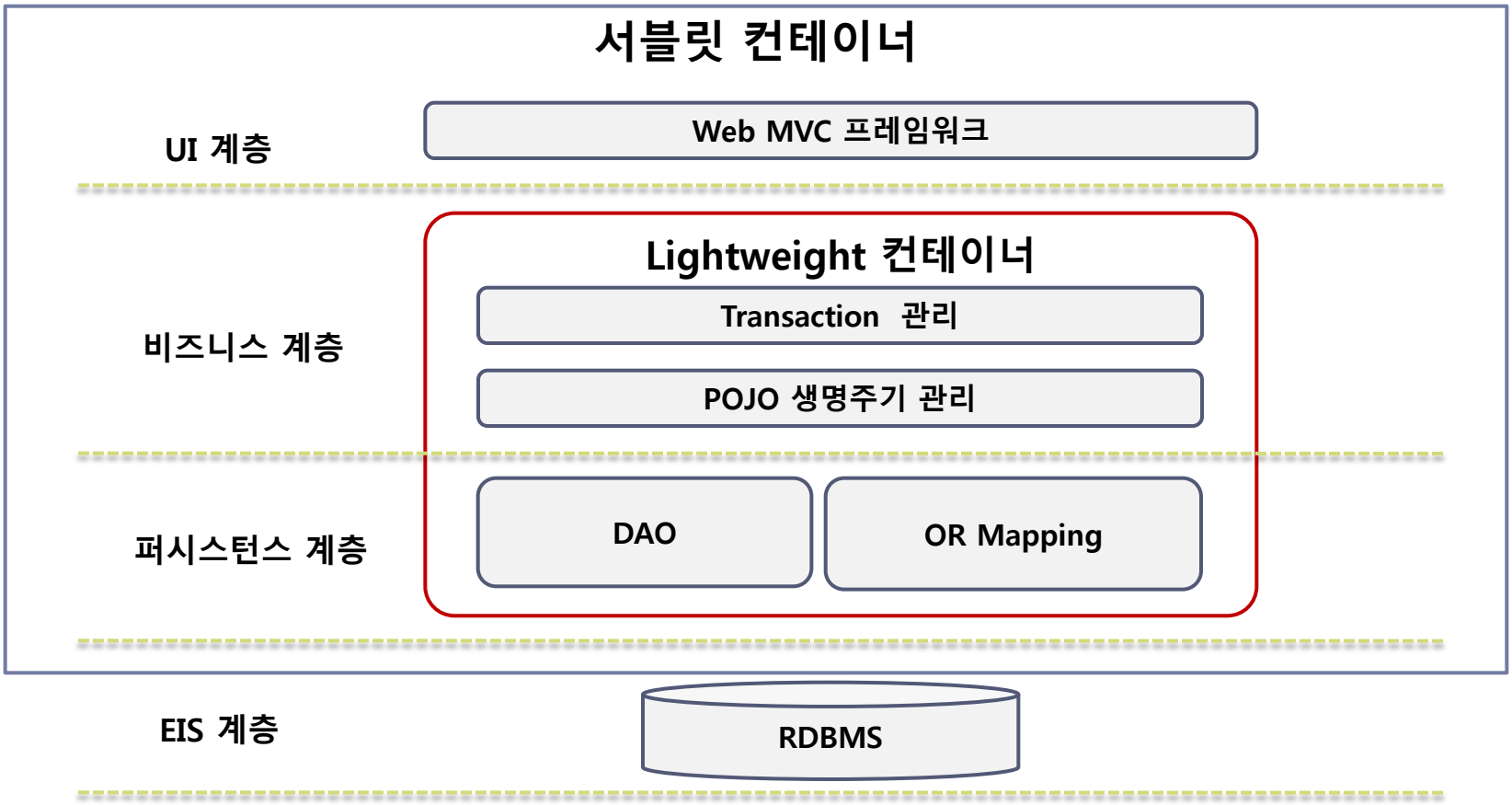
- IoC(Inversion of Control: 역제어) 컨테이너
  - 개발자가 직접 객체를 생성을 하지 않고, 객체의 생성에서 소멸까지 컨테이너가 관리
  - Dependency Injection을 통해 객체간의 의존성 주입

- Lightweight 컨테이너
  - EJB 컨테이너에 비해 가벼운 Ioc 컨테이너
  - 컨테이너의 API에 의존적이지 않은 POJO 관리

POJO(Plain Old Java Object)
특정 인터페이스 또는 클래스를 상속하지 않는 일반 자바 객체
Cf) Servlet 개체는 HttpServlet를 반드시 상속 , EJB 개체는 SessionBeane을 반드시 구현

Spring Framework 특징-경량(Lightweight) 컨테이너(5/10)

경량(lightweight) 애플리케이션 컨테이너



Spring Framework 특징(6/10)

Dependency Injection[DI] 지원

- Spring은 설정 파일이나 애노테이션을 통해서 객체간의 의존 관계를 설정 할 수 있도록 함
- 객체는 의존하고 있는 객체를 코드 상에서 직접 생성하고나 검색할 필요가 없음

Spring Framework 특징(7/10)

Aspect Oriented Programming[AOP] 지원

- Spring은 자체적으로 AOP를 지원하고 있기 때문에 트랜잭션이나 로깅, 보안과 같이 여러 비즈니스 모듈에서 공통적으로 필요로 하는 공통관심 사항을 핵심 로직과 분리시켜 각 모듈에 적용할 수 있음

- 중복 코드 삭제

Spring Framework 특징(8/10)

Plane Old Java Object[POJO] 지원

- 특정 규약 및 환경에 종속적이지 않은 평범한 일반 자바 클래스 의미
- Spring 개발에 POJO 클래스를 활용할 수 있다는건 특정 구현 기술에 종속적이지 않다는 의미
- 개발 후의 테스트시에도 DB와 특정 서버 없이도 테스트를 할 수 있기 때문에 개발이 빨라진다는 장점이 있음

Spring Framework 특징(9/10)

트랜잭션 처리를 위한 일관된 방법 제공

- JDBC API 및 JTA를 사용하거나 컨테이너가 제공하는 트랜잭션을 사용하든, 설정 파일을 통해 트랜잭션 관련 정보를 관리하기 때문에 특정 트랜잭션 구현 방법에 상관없이 동일한 코드를 여러 환경에서 사용 가능

- 선언적인 트랜잭션을 지원하여 코드를 수정하지 않고도 트랜잭션을 적용 및 변경 가능

Spring Framework 특징(10/10)

영속성과 관련된 다양한 API 지원 및 연동 지원

- Spring은 JDBC API를 비롯하여 iBatis/Hibernate, JPA등 데이터베이스 처리를 위해 널리 사용되는 library들과의 연동을 지원

Spring Framework 장점

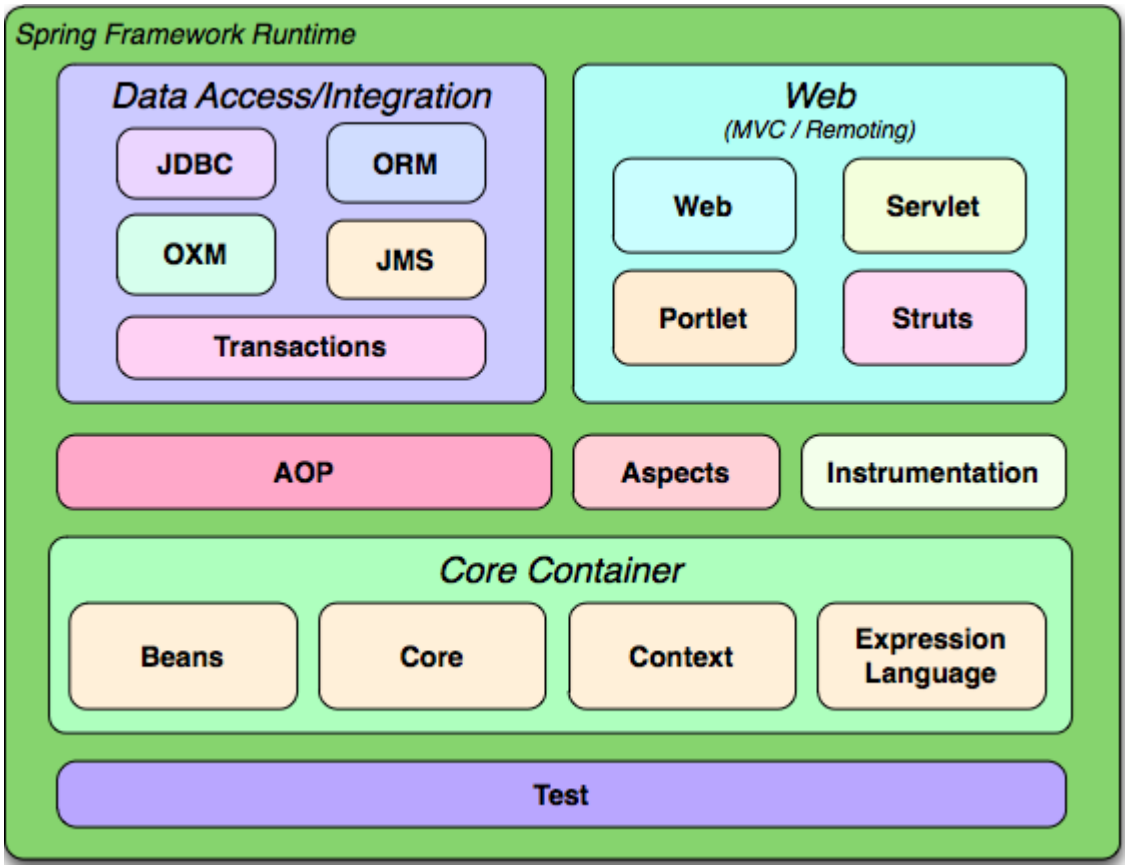
- 개발자들이 개발하고자 하는 애플리케이션 로직 개발에만 집중할 수 있음
  - 기술에 대한 접근 방식이 일관성이 없거나, 특정 환경에 종속적이지 않음
  - 따라서 실행 로직의 기능이 변경되는 것이 아니라 서버 등의 실행 환경이 바뀌고 적용되는 조건이 바뀐다 해도 코드까지 수정할 필요가 없음

- 개발이 단순해짐
  - Spring의 의존 관계, 트랜잭션등의 설정 방법에 대한 지식을 습득한 후에는 설정 적용 기술만으로도 Enterprise 개발의 기술적인 복잡함과 그에 따른 수고를 제거 가능

- POJO 방식의 기술 사용이 가능
  - 특정 규약 및 환경에 종속되지 않은 일반 자바 클래스를 지원하므로 컨테이너에 의존적인 코드를 추가하지 않아도 애플리케이션을 개발 할 수 있음
  - 개발후의 테스트도 쉽고 빠르게 할 수 있음

Spring Framework의 모듈(1/2)

- Spring3.0은 20여개의 모듈로 구성



[그림 출처] http://www.springsource.org

Spring Framework의 모듈(2/2)

모듈 명	설 명
Beans	BeanFactory 인터페이스를 통해 구현됨
Core	프레임워크의 가장 기본적인 부분 컨테이너 기능을 수행하기 위해 의존성 주입[DI] 기능을 제공
Context	spring-core, spring-beans 모듈을 확장해서 국제화, 이벤트 처리, 리소스 로딩, 서블릿 컨테이너를 위한 컨텍스트 생성 등의 기능을 추가로 제공. ApplicationContext 인터페이스를 통해 구현됨
Expression Language	객체에 접근하고 객체를 조작하기 위한 표현 언어를 제공
AOP	AOP Alliance에 호환되는 AOP 구현을 제공
Aspects	AspectJ와의 통합을 제공
Web(MVC/Remoting)	Spring MVC를 제공하며 struts와도 연동 기능 제공등 웹 관련 기능 지원
Data Access/Integration	JDBC를 위한 템플릿 제공. 따라서 간결한 코드로 JDBC 프로그램 가능, iBatis 및 하이버네이트 등의 ORM api를 위한 통합 레이어 제공. Spring이 제공하는 트랜젝션과의 연동 지원

2. Spring Framework 개발 환경 구축

- Spring Framework 다운로드
- Spring IDE 설치
- Spring Project 를 위한 구성
- Spring Project 생성해 보기

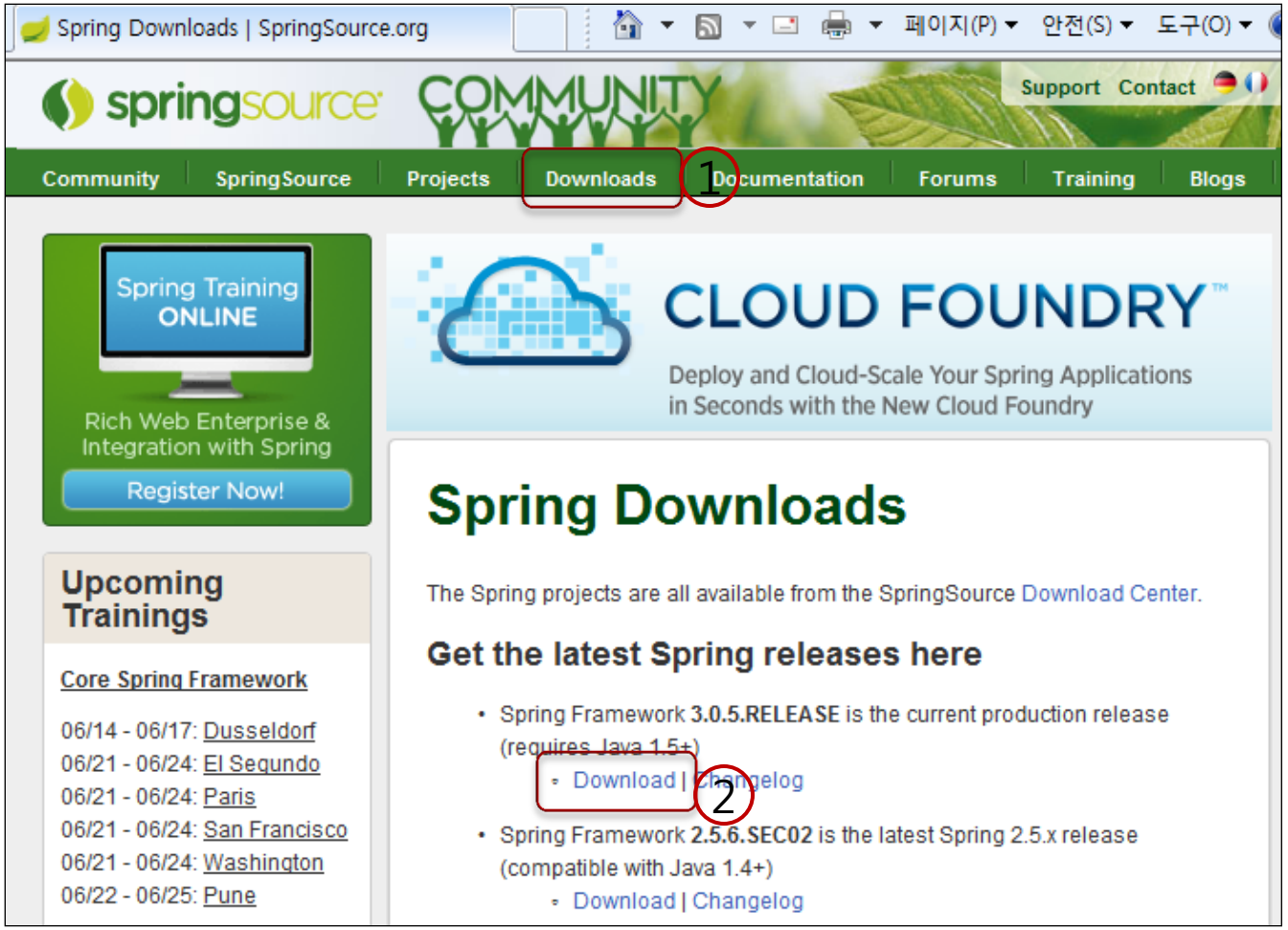
Spring Framework 다운로드(1/2)

- 사이트에서 받기
  - http://www.springsource.org/
  - 라이브러리 설정 위치
    - 프로젝트명/[WebContent]/WEB-INF/lib

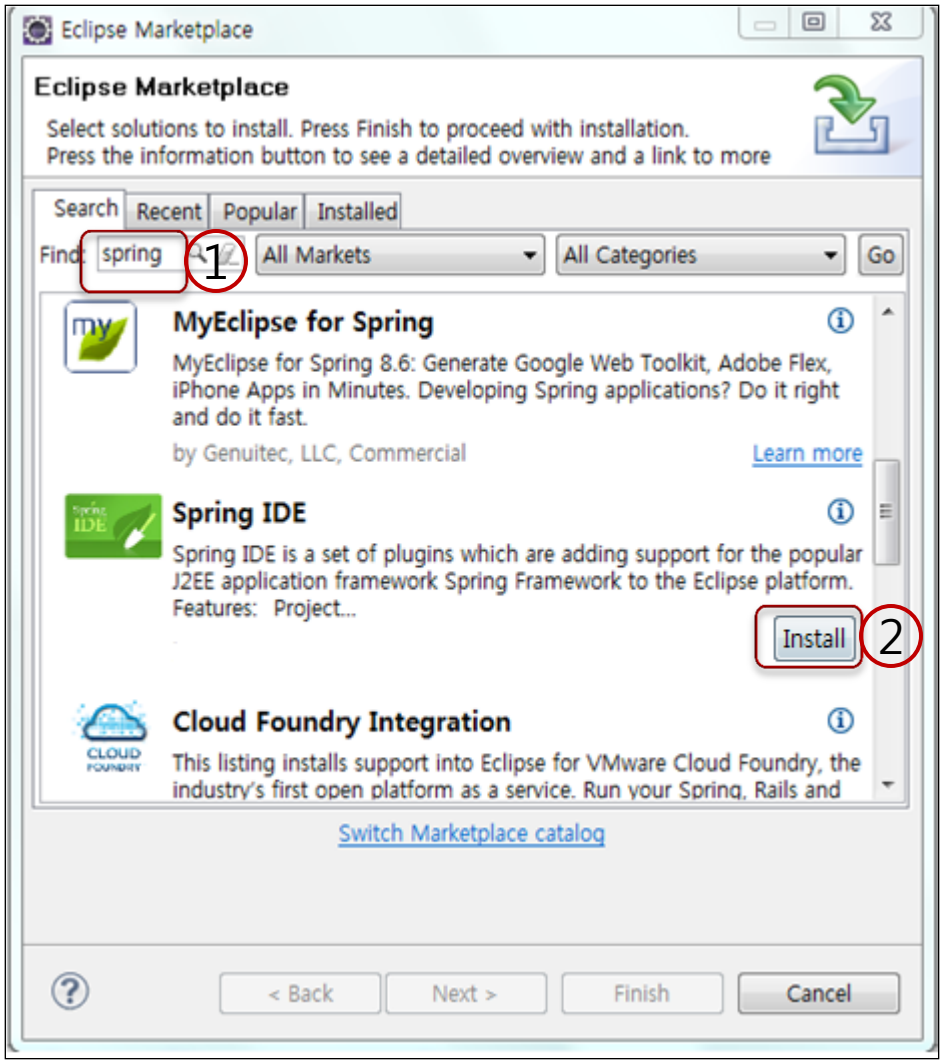
- m2eclipse 이용하기
  - 의존 관계의 library들도 자동 다운로드

Spring Framework 다운로드(2/2)

- http://www.springsource.org/



Spring IDE 설치



Spring IDE 설치

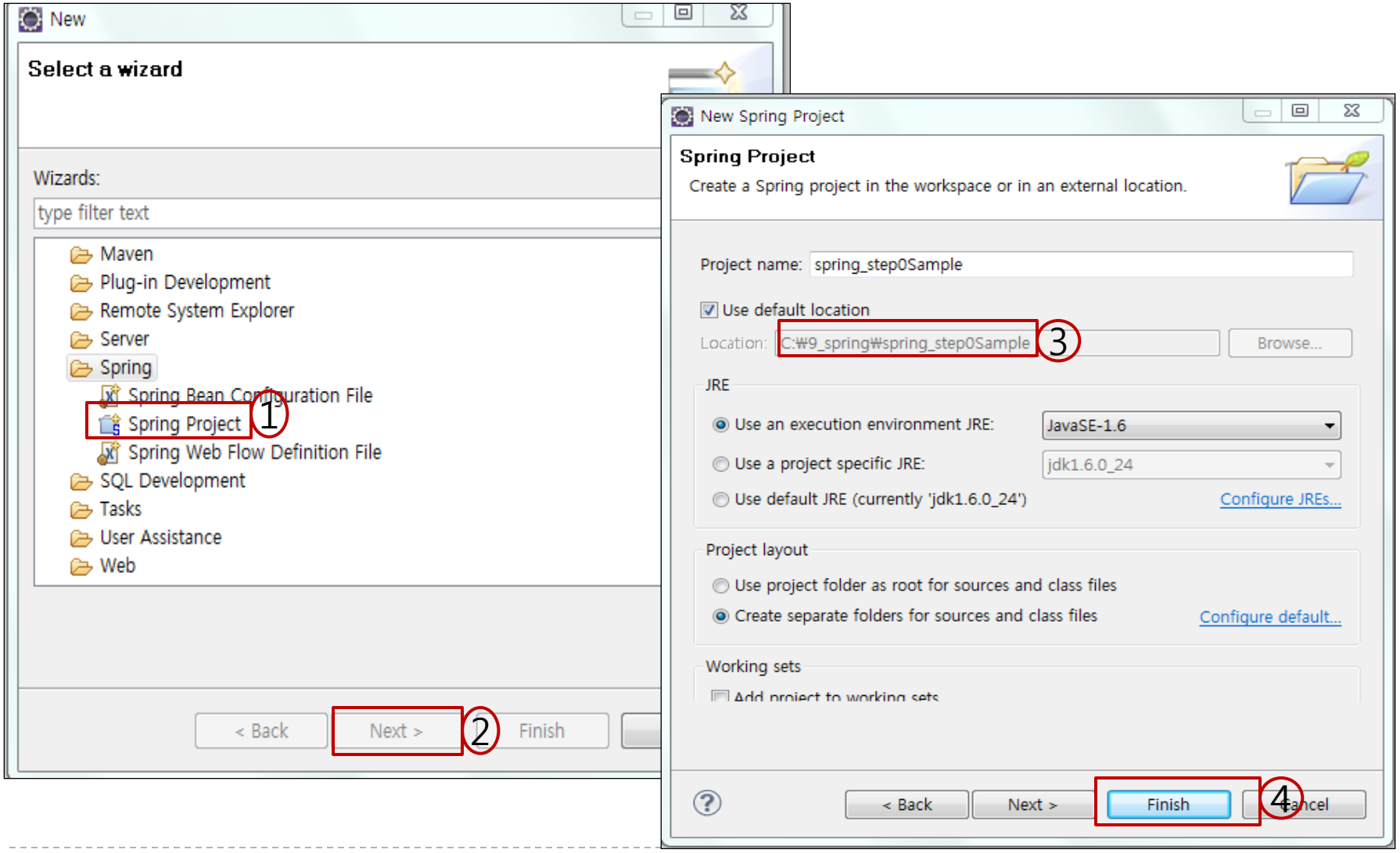
Spring Project 를 위한 구성

- Spring Framework library

- 설정 메타정보 파일
  - Sprind은 설정 메타데이터 정보를 필요로 함
  - 이 설정 메타데이터는 Spring 컨테이너에 객체 생성 및 관계 설정 내용을 XML 또는 properties[프로퍼티] 파일, 소스코드 애노테이션과 같은 외부 리소스로 작성

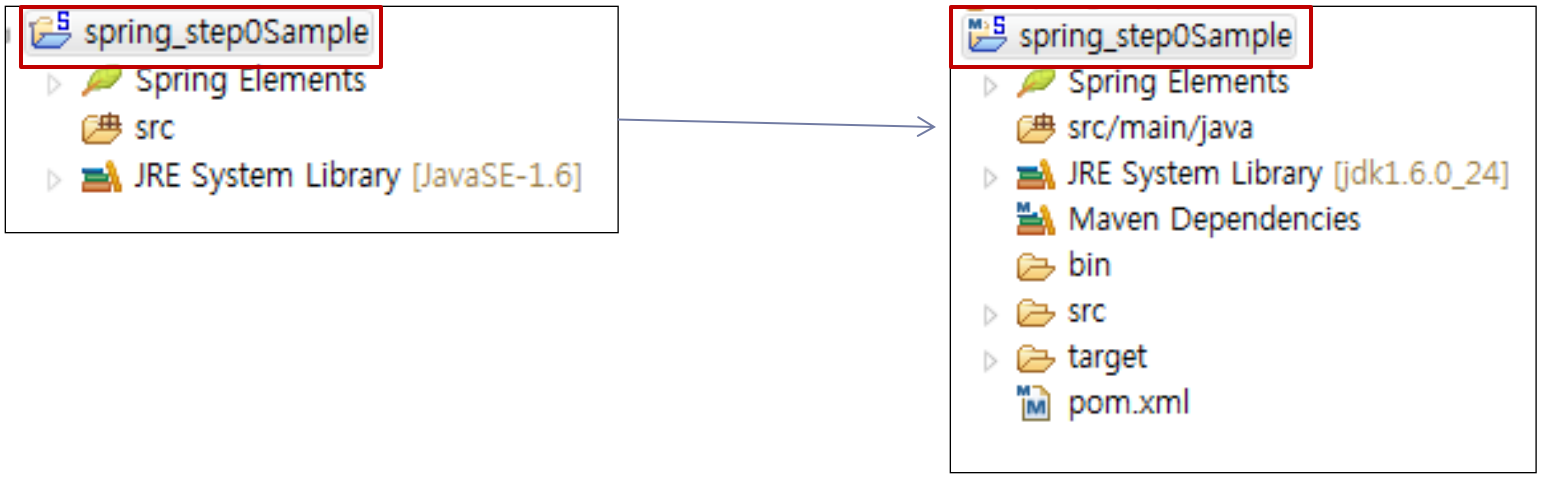
- 자바 소스들
  - Spring 빈
  - Spring 빈 사용 클래스들
  - 이외의 자바 클래스들

Spring Project 생성해 보기



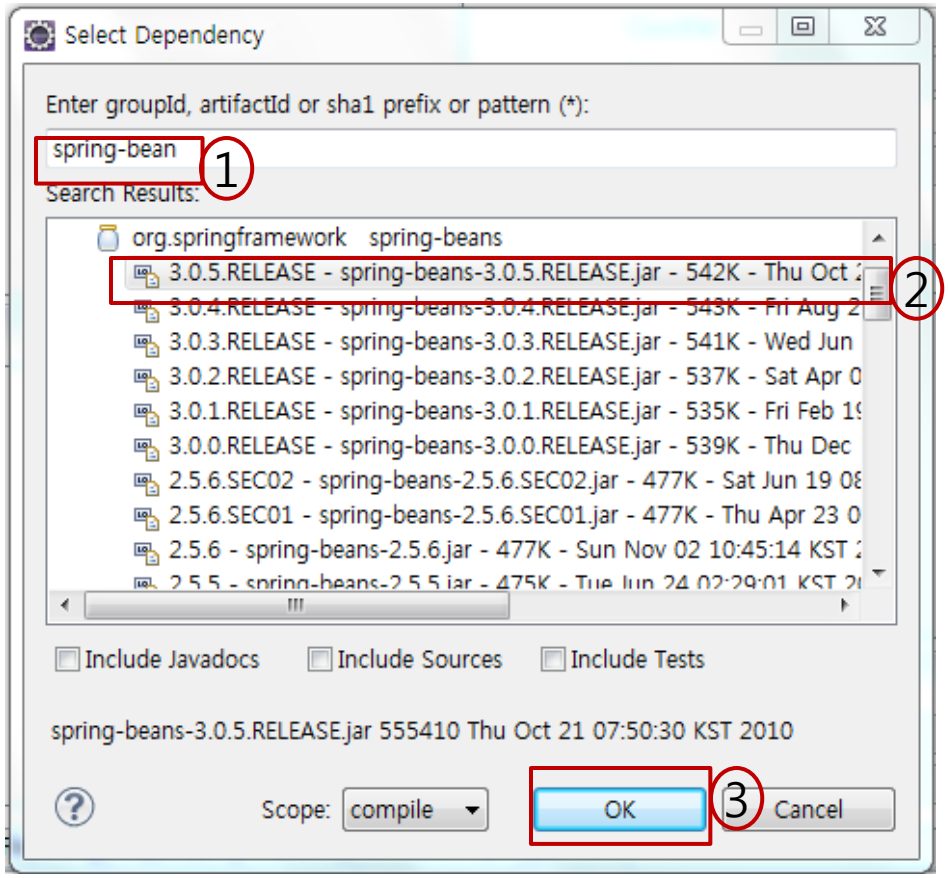
Spring Project 생성해 보기

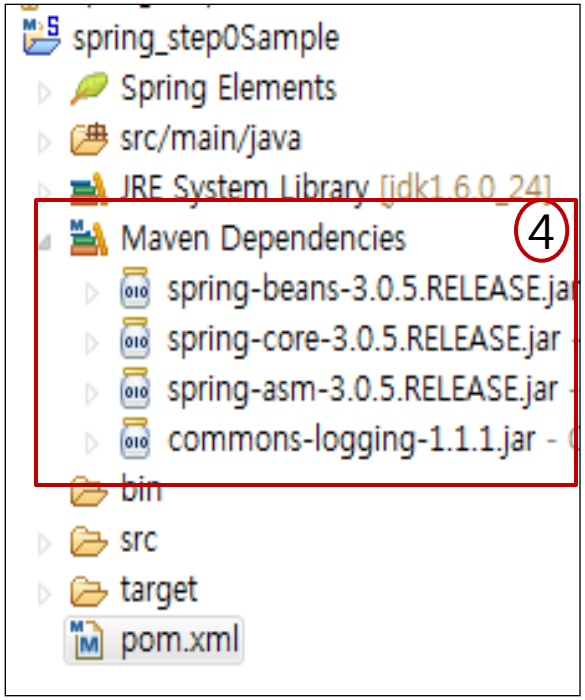
- 생성된 project를 maven 기반 project로 변환
  - Maven 기반의 project로 업그레이드
  - 소스 개발을 위한 src/main/java 폴더 생성



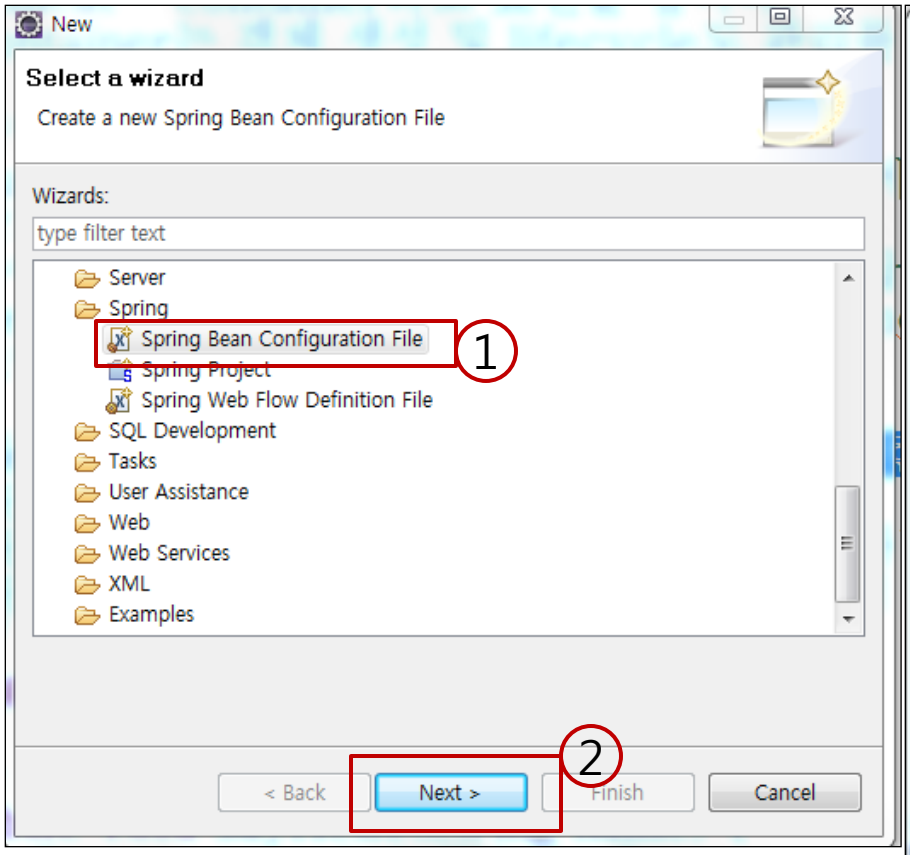
Spring Project 생성해 보기 – Spring library 다운로드

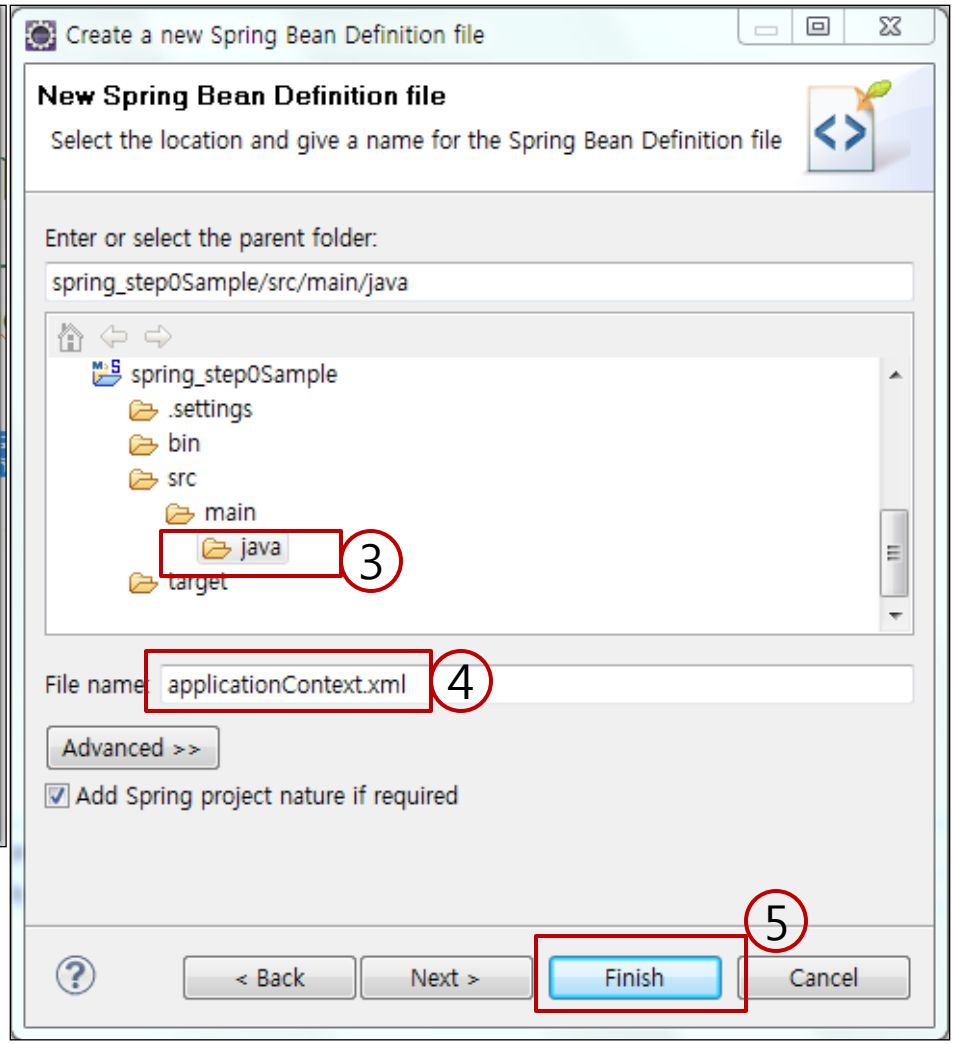
- spring-bean.jar와 의존 관계의 library들이 자동 설정



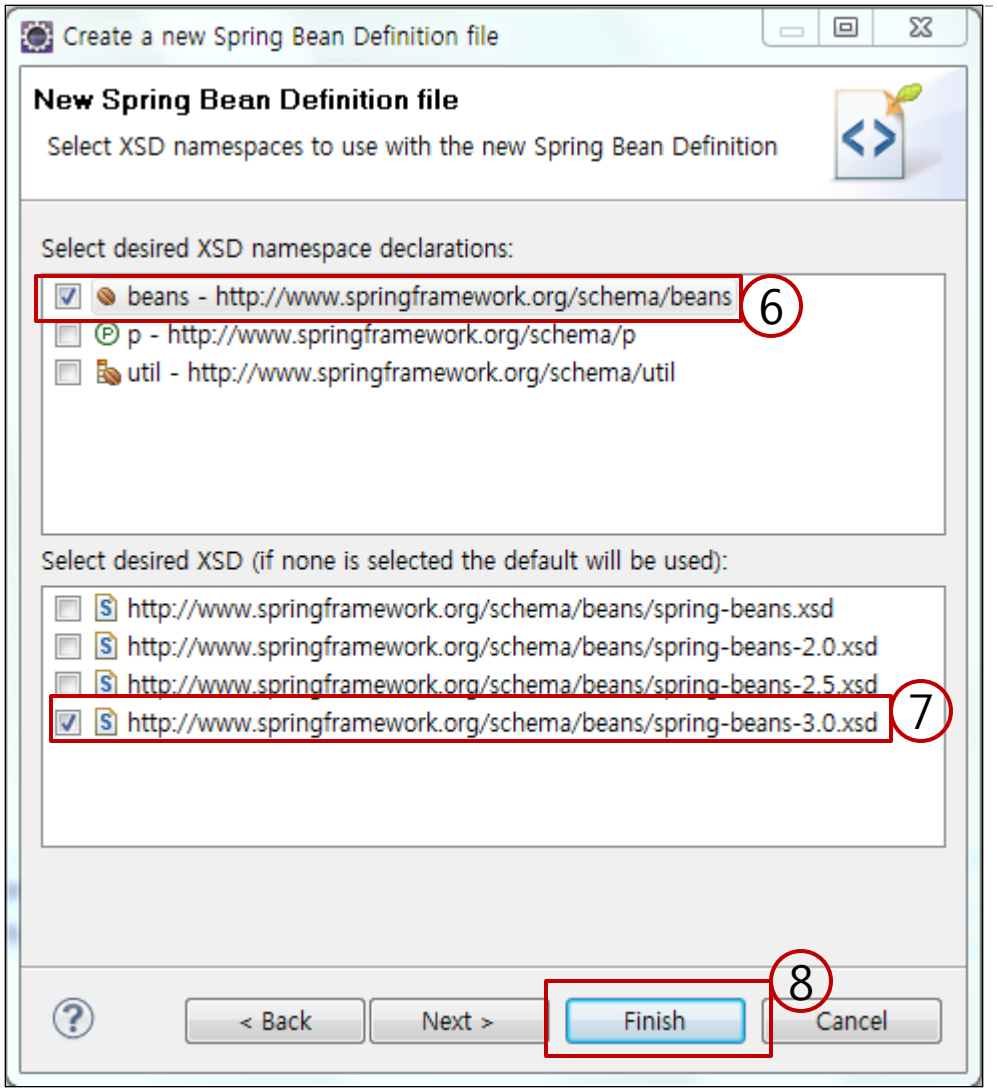


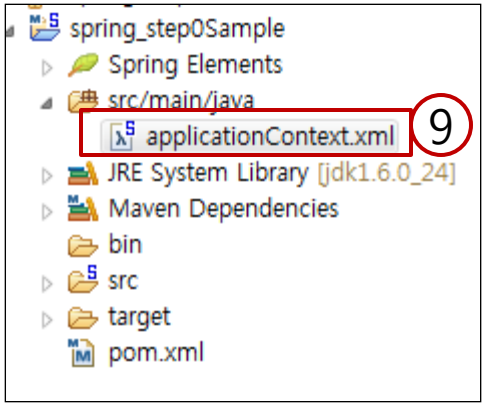
Spring Project 생성해 보기 - Spring 설정 파일 생성하기



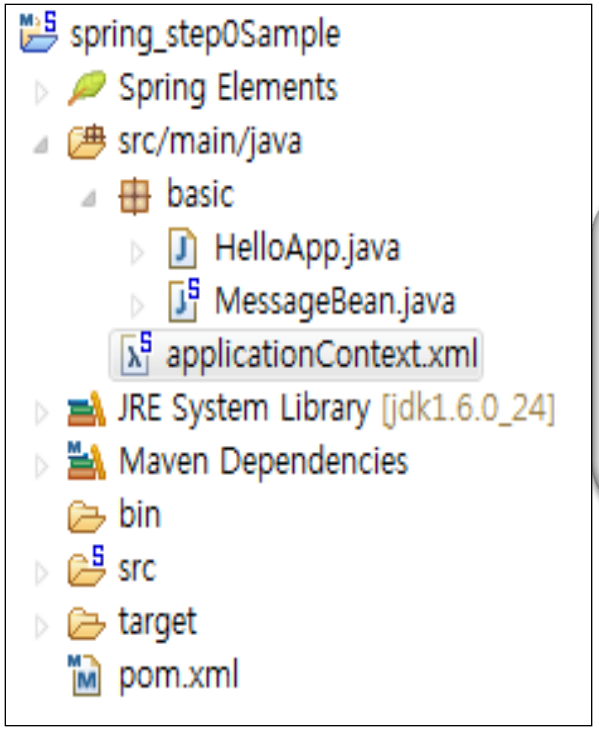


Spring Project 생성해 보기 - Spring 설정 파일 생성하기





Spring Project 생성해 보기 - 샘플 예제 구조 및 실행 원리



<<설정 파일>>
applicationContext.xml
파일명 임의 설정 가능
어노테이션으로 대체 가능

<<POJO>>
basic.MessageBean

Spring Framework

<<main()>>
basic.HelloApp

Spring Project 생성해 보기- Spring Bean으로 활용될 클래스

- MessageBean.java

```java
package basic;

public class MessageBean extends Object {

    private String day;
    public MessageBean() {}

    public MessageBean(String day) {
        this.day = day;
    }

    public String getDay() {
        return day;
    }

    public void setDay(String day) {
        this.day = day;
    }

    public void sayHello(String name) {
            System.out.println(day + "에 방문하신 " + name + "님 반갑습니다" );
    }
}
```

Spring Project 생성해 보기- Spring 설정 파일에 Spring 빈 등록

- applicationContext.xml
- Ctrl + space 키보드를 이용해서 xml 문서 작성

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <beans xmlns="http://www.springframework.org/schema/beans"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
6                         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7
8
9      <bean  id="message"  class="basic.MessageBean">
10       <property name="day"  value="월요일"></property>
11     </bean>
12
13 </beans>
```

Spring Project 생성해 보기 - Spring 설정 파일 생성하기

- applicationContext.xml

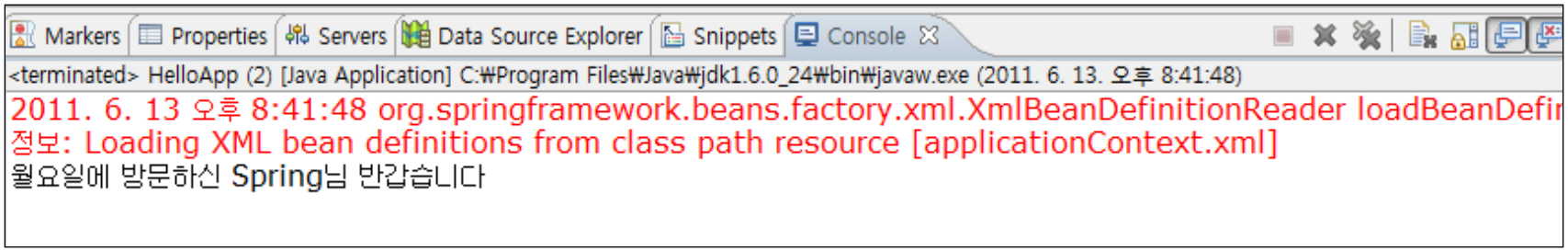
applicationContext.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <beans xmlns="http://www.springframework.org/schema/beans"
4     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5     xsi:schemaLocation="http://www.springframework.org/schema/beans
6     http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
7
8
9  </beans>
```

Spring Project 생성해 보기 - 실행을 위한 자바 클래스 및 실행 결과

- Hello.java

```java
package basic;

import org.springframework.beans.factory.BeanFactory;

public class HelloApp {
    public static void main(String[] args) {

        Resource resource = new ClassPathResource("applicationContext.xml");
        BeanFactory factory = new XmlBeanFactory(resource);
        MessageBean bean = (MessageBean)factory.getBean("message");

        bean.sayHello("Spring");
    }
}
```



3. Dependency Injection

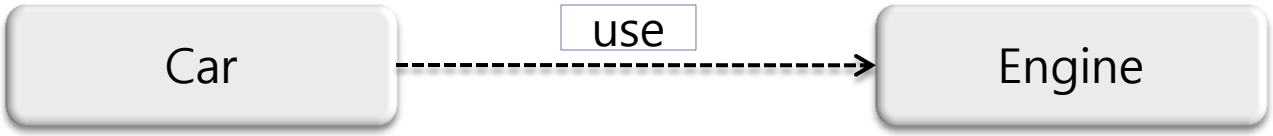
- DI 개요
- Spring 컨테이너
- Constructor Injection
- Setter Injection
- 컬렉션 타입의 설정
- 의존관계 자동 설정
- 빈 객체 범위

Dependency Injection[DI]

- 의존성 주입
- Spring Framework가 지원하는 핵심 기능
- 객체 사이의 의존 관계가 객체 자신이 아닌 외부(조립기)에 의해 설정됨
- 컨테이너의 역할
  - A 객체가 필요로 하는 의존 관계에 있는 다른 객체 B 객체를 직접 생성하여 A 객체로 주입(설정)해주는 역할을 담당

DI - 의존 관계 이해하기

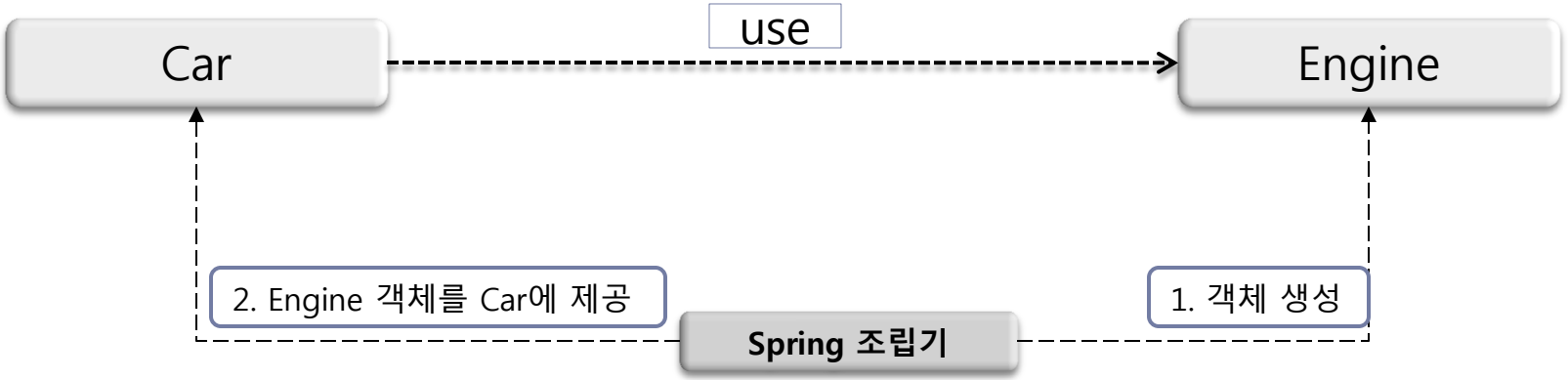
- Car 클래스와 Engine 클래스 의존관계



1. Car 클래스는 Engine 클래스에 의존되어 Engine 클래스에 의존
2. Engine 클래스가 이미 정의되어 있던 메소드 형식이 바뀐다거나 새로운 메소드가 추가되면 Car 클래스도 영향을 받아 Car 클래스의 코드도 변함
3. Car 클래스가 변경된다고 해서 Engine 클래스가 종속적으로 변경되지 않음
4. Car 클래스는 Engine 클래스에 의존하고 있지만, Engine 클래스는 Car 클래스에 의존하지 않는 구조
5. 의존 관계는 방향성이 부여되어 있음

DI(Dependency Injection)이란?

- 구체적인 의존 객체와 그것을 사용할 객체를 런타임에 연결해주는 작업



- Car 객체가 실행되면서 사용할 Engine 객체를 런타임에 주입 받음
- Spring은 객체를 생성하고 객체간의 의존 관계를 자동 설정해주는 조립기 기능을 보유한 컨테이너가 자동으로 제공(주입)
- 용어 정리
  - 조립기(Assembler) - 객체를 생성하고 객체간의 의존 관계를 자동 설정해 줌)

Dependency Injection 장점

- DI 개념을 적용한 프로그램들은 이 개념을 적용하지 않은 프로그램들에 비해 **설계가 쉽고, 추후 이미 개발되어 있는 프로그램에 변경 사항이 발생했을 경우라 하더라도 변경 내용 적용이 용이하므로 확장성이 매우 좋음**

- 각 객체간의 의존 관계와 객체들의 생명주기를 Spring을 기반으로 간편하게 개발 및 유지 보수 하는 메커니즘을 제공받는 것

DI - Spring Bean과 IoC 컨테이너

- **Spring 빈(bean) 이란?**
  - **Spring이 제어권을 가지고 생성 및 객체간의 관계를 관리하는 객체 의미**
  - **형태는 제한이 없음**
  - **기존 일반 자바빈(DTO=VO)와는 다른 개념**

- IoC 컨테이너 또는 Spring 컨테이너란?
  - Spring 빈의 생성과 관계 설정, 사용, 생명주기 관리 등을 관장하는 기능을 제공해주며 Spring의 주요 핵심 기능이므로 컨테이너라고도 함

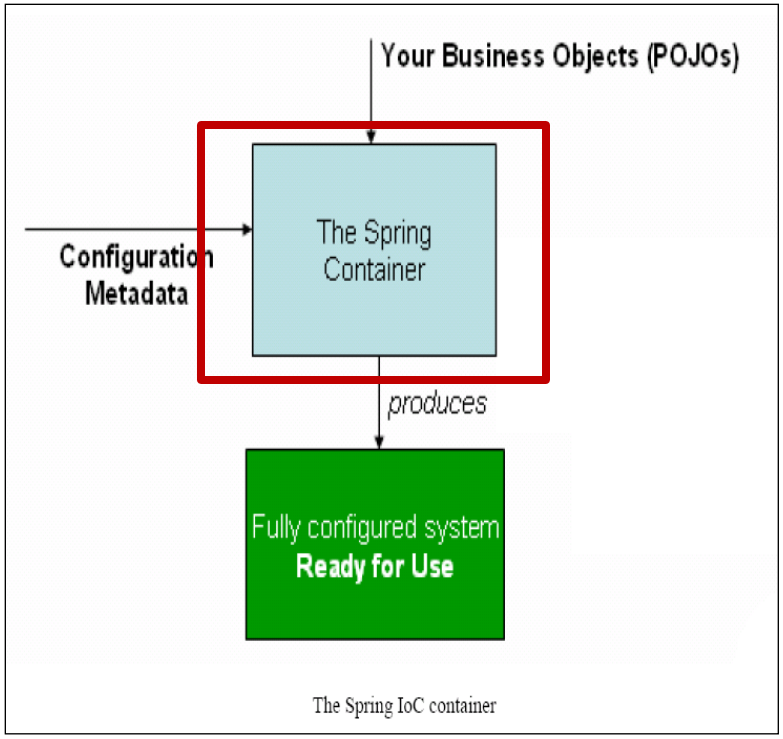
Spring Bean과 IoC[DI] 컨테이너

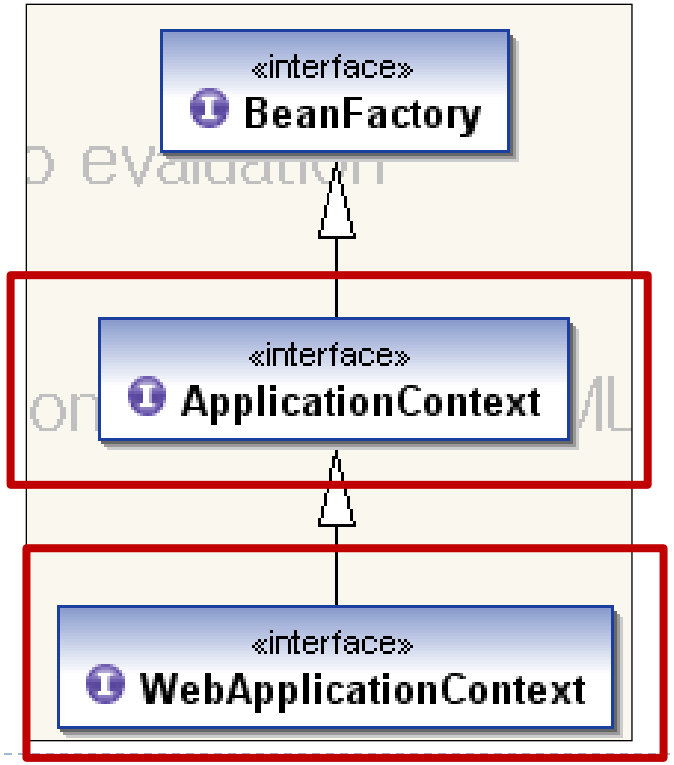
- IoC 컨테이너 또는 Spring 컨테이너 기능
  - 컨테이너 내에는 설정 정보 파일인 XML 파일을 읽어서 객체를 생성하고 관계된 객체의 변수에 자동 대입해 주는 조립기 기능이 내장되어 있음

- 의존성 주입 용어 정리
  - IoC는 DI
    - 런타임에 객체간의 의존성 주입(Dependency Injection) 을 하기 때문에 IoC를 직관적으로 Dependency Injection 줄임말인 DI라고 명하기도 함

Spring 컨테이너 종류

- **빈팩토리 또는 애플리케이션컨텍스트란?**
  - Spring에선 빈의 생성과 관계 설정, 사용, 제거등의 기능을 담당하는 컨테이너를 의미
- 컨테이너 기능을 담당하는 Spring의 주요 API





Spring 컨테이너 - BeanFactory

org.springframework.beans.factory.BeanFactory

- 기능
  - 빈의 생성과 빈 사이의 관계 설정 같은 단순 제어만을 담당하는 DI의 기본 기능에 초점을 맞춘 팩토리로 Spring 컨테이너의 핵심
- 원리
  - 빈 팩토리를 상속받고 있는 하위 클래스들로 XML과 같은 외부 설정 파일의 내용을 기반으로 객체 생성을 하게 되면 Spring 컨테이너가 생성되는 것
- 주요 자식 API
  - XML 파일로부터 설정 정보를 활용하는 가장 많이 사용되는 하위 클래스

    - org.springframework.beans.factory.xml.XmlBeanFactory

Spring 컨테이너 - BeanFactory

```
19      Resource resource = new ClassPathResource("applicationContext.xml");
20      BeanFactory factory = new XmlBeanFactory(resource);
21
22      MessageBean bean = (MessageBean)factory.getBean("message");
```

Spring 컨테이너 - ApplicationContext

org.springframework.context.ApplicationContext

- BeanFactory 인터페이스를 상속받은 하위 인터페이스
- 기능
  - BeanFactory의 빈관리 기능 이외에 Enterprise 애플리케이션을 개발하는데 필요한 여러 가지 컨테이너 기능을 추가한 것
    - 메시지의 국제화
    - 리소스 관리 및 로딩
    - 이벤트 처리 등의 유용한 부가 기능
- 장점
  - Spring과 Spring이 관리하는 리소스를 선언적으로 설정하고 관리할 수 있게 해 주며, 일반적으로 Spring 컨테이너라 하면 애플리케이션컨텍스트를 의미

Spring 컨테이너 종류 - ApplicationContext

org.springframework.context.ApplicationContext

- 주요 자식 API
  - org.springframework.context.support.ClassPathXmlApplicationContext
    - 클래스패스에 위치한 XML파일로부터 설정 정보 로딩
  - org.springframework.context.support.FileSystemXmlApplicationContext
    - 파일 시스템에 위치한 XML파일로부터 설정 정보 로딩
  - org.springframework.web.context.suppirt.XmlWebApplicationCOntext
    - 웹 애플리케이션에 위치한 XML파일로부터 설정 정보 로딩

Spring 컨테이너 - ApplicationContext

```
AbstractApplicationContext context
                            = new ClassPathXmlApplicationContext("applicationContext.xml" );
LoginProcessor loginProcessor = context.getBean("loginProcessor", LoginProcessor.class);

Locale locale = Locale.getDefault();
String greeting = context.getMessage("greeting", new Object[0], locale);
```

Spring 컨테이너 종류 - WebApplicationContext

org.springframework.context. WebApplicationContext

- 기능
  - Spring 애플리케이션에서 가장 많이 사용되는 컨텍스트로, 웹 애플리케이션을 위한 애플리케이션컨텍스트
  - 하나의 웹 어플리케이션마다 한 개씩 존재
  - 웹 어플리케이션을 위해 추가적으로 제공되는 빈 영역(bean scope)을 정의

- 주요 자식 API
  - org.springframework.web.context.support .XmlWebApplicationContext
    - 가장 많이 사용되는 구현 클래스

의존 객체간의 관계도를 적용하는 개발 방법

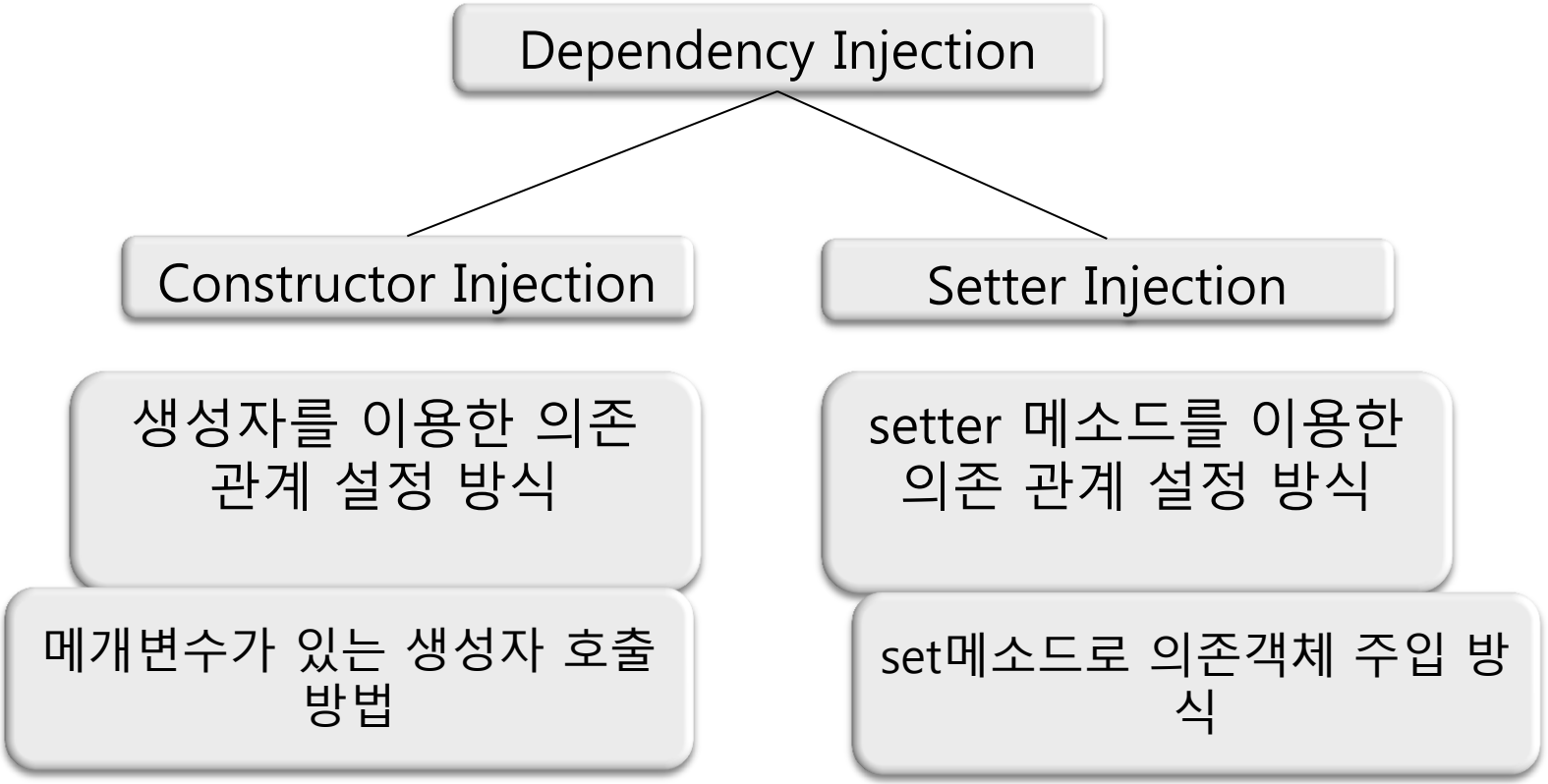
- 직접 의존하는 객체를 코드에 명시하는 방법
  - 단위 테스트가 어려움
  - 의존 객체 변경 시 코드 수정이 불가피

- Factory나 JNDI를 이용하여 검색하는 방법
  - 단위 테스트가 어려움
  - 실제 의존 객체와의 느슨한 의존성 대신 Factory나 JNDI와의 의존성 형성

- 외부의 조립기(Assembler)를 이용하는 방법
  - 단위 테스트 용이
  - 느슨한 의존성

Spring의 Dependency Injection 방법

- 객체간의 의존성을 설정 파일로 simple하게 설정 및 관리



Constructor Injection

- 의존하는 빈 객체를 컨테이너로부터 생성자의 파라미터를 통해서 전달받는 방식

- 클래스를 초기화 할 때 컨테이너로 부터 의존 관계에 있는 특정 리소스인 빈 객체를 생성자를 통해서 할당 받는 방법
  - 설정 파일에 <constructor-arg> 태그를 이용
  - 방법1 – index로 지정하는 방법
    - index는 0부터 시작
  - 방법2 – 타입으로 지정하는 방법

Constructor Injection

Car.java

```
public class Car{
        private Engine engine;
        public Car(Engine engine){
            this.engine = engine;
        }
}
```

applicationContext.xml

```
…
<bean id="engine" class="Engine" />
<bean id="car" class="Car">
        <constructor-arg>
            <ref bean="engine" />
        </constructor-arg>
</bean>
…
```

Setter Injection

- setXxx() 형태의 설정 메소드를 통해서 전달받는 방법으로 프로퍼티 설정 방식

- 의존하는 객체를 전달받을 setter 메소드를 작성
- 설정파일에 **<property>**태그를 이용
  - 객체인 경우 <ref>태그 이용
  - 문자열이나 기본데이터 타입이라면 <value>태그 이용

Setter Injection

Car.java

```
public class Car{
  private Engine engine;
  public void setEngine(Engine engine){
      this.engine = engine;
  }
}
```

applicationContext.xml

```
…
<bean id="engine" class="Engine" />
<bean id="car" class="Car">
      <property name="engine">
          <ref bean="engine" />
      </property>
</bean>
…
```

컬렉션 타입의 프로퍼티 설정

태그	컬렉션 타입
<list>	java.util.List나 배열
<set>	java.util.Set
<map>	java.util.Map
<properties>	java.util.Properties

- 컬렉션 원소가 객체인 경우 <ref> 태그 이용
- 컬렉션 원소가 기본타입인 경우 <value> 태그 이용
  - type 속성 이용하여 타입 지정 가능
  - 제네릭을 이용하면 타입 지정하지 않아도 됨

List 타입의 프로퍼티 설정

CalculatorServiceImpl.java

```
...
private List valueList;
public void setValueList(List valueList) {
      this.valueList = valueList;
}
...
```

applicationContext.xml

```
...
<bean id="calculator" class="sprnig.study.CalculatorServiceImpl">
      <property name="valueList">
            <list>
                <value type="java.lang.Integer">10</value>
                <value type="java.lang.Integer">20</value>
            </list>
     </property>
</bean>
...
```

의존관계 자동 설정

- 의존하는 빈객체의 타입이나 이름을 이용하여 의존객체를 자동으로 설정할 수 있는 기능
- autowire속성을 이용
- 자동 설정과 직접 설정의 혼합도 가능

방식	설명
byName	프로퍼티의 이름과 같은 이름을 갖는 빈 객체를 설정
byType	프로퍼티의 타입과 같은 타입을 갖는 빈 객체를 설정
constructor	생성자 파라미터 타입과 같은 타입을 갖는 빈 객체를 생성자에 전달
autodetect	Constructor 방식을 먼저 적용하고, 적용할 수 없을 경우 byType 방식을 적용하여 빈객체를 설정

의존관계 자동 설정 - byName

java

```
...
private OutputService outputter;
public void setOutputter(OutputService outputter) {
        this.outputter = outputter;
}
...
```

applicationContext.xml

```
...
<bean id="greeting" class="autosetting.GreetingServiceImpl"
        autowire="byName"/>

<bean id="outputter" class="autosetting.OutputServiceImplConsole"/>
...
```

의존관계 자동 설정 – byType

java

```
...
private OutputService outputter;
public void setOutputter(OutputService outputter) {
        this.outputter = outputter;
}
...
```

applicationContext.xml

```
...
<bean id="greeting" class="autosetting.GreetingServiceImpl"
        autowire="byType"/>

<bean id="consoleOutput"
        class="autosetting.OutputServiceImplConsole"/>
...
```

의존관계 자동 설정 – constructor

java

```
...
private OutputService outputter;
public GreetingServiceImpl2(OutputService outputter){
    this.outputter = outputter;
}
...
```

applicationContext.xml

```
...
<bean id="greeting2" class="autosetting.GreetingServiceImpl2"
        autowire="constructor" />

<bean id="consoleOutput"
        class="autosetting.OutputServiceImplConsole"/>
...
```

의존관계 자동 설정 – autodetect

java

```
...
private OutputService outputter;
public GreetingServiceImpl2(OutputService outputter){
      this.outputter = outputter;
}
public void setOutputter(OutputService outputter) {
       this.outputter = outputter;
}
```

...

applicationContext.xml

```
...
<bean id="greeting2" class="autosetting.GreetingServiceImpl2"
          autowire="autodetect" />
<bean id="consoleOutput"
          class="autosetting.OutputServiceImplConsole"/>
...
```

빈 객체 범위

- 기본적으로 컨테이너에 한 개의 빈 객체를 생성
- 빈의 범위를 설정할 수 있는 방법을 제공
- scope 속성을 이용

방식	설명
singleton	컨테이너에 한 개의 빈 객체만 생성한다.(기본값)
prototype	빈을 요청할 때마다 빈 객체를 생성한다.
request	HTTP 요청마다 빈 객체를 생성한다.(WebApplicationContext에서만 적용)
session	HTTP 세션마다 빈 객체를 생성한다.(WebApplicationContext에서만 적용)

빈 객체 범위

java

```
...
GreetingService bean = (GreetingService)factory.getBean("greeting");
GreetingService bean2 = (GreetingService)factory.getBean("greeting");

System.out.println(bean == bean2); // true
...
```

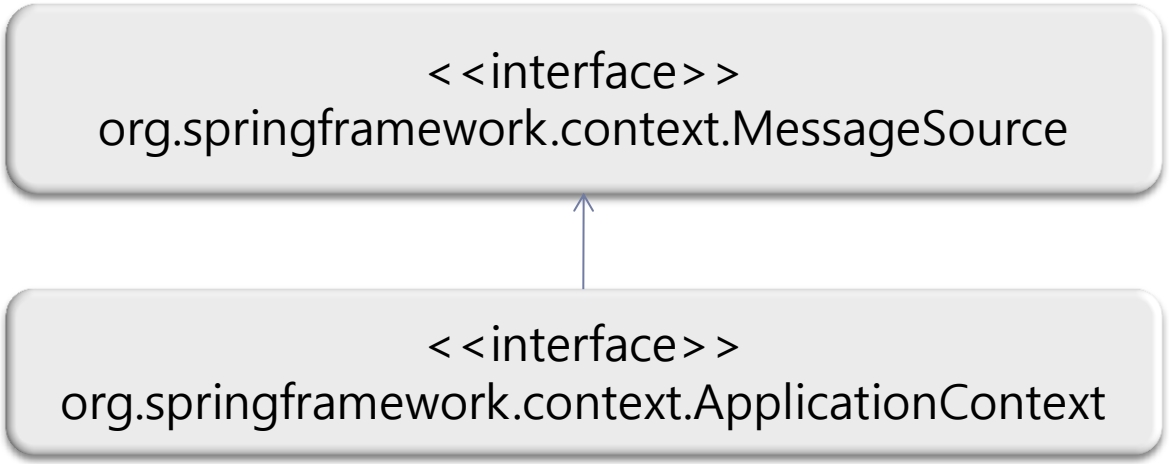
applicationContext.xml

```
...
<bean id="greeting" class="beanscope.GreetingServiceImpl"
        scope="singleton">
    <property name="greeting">
        <value>Hello</value>
    </property>
</bean>
...
```

4. 메시지 및 이벤트 처리

- 메시지 국제화 처리

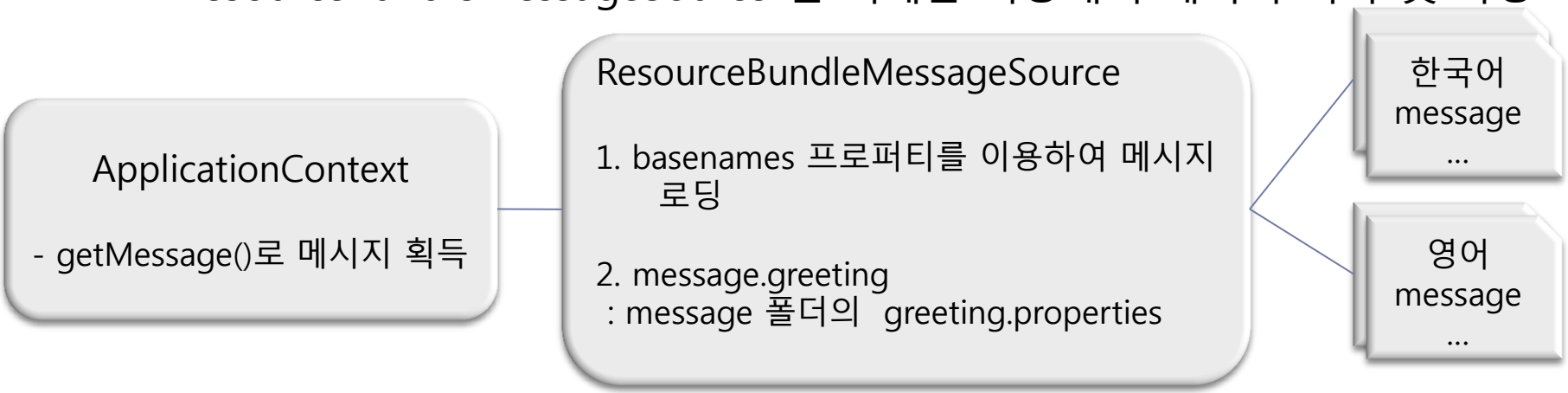
메시지 국제화 처리 – Spring API



- 메시지 국제화를 지원하기 위한 interface
- 지역 및 언어에 따라 알맞은 메시지를 구할 수 있는 메소드 제공

메시지 국제화 처리 – Spring API

- ApplicationContext의 메시지 획득 방법
  - 등록된 빈 객체 중에서 이름이 "messageSource"라는 ResourceBundleMessageSource 빈 객체를 이용해서 메시지 획득 및 사용

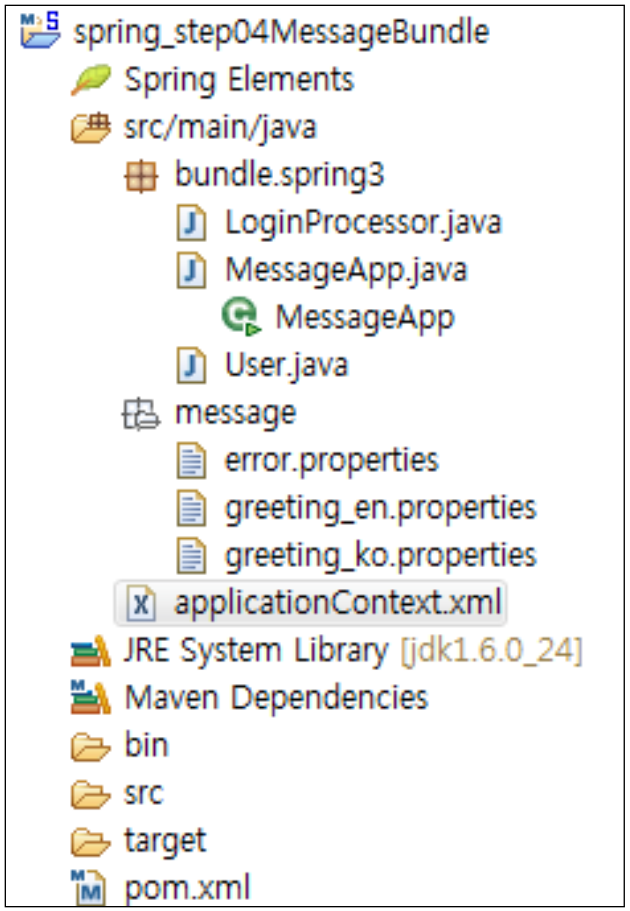



```
8  <bean id="messageSource"
9        class="org.springframework.context.support.ResourceBundleMessageSource">
10   <property name="basenames">
11     <list>
12       <value>message.greeting</value>
13       <value>message.error</value>
14     </list>
15   </property>
16 </bean>
```

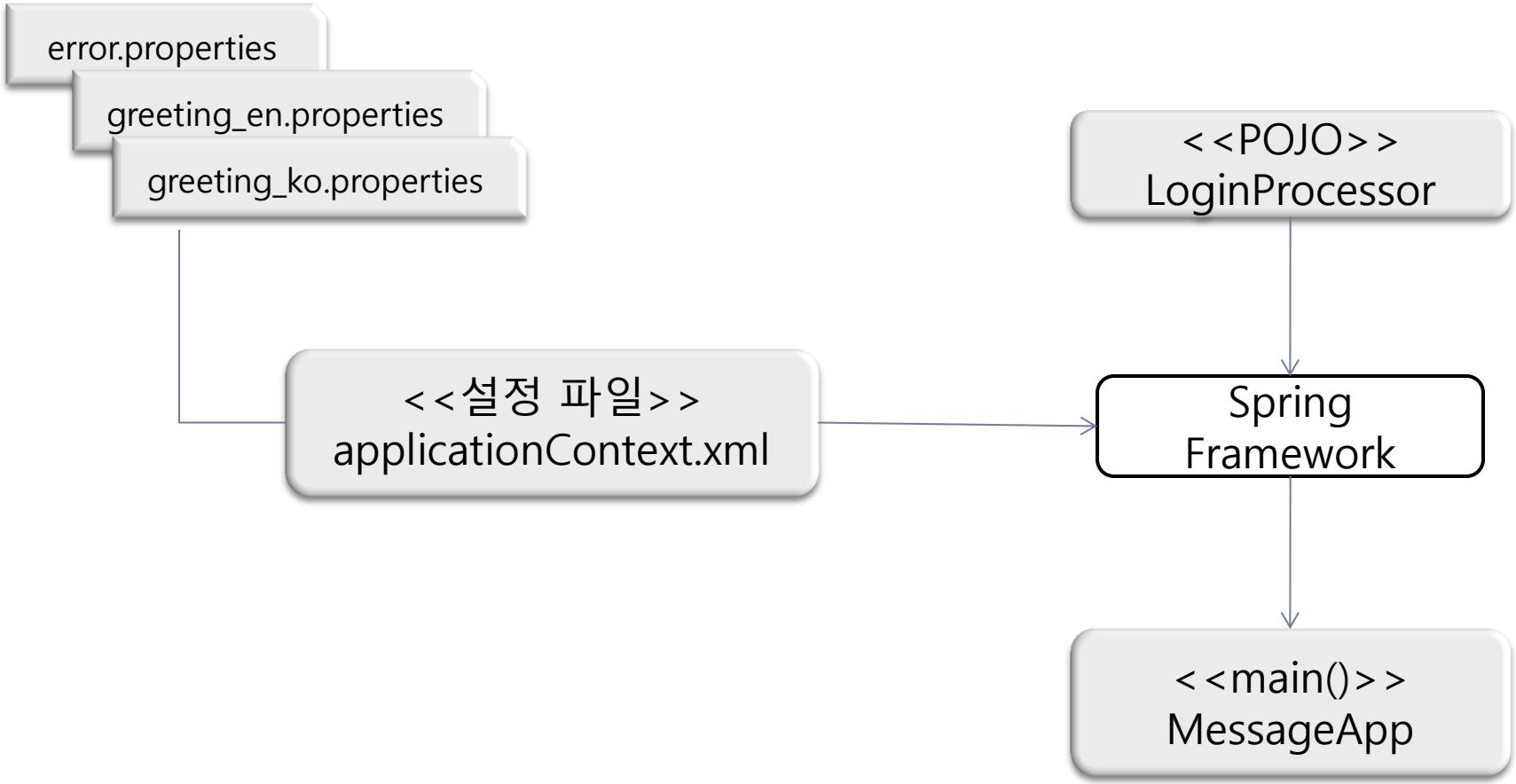
Spring 설정 파일의 message 제공 객체 등록

메시지 국제화 처리 – 빈 객체에서 메시지 이용하기

메시지 국제화 처리 – 실습 구조



메시지 국제화 처리


error.properties
greeting_en.properties
greeting_ko.properties
<<POJO>>
LoginProcessor
<<설정 파일>>
applicationContext.xml
Spring
Framework
<<main()>>
MessageApp

메시지 국제화 처리 – 메시지 사용하는 빈 객체

```java
package bundle.spring3;

import java.util.Locale;

import org.springframework.context.MessageSource;
import org.springframework.context.MessageSourceAware;

public class LoginProcessor implements MessageSourceAware {
    private MessageSource messageSource;
    @Override
    public void setMessageSource(MessageSource messageSource) {
        this.messageSource = messageSource;
    }
    /* 1. 로직
     *     - id/pw의 유효성 체크 메소드
     * 2. 유효성
     *     -   id값이 tester인 경우만 유효하다 간주 */
    public void login(String id, String password) {

        if (!id.equals("tester")) {

            Object[] args = new String[] { id };

            String failMessage = messageSource.getMessage("login.fail", args, Locale.getDefault());

            throw new IllegalArgumentException(failMessage);
        }
    }

}
```

error.properites

login.fail=Member ID {0} is not matching password

메시지 국제화 처리 – spring 설정 파일

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <beans  xmlns="http://www.springframework.org/schema/beans"
4          xmlns:p="http://www.springframework.org/schema/p"
5          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
6          xsi:schemaLocation="http://www.springframework.org/schema/beans
7          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
8
9     <bean   id="messageSource"
10            class="org.springframework.context.support.ResourceBundleMessageSource">
11
12       <property name="basenames">
13          <list>
14             <value>message.greeting</value>
15             <value>message.error</value>
16          </list>
17       </property>
18
19    </bean>
20
21    <bean id="loginProcessor" class="bundle.spring3.LoginProcessor" />
22
23 </beans>
```

메시지 국제화 처리 – message bundle 활용 코드

```
3  import java.util.Locale;
4
5  import org.springframework.context.support.AbstractApplicationContext;
6  import org.springframework.context.support.ClassPathXmlApplicationContext;
7
8  public class MessageApp {
9
0      public static void main(String[] args) {
1
2          AbstractApplicationContext context
3                                  = new ClassPathXmlApplicationContext("applicationContext.xml" );
4
5          Locale locale = Locale.getDefault();
6          String greeting = context.getMessage("greeting", new Object[0], locale);
7
8          System.out.println("Default Locale Greeting: " + greeting);
9
0          Locale englishLocale = Locale.ENGLISH;
1          //Locale englishLocale = Locale.KOREA;
2
3          String englishGreeting = context.getMessage("greeting", new Object[0], englishLocale);
4          System.out.println("English Locale Greeting: " + englishGreeting);
5
6          LoginProcessor loginProcessor = context.getBean("loginProcessor", LoginProcessor.class);
7          try {
8              loginProcessor.login("tester", "1234");
9          } catch (Throwable e) {
0              System.out.println("예외 발생: " + e.getMessage());
1          }
2      }
3  }
```

Locale 설정의 차이에 따른 차이점 파악하기

예외 발생시 error.properties 내용 출력

5. 애노테이션 기반 설정 및 자바 코드 기반 설정

- 애노테이션 기반 설정
- 빈 객체 스캔
- 자바 코드 기반 설정

애노테이션 기반 설정

- 애노테이션이란?
  - JDK5 버전부터 추가된 것으로 메타데이터를 XML등의 문서에 설정하는 것이 아니라 소스 코드에 "@애노테이션"의 형태로 표현하며 클래스, 필드, 메소드의 선언부에 적용 할 수 있는 특정 기능이 부여된 표현법

- 애노테이션 사용 이유
  - 프레임워크들이 활성화 되고 애플리케이션 규모가 커질수록 XML 환경 설정은 복잡해지는데, 이러한 어려움을 개선시키기 위하여 자바 파일에 애노테이션을 적용해서 코드를 작성함으로써 개발자가 설정 파일에 작업하게 될 때 발생시키는 오류의 발생 빈도를 낮춰주기도 함
  - 애노테이션을 사용하면 소스 코드에 메타데이터를 보관할 수 있으며 컴파일 타임의 체크뿐 아니라 애노테이션 API를 사용하여 코드의 가독성을 높일 수도 있음

애노테이션 기반 설정

- 애노테이션 문법
  - @애노테이션"[앳애노테이션]"의 형태로 표현하며 클래스, 필드, 메소드의 선언부에 적용 할 수 있고 특정 기능이 부여된 표현법입니다

```
 9  @Component
10  public class PersonServiceImpl implements PersonService{
11      private String message;
12
13      @Resource(name="person1")
14      private Person person1;
15
16      @Resource(name="person2")
17      private Person person2;
```

애노테이션 기반 설정

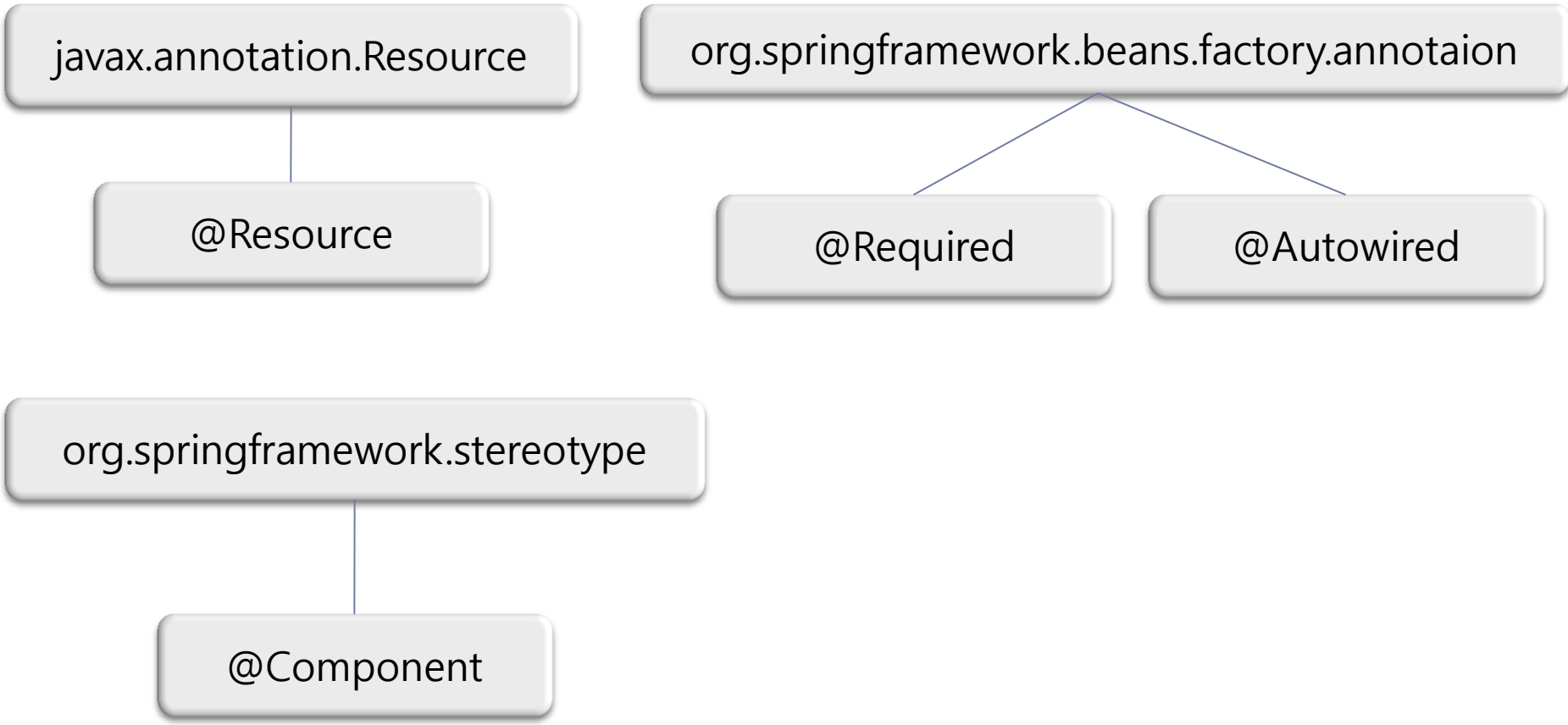
- 주석과 애노테이션의 차이점
  - //과, @ 은 개발자가 보기에는 똑같이 실행되지 않는 코드 같지만 실제로 //은 컴파일러가 실행은 안 해도 @은 컴파일러가 실행되기 전에 애노테이션으로 설정한 내용대로 코드가 작성되었는지 확인하기 위해 실행을 함
  - "@Override" 애노테이션이 선언된 메소드를 Eclipse 툴에서 재정의 할 경우 재정의 문법에 위배된 코드로 개발이 되면 소스 화면에 문법 오류 발생이라는 표현을 해 주기 때문에 개발자의 실수도 쉽게 알 수 있으며, 빠른 시간에 문제점에 대한 해결을 하게 됨
  - 개발자가 환경 설정에서의 실수를 했을 경우에도 수정해주는 역할을 하기도 하고, 메타 정보를 선언하게 되는 XML 설정 파일이 점점 복잡해지지 않도록 자바 코드 안에서 설정하므로 개발자의 도움말이 되는 역할을 함

Spring DI[디아이]와 주요 애노테이션

- Spring2.5 버전 부터는 애노테이션을 이용하여 빈과 관련된 정보를 설정할 수 있게 되었으며, 복잡한 XML 문서 생성과 관리에 따른 수고를 덜어주어 개발 속도를 향상 시킬 수 있음

애노테이션	설 명
@Required	setter주입방식을 이용한 애노테이션
@Autowired	타입을 기준으로(byType) 빈을 찾아 주입하는 애노테이션
@Resource	이름을 기준으로(byname) 빈을 찾아 주입하는 애노테이션
@Component	빈 스캐닝 기능을 이용한 애노테이션

주요 애노테이션과 API 관계



애노테이션 기반 설정 - @Required

@Required

- 기능
  - Spring 빈의 멤버 변수인 필수 프로퍼티를 명시할 때 사용
- 적용 문법
  - 자바 코드 - @Required 선언
  - 설정 파일 - <context:annotation-config/>
  - 적용 위치 - 해당 setter 메소드 선언구에 설정

애노테이션 기반 설정 - @Required

자바 코드

```
 9 import org.springframework.beans.factory.annotation.Required;
10 public class PersonServiceImpl implements PersonService{
11     private Person person;
12
13     //... 중략 ...
14
15     @Required
16     public void setPerson(Person person) {
17         this.person = person;
18     }
```

설정 파일

```
11     <context:annotation-config/>
12
13     <bean name="personServiceImpl" class="di.PersonServiceImpl">
14
15         <property name="person" ref="person"/>
16
17     </bean>
18
19     <bean name="person" class="di.Person">
20         <property name="name" value="애노테이션맨"/>
21         <property name="age" value="3"/>
22     </bean>
```

애노테이션 기반 설정 - @Autowired

@Autowired

- 기능
  - Spring 빈의 의존 관계를 자동 설정할 때 사용
  - 타입을 이용한 프로퍼티 자동 설정 기능 제공

- 적용 문법
  - 자바 코드 - @Autowired 선언
  - 설정 파일 - <context:annotation-config/>
  - 적용 위치 – 생성자, 필드, 메소드 선언구에 설정

애노테이션 기반 설정 - @Autowired

자바 코드

```
11 import org.springframework.beans.factory.annotation.Autowired;
12 public class PersonServiceImpl implements PersonService{
13     private Person person;
14
15     //... 중략 ...
16
17     @Autowired
18     public void setPerson(Person person) {
19         this.person = person;
20     }
21
```

설정 파일

```
26     <context:annotation-config/>
27
28     <bean name="personServiceImpl" class="di.PersonServiceImpl"/>
29
30     <bean name="person" class="di.Person">
31         <property name="name" value="애노테이션맨"/>
32         <property name="age" value="3"/>
33     </bean>
34
35 </beans>
36
```

애노테이션 기반 설정 - @Resource

@Resource

- 기능
  - JDK6 부터 제공
  - 애플리케이션이 필요로 하는 자원을 자동 연결해주는 기능 제공
  - Spring에선 빈 객체를 전달할때 사용

- 적용 문법
  - 자바 코드 - @Resource 선언
  - 설정 파일 - <context:annotation-config/>
  - 적용 위치 - 해당 setter 메소드 선언구에 설정

애노테이션 기반 설정 - @Resource

자바 코드

```
13 import javax.annotation.Resource;
14
15 public class PersonServiceImpl implements PersonService{
16
17     @Resource(name="person")
18     private Person person;
19
20     public PersonServiceImpl() {}
21
22     public PersonServiceImpl(Person person) {
23         this.person = person;
24     }
```

설정 파일

```
39     <context:annotation-config/>
40
41     <bean name="personServiceImpl" class="di.PersonServiceImpl"/>
42
43     <bean name="person" class="di.Person">
44         <property name="name" value="애노테이션맨"/>
45         <property name="age" value="3"/>
46     </bean>
47
48
```

빈 객체 스캔

- XML 문서와 같이 한곳에 명시적으로 선언하지 않고 빈으로 사용될 클래스에 특별한 애노테이션을 부여해주면 특정 애노테이션이 붙은 클래스를 자동으로 찾아서 빈으로 등록해주는 방식
  - 빈 스캐닝 – 빈 클래스에 애노테이션을 선언해 둘 경우 애노테이션이 붙은 클래스들을 자동으로 찾아서 빈으로 등록해주는 방식의 미
  - 빈 스캐너(bean scanner) – 스캐닝 작업을 담당하는 객체

- XML 설정 파일에 다양한 빈 정보를 추가하지 않고도 특정한 클래스를 빈으로 등록해 줌

- 빈 스캐너의 작동 원리
  - 지정된 클래스패스 아래에 있는 모든 패키지의 클래스를 대상으로 빈 스캐너가 자바 코드에 선언된 애노테이션 기능에 맞게 자동으로 선별해 내서 Spring 빈으로 설정

빈 객체 스캔

@Component

- 기능
  - 클래스 패스에 위치한 클래스를 검색하여 @Component 애노테이션이 붙은 클래스를 자동으로 빈으로 등록하는 기능
  - XML 설정 파일에 다양한 빈 정보를 추가하지 않고 특정한 클래스를 빈으로 등록 가능

- 명명규칙
  - 검색된 클래스를 빈으로 등록할 경우 클래스의 이름을 빈의 이름으로 사용(첫글자는 소문자로 변경)

```
@Component
public class SpringBean{
  …
}
```

```
…
SpringBean sb =
(SpringBean)context.getBean("springBean");
```

빈 객체 스캔

@Component

- 적용 문법
  - 자바 코드 - @Component 선언
  - 설정 파일

    - <context:component-scan base-package="package명"/>

    : 클래스를 검색할 패키지 지정
  - 적용 위치 – 클래스 선언구 부분에 위치

빈 객체 스캔

자바 코드

```java
import javax.annotation.Resource;
import org.springframework.stereotype.Component;

@Component
public class PersonServiceImpl implements PersonService{

    @Resource(name="person")
    private Person person;
```

설정 파일

```xml
    <context:component-scan base-package="bean.scan"/>
```

자바 코드 기반 설정

- **Spring 3 버전 부터 추가된 기능**
- XML 설정 없이도 자바 코드를 이용해서 생성할 빈 객체와 각 빈간의 의존 관계 설정
- 하나의 클래스 안에 여러 개의 빈을 정의 할 수도 있고, 클래스 자체가 자동인식 빈의 대상이 되기 때문에 XML에 명시적으로 등록하지 않아도 됨

- **사용되는 애노테이션**
  - **@Configuration**
    - **빈 설정 메타 정보를 담고 있는 자바 코드에 선언**
  - **@Bean**
    - **클래스 내의 메소드를 정의 할 수 있음**

자바 코드 기반 설정

- Spring 컨테이너에 새로운 빈 객체 제공

@Configuration

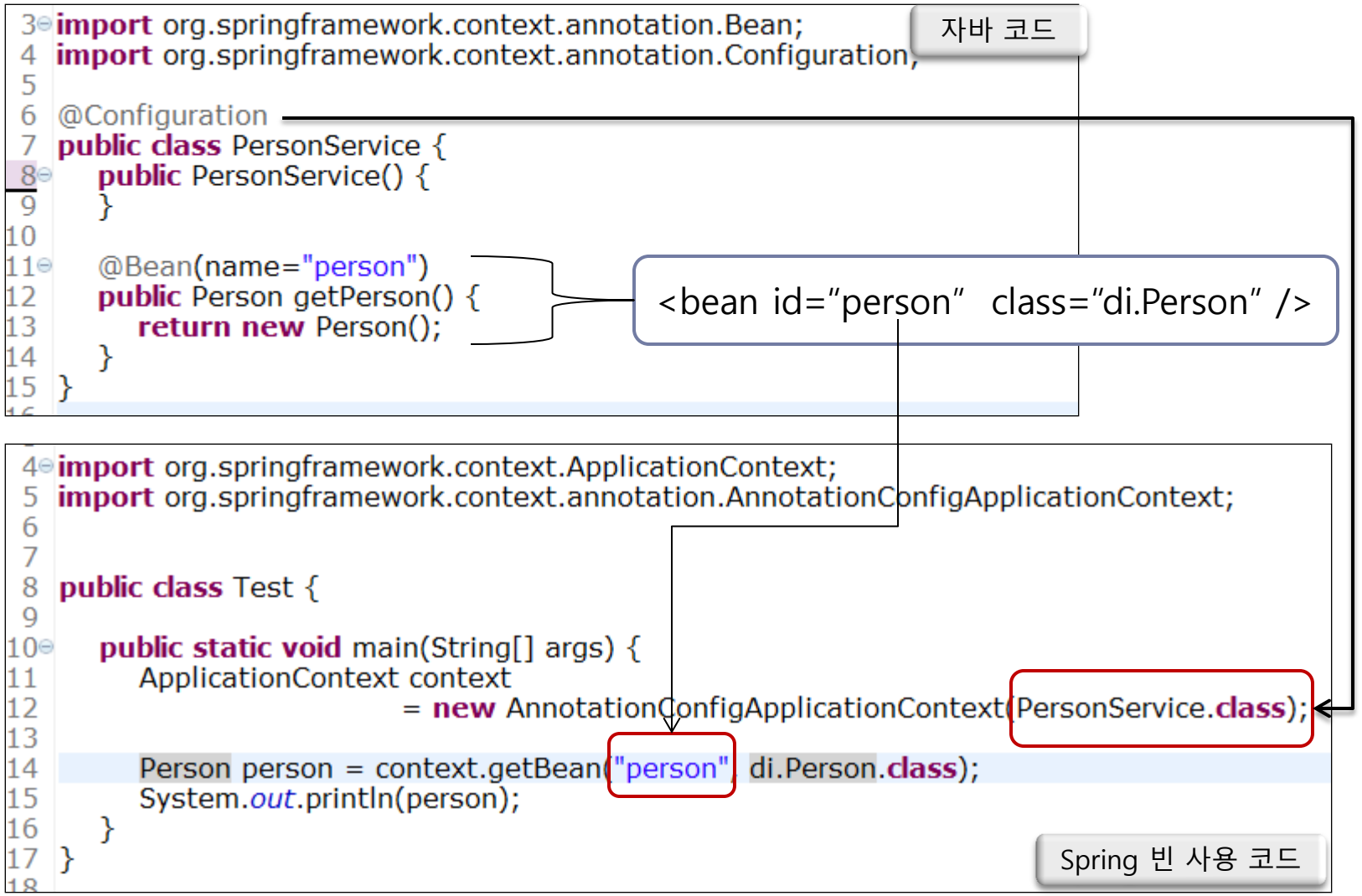
빈 설정 메타 정보를담고 있는 클래스가 됨

@Bean

1. 새로운 빈 객체를 제공할 때 사용
2. 적용된 메서드의 이름을 빈의 식별값으로 사용
3. name 속성을 사용하여 새로운 빈 이름 적용 가능
4. @Scope(value="prototype") 으로 범위 설정 가능

ApplicationContext context = new AnnotationConfigApplicationContext(**클래스명.class**)

자바 코드 기반 설정




```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class PersonService {
    public PersonService() {
    }

    @Bean(name="person")
    public Person getPerson() {
        return new Person();
    }
}
```

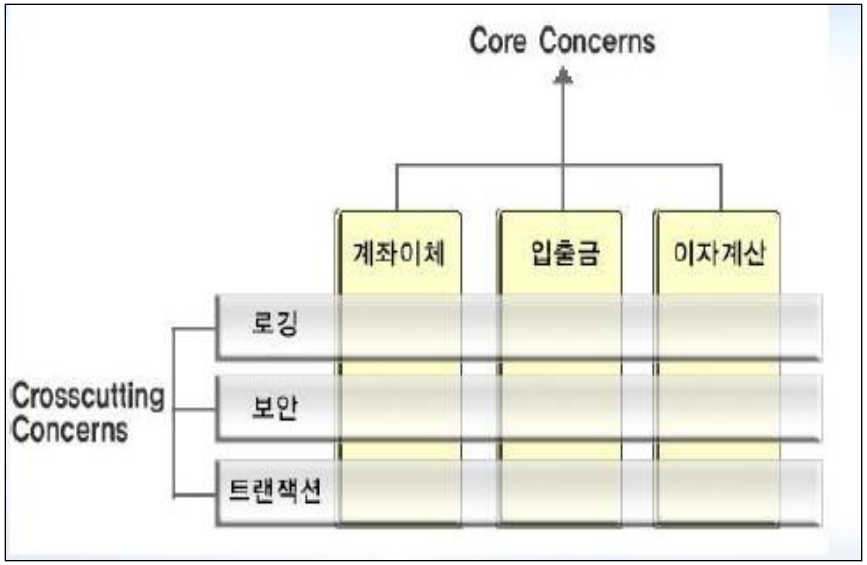
```java
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;


public class Test {

    public static void main(String[] args) {
        ApplicationContext context
                        = new AnnotationConfigApplicationContext(PersonService.class);

        Person person = context.getBean("person", di.Person.class);
        System.out.println(person);
    }
}
```

6. Aspect Oriented Programming

- AOP
  - 장점
  - AOP 주요 용어
  - Proxy
  - Weaving
- Spring에서의 AOP
- XML 기반의 POJO 클래스를 이용한 AOP 구현
- 애노테이션 기반의 AOP 구현
- 표현식

AOP[Aspect Oriented Programming] 개요

- 그림 설명
  - 핵심 비즈니스 로직
    - 계좌이체, 입출금, 이자계산
  - 핵심 로직에 적용되는 공통 로직
    - 로깅, 보안, 트랜잭션



[그림 출처] http://www.egovframe.go.kr/Egovcmm.jsp

AOP 개요

- 용어 정리
  - Concern
    - 애플리케이션을 개발하기 위하여 관심을 가지고 구현 해야 하는 각각의 기능들을 관심 사항(Concern)이라 함
  - core concern
    - 핵심 관심 사항
    - 해당 애플리케이션이 제공하는 핵심이 되는 비즈니스 로직 의미
  - cross-cutting concern
    - 횡단[공통] 관심 사항
    - 하나의 영역에서만 활용되는 고유한 관심 사항(Concern)이 아니라, 여러 클래스 혹은 여러 계층의 애플리케이션 전반에 걸쳐서 공통적으로 필요로 하는 기능들 의미

AOP 개요

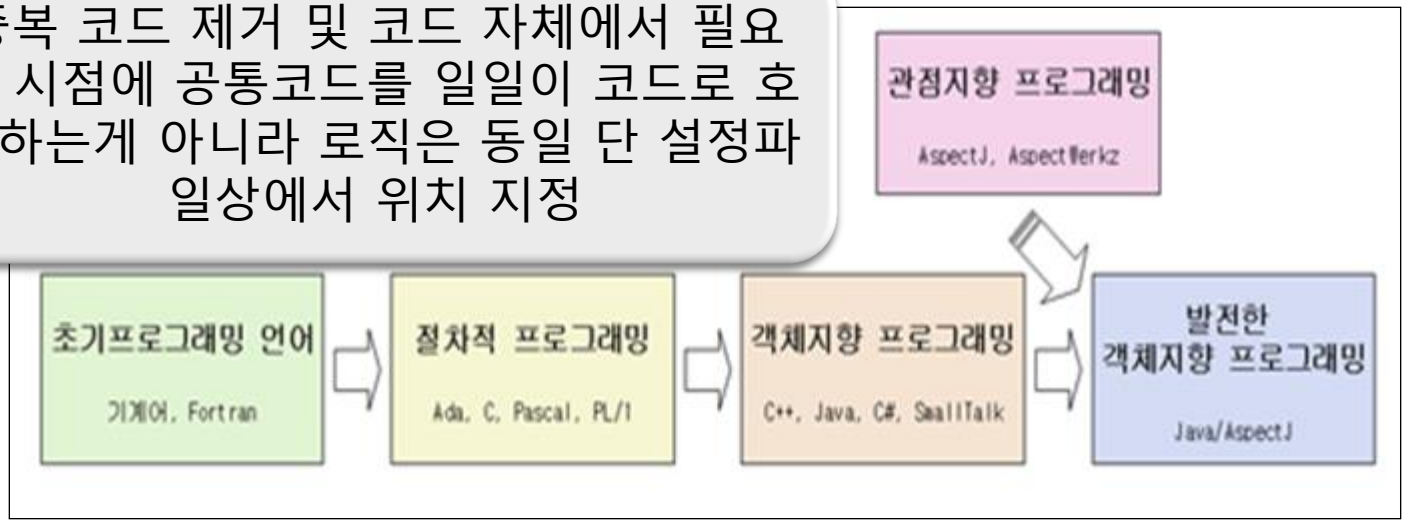
Aspect(애스펙트)란?

보안과 로깅 모듈처럼 그 자체로 애플리케이션의 핵심 기능을 다루고 있지는 않지만, 애플리케이션을 구성하는 중요한 요소이고, 핵심 기능에 적용되어야만 의미를 갖는 특별한 모듈을 의미

AOP(Aspect-Oriented Programming) 란?

- AOP에서 중요한 개념
  - 「횡단 관심 사항의 분리(Separation of Cross-Cutting Concern)」
  - 로깅, 보안, 트랜잭션처럼 비즈니스 로직의 여러 부분에 공통적으로 사용되는 부가적인 횡단 관심 사항들을 비즈니스 로직에서 분리해서 Aspect 라는 별도의 모듈로 설계, 개발하는 방법 의미
  - 문제를 바라보는 관점을 기준으로 프로그래밍하는 기법

중복 코드 제거 및 코드 자체에서 필요한 시점에 공통코드를 일일이 코드로 호출하는게 아니라 로직은 동일 단 설정파일상에서 위치 지정



[그림] 프로그래밍 패러다임의 변환

AOP 장점 – AOP를 적용하지 않은 OOP

- AOP를 적용하지 않은 OOP 개발시에 발생될 수 있는 문제점

- 중복되고 지저분한 코드
  - 핵심 기능의 코드 사이 사이에 적용되는 횡단 관심 로직들로 인해 코드의 양도 많아지고 지저분해 지므로 가독성 또한 떨어짐

- 개발 생산성 및 재사용성의 저하
  - 핵심 로직의 코드 개발에만 집중할 수 없고, 전체적으로 많은 부분에 영향을 미치는 횡단 관심 로직도 적용해야 하기 때문에 개발 생산성 및 코드의 재 사용성도 떨어짐

AOP 장점 – AOP를 적용한 OOP

- 해결책
  - OOP의 단점을 극복하고자 핵심 관심 로직과 횡단 관심 로직을 분리해서 각 모듈로 개발한 이후에 핵심 로직 코드의 수정 없이도 횡단 관심 로직을 적용하도록 하는 것이 AOP의 기술
  - 이처럼 AOP는 OOP의 대체용이 아닌 OOP와 같이 적용하면서 OOP의 단점을 보완해 줄 수 있는 언어

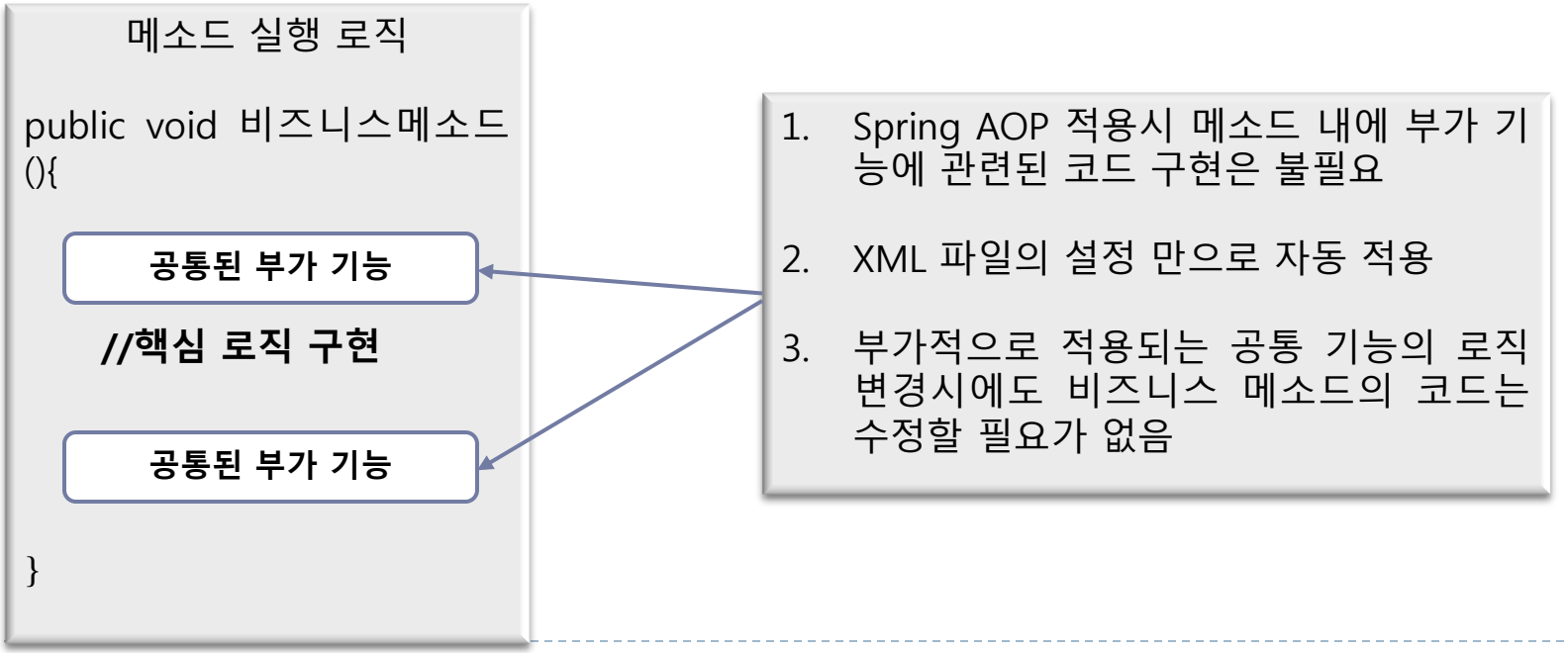
- 애플리케이션을 다양한 측면에서 독립적으로 모델링하고, 설계하고, 개발할 수 있도록 해주며 핵심 관점 및 횡단 관심 관점에서 해당 부분에만 집중해서 설계하고 개발 할 수 있기 때문에 애플리케이션을 특정한 관점을 기준으로 바라 볼 수 있게 해 준다는 의미에서 AOP를 '관점지향 프로그래밍'이라고 함

AOP 장점 – AOP를 적용한 OOP

- AOP를 적용하게 되면 개발자들은 핵심 관심 사항 코드에도 비즈니스 기능 처리에 대한 구현만 할 수 있음
- 필요에 따라서 횡단관심 사항의 로직들을 언제든지 쉽게 추가, 삭제 할 수 있음
- AOP 기술을 프로젝트에 적용하기 위해서는 Spring 또는 AspectJ와 같이 AOP를 지원하는 프레임워크를 주로 사용

AOP 장점 – AOP를 적용한 OOP

- AOP 적용
  - 개발자들은 횡단 관심 모듈을 각각 독립된 모듈로 중복 없이 작성하고 이를 적합한 위치에 XML 설정 파일에 설정하는 등의 작업을 통해 핵심 관심 모듈과 결합시키기 때문에 서로 독립성을 가진 다차원의 모듈로 작성 할 수 있음
  - 재사용성과 보수성을 높일수 있음

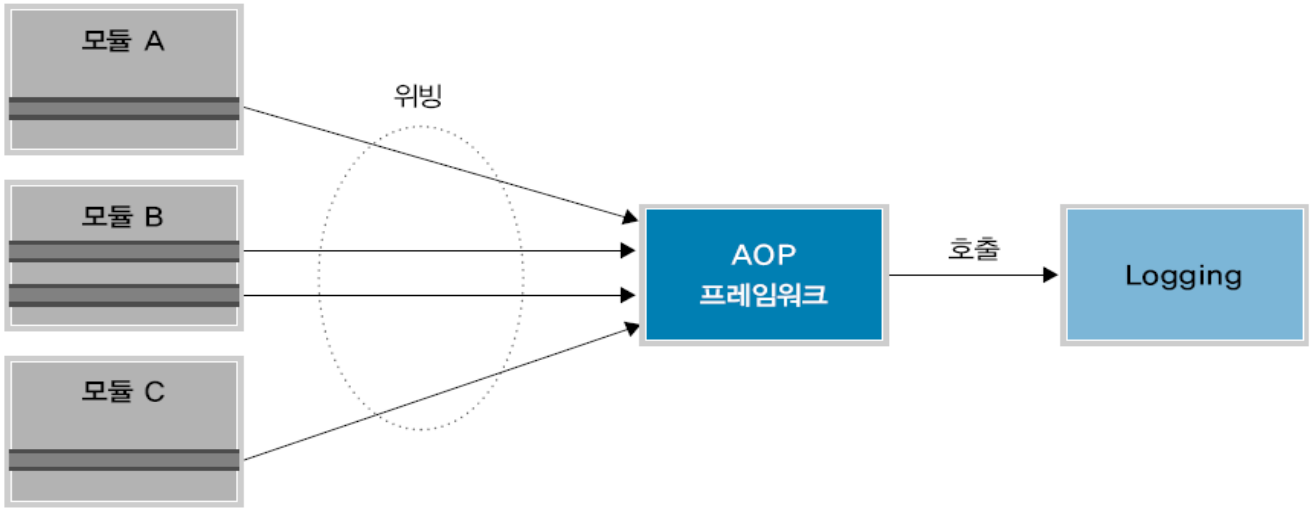


횡단 관점의 분리를 위한 개발 방법 - AOP

- AOP에서는 핵심 로직을 구현한 코드에서 공통 기능을 직접적으로 호출하지 않음
- AOP에서는 분리한 공통 기능의 호출까지도 관점으로 다룸
- 이러한 각 모듈로 산재한 관점을 횡단 관점이라 부름
- AOP에서는 이러한 횡단 관점까지 분리함으로써 각 모듈로부터 관점에 관한 코드를 완전히 제거하는 것을 목표로 함

횡단 관점의 분리를 위한 개발 방법 - AOP

- AOP에서는 핵심 로직을 구현한 코드를 컴파일 하거나, 컴파일된 클래스를 로딩하거나 또는 로딩한 클래스의 객체를 생성할때 핵심 로직 구현 코드 안에 공통기능이 삽입



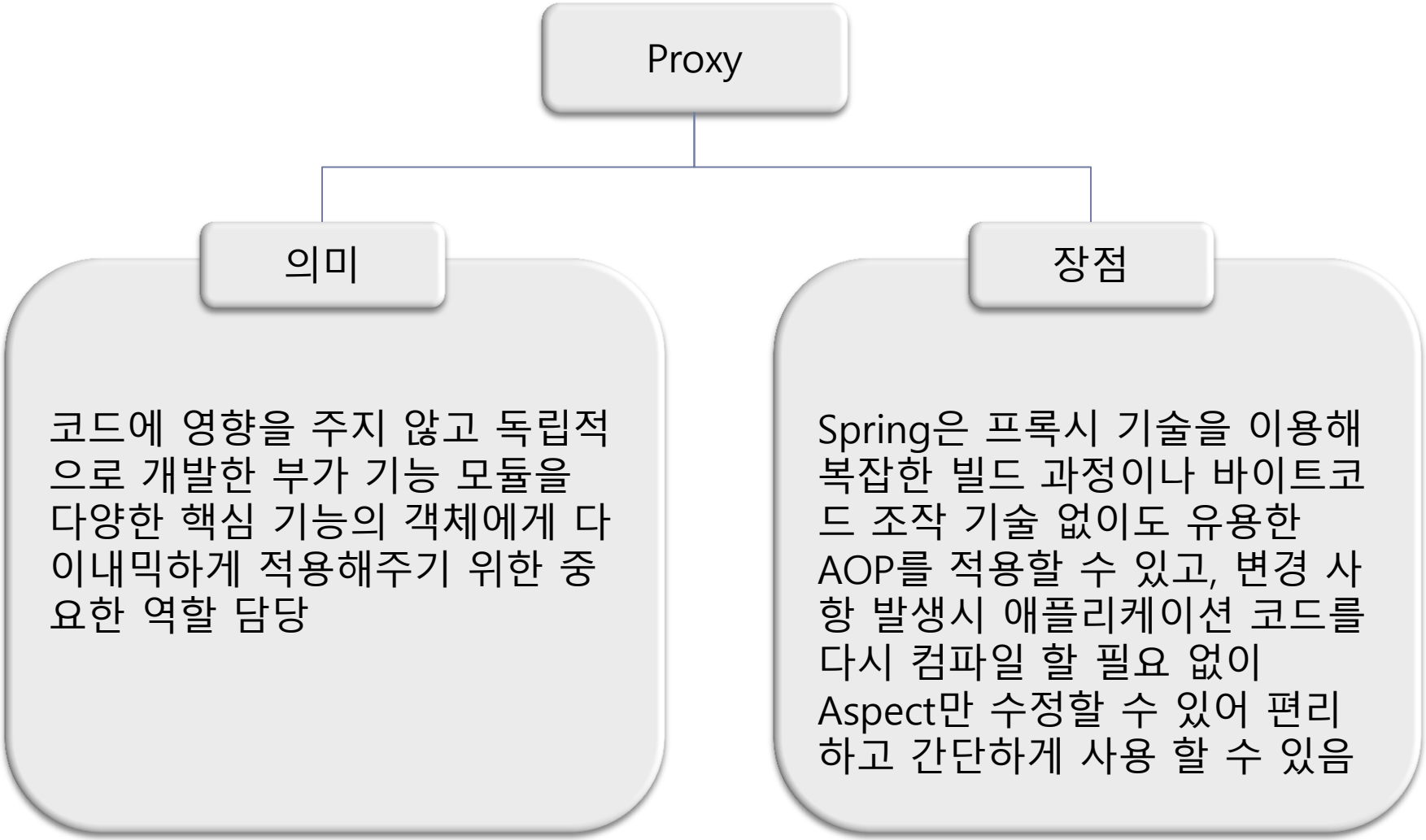
AOP 주요 용어

용어	설명
Target	핵심 로직을 구현하는 클래스 공통 관심 사항을 적용 받게 되는 대상으로 어드바이스가 적용되는 객체
Aspect	여러 객체에 공통으로 적용되는 공통 관심 사항
Advice	조인 포인트에 삽입되어 동작할 수 있는 공통 관심 사항의 코드 *동작시점 : Spring에서는 조인포인트 실행 전, 후로 before, after, after returning, after throwing, around로 구분
joinpoint	「클래스의 인스턴스 생성 시점」,「메소드 호출 시점」 및「예외 발생 시점」과 같이 애플리케이션을 실행할 때 특정 작업이 시작되는 시점으로 Advice를 적용 가능한 지점 즉 어드바이스가 적용될 수 있는 위치
pointcut	여러 개의 Joinpoint를 하나로 결합한(묶은) 것
Advisor	Advice와 Pointcut를 하나로 묶어 취급한 것
weaving	공통 관심 사항의 코드인 Advice를 핵심 관심 사항의 로직에 적용 하는 것을 의미

Weaving방식- Advice를 위빙하는 방식 3가지

용어	설명
컴파일시 위빙하기	AOP가 적용된 새로운 클래스 파일이 생성됨 AspecyJ에서 사용 하는 방식
클래스 로딩시 위빙하기	로딩한 바이트 코드를 AOP가 변경하여 사용 AOP 라이브러리는 JVM이 클래스를 로딩할때 클래스 정보를 변경할 수 있는 에이전트 제공 (원본 클래스 파일 변경 없이 클래스를 로딩할 때 JVM이 변경된 바이트 코드를 사용하도록 함)
런타임시 위빙하기	소스 코드나 클래스 정보 자체를 변경하지 않음 프록시를 이용하여 AOP적용 프록시 기반의 AOP는 핵심 로직을 구현한 프록시를 통해서 핵심 로직을 구현한 객체에 access 프록시 기반의 제한사항 : 메소드 호출할 경우에만 Advice를 적용

Proxy


Proxy
의미
코드에 영향을 주지 않고 독립적으로 개발한 부가 기능 모듈을 다양한 핵심 기능의 객체에게 다이내믹하게 적용해주기 위한 중요한 역할 담당
장점
Spring은 프록시 기술을 이용해 복잡한 빌드 과정이나 바이트코드 조작 기술 없이도 유용한 AOP를 적용할 수 있고, 변경 사항 발생시 애플리케이션 코드를 다시 컴파일 할 필요 없이 Aspect만 수정할 수 있어 편리하고 간단하게 사용 할 수 있음

Spring에서의 AOP

- Spring에서는 자체적으로 런타임시에 위빙하는 "프록시 기반의 AOP"를 지원
- 프록시 기반의 AOP는 메소드 호출 조인포인트만 지원
- Spring에서 어떤 대상 객체에 대해 AOP를 적용할 지의 여부는 설정 파일을 통해서 지정
- Spring은 설정 정보를 이용하여 런타임에 대상 객체에 대한 프록시 객체를 생성 따라서, 대상 객체를 직접 접근하는 것이 아니라 프록시를 통한 간접 접근을 하게 됨

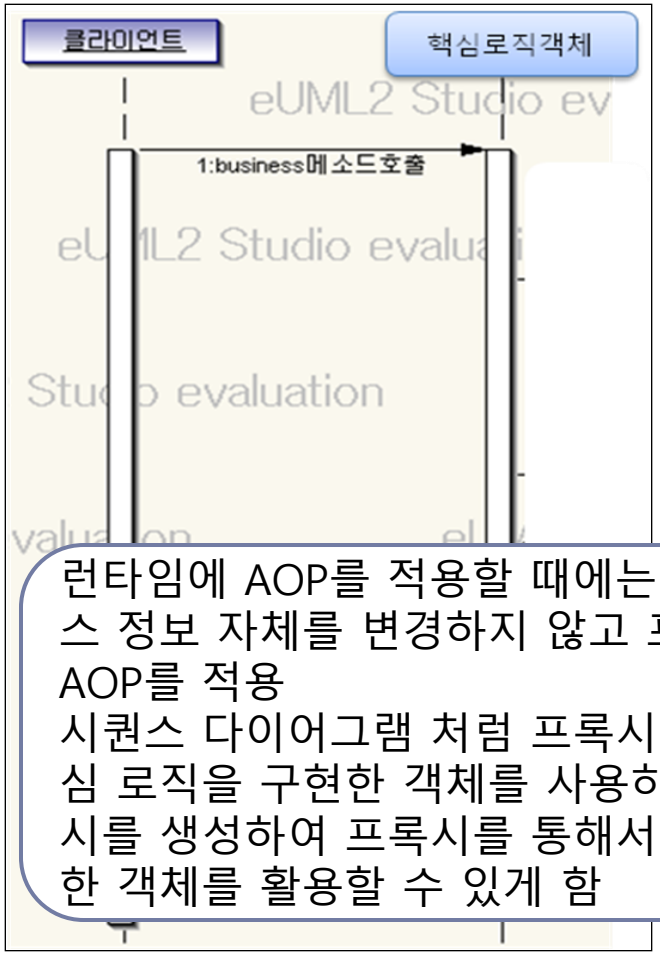
Spring에서의 AOP

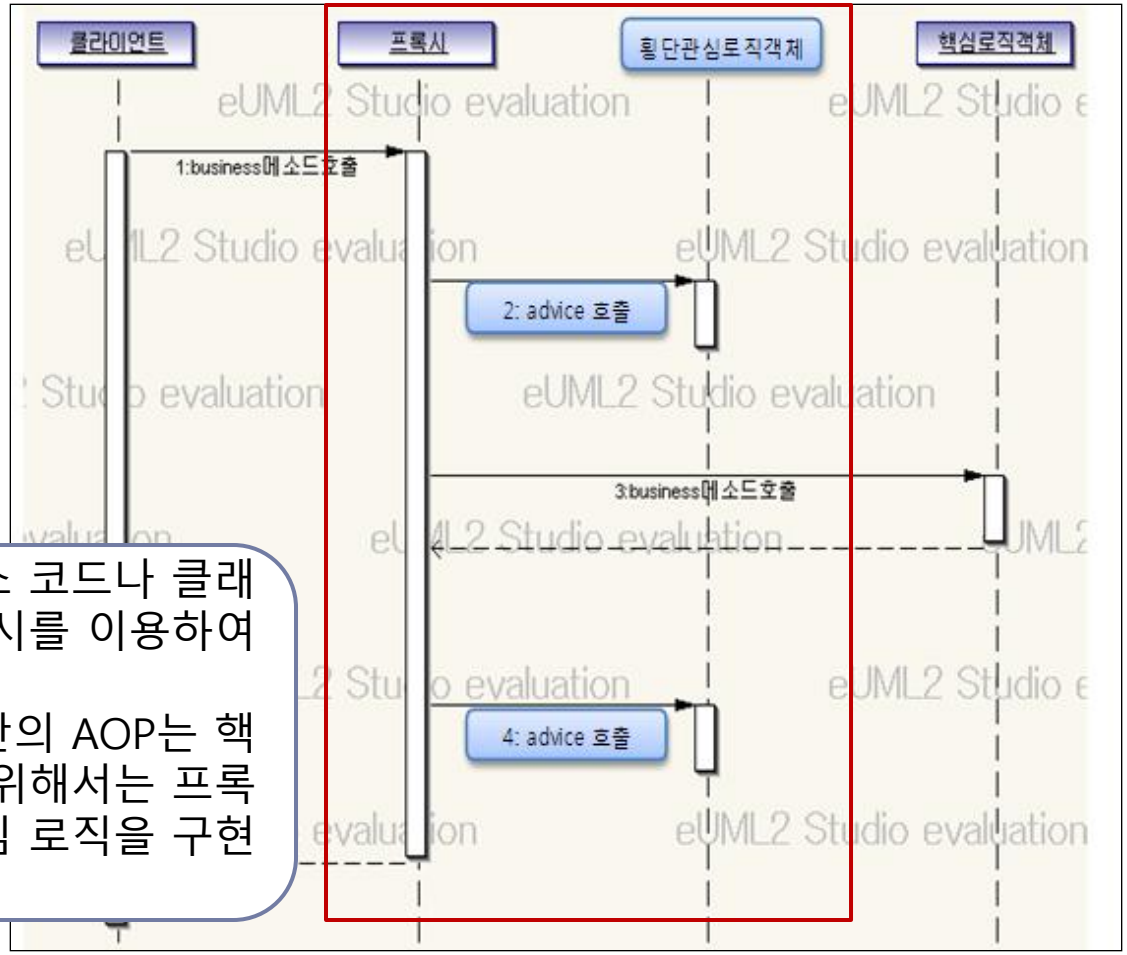
- Spring은 완전한 AOP 기능을 제공하는 것이 목적이 아니라 Enterprise Application을 구현하는데 필요한 기능을 제공하는 것을 목적으로 하고 있음

- 필드값 변경 등 다양한 조인포인트를 이용하려면 AspectJ와 같은 다른 AOP솔루션을 이용해야 함

프록시 기반의 AOP 적용 process

- 코드상에서의 호출 process
- Spring에서의 실제 내부적인 process





런타임에 AOP를 적용할 때에는 소스 코드나 클래스 정보 자체를 변경하지 않고 프록시를 이용하여 AOP를 적용
시퀀스 다이어그램 처럼 프록시 기반의 AOP는 핵심 로직을 구현한 객체를 사용하기 위해서는 프록시를 생성하여 프록시를 통해서 핵심 로직을 구현한 객체를 활용할 수 있게 함

Spring AOP 프록시의 종류

- 핵심 로직 구현 방법에 따른 proxy 종류

핵심 로직의 클래스 구현시 interface가 정의되어 있는 경우

JDK 동적 프록시 사용

JDK 프록시는 오직 인터페이스의 프록시만을 생성
대상 객체가 인터페이스를 구현하고 있다면,
Spring은 자바 리플렉션 API가 제공하는 java.lang.reflect.Proxy 를 이용하여 프록시 객체를 생성

interface가 정의 되어 있지 않은 핵심 로직의 클래스인 경우

CGlib 사용

바이트코드 생성 라이브러리를 이용하여 동적으로 각각의 클래스에 대한 프록시 객체를 생성
이미 생성한 클래스가 있을 때는 그것을 재사용

Spring에서의 AOP

- AOP 구현을 위한 개발 방법

XML 스키마 기반의
POJO 클래스를 이용한 AOP 구현

애노테이션 기반의 AOP 구현

Spring에서의 AOP – Advice 개요

- Advice란?
  - 조인 포인트에 삽입되어 동작할 수 있는 공통 관심 사항의 코드
  - 동작시점 : Spring에서는 조인포인트 실행 전, 후로 before, after, after returning, after throwing, around로 구분

Spring에서의 AOP - Advice 종류

Advice 종 류	XML 스키마 기반의 POJO 클래스를 이용	@Aspect 애노테이션 기반	설 명
Before	<aop:before>	@Before	target 객체의 메소드 호출시 호출 전에 실행
AfterReturning	<aop:after-returning>	@AfterRetuning	target 객체의 메소드가 예외 없이 실행된 후 호출
AfterThrowing	<aop:after-throwing>	@AfterThrowing	target 객체의 메소드가 실행하는중 예외가 발생한 경우에 실행
After	<aop:after>	@After	target 객체의 메소드를 정상 또는 예외 발생 유무와 상관없이 실행 try의 finally와 흡사
Around	<aop:around>	@Around	target 객체의 메소드 실행 전, 후 또는 예외 발생 시점에 모두 실행해야 할 로직을 담아야 할 경우

XML 기반의 POJO 클래스를 이용한 AOP 구현

- Spring API를 사용하지 않은 POJO 클래스를 이용하여 어드바이스를 개발하고 적용할 수 있는 방법이 추가

- XML 확장을 통해 설정 파일도 보다 쉽게 작성 할 수 있음

XML 기반의 POJO 클래스를 이용한 AOP 구현

개발 단계

Advice 클래스를 구현

aop 네임스페이스가 있는 XML 설정 파일에 Advice 클래스를 빈으로 등록

<aop:config> 태그를 이용해서 Advice를 어떤 포인트컷에 적용할지 지정

XML 기반의 POJO 클래스를 이용한 AOP 구현

Advice 클래스를 구현

```
4  import org.aspectj.lang.ProceedingJoinPoint;
5
6  public class CommonAdvice {
7
8     /* 1. around advice
9      * 2. 실행 로직  - Advice가 적용될 target 객체를 호출하기 전화 후를 구해서 target 객체의
10    *                메소드 호출 실행 시간 출력*/
11    public void around(ProceedingJoinPoint point) throws Throwable{
12       String methodName = point.getSignature().getName();
13       String targetName = point.getTarget().getClass().getName();
14       System.out.println("target 클래스명 : " + targetName + " 및 메소드명 : " + methodName);
15
16       long startTime = System.nanoTime();
17       System.out.println("[Log]핵심 로직 메소드 호출전 --> "+ methodName+" time check start");
18
19       Object obj = point.proceed();
20
21       long endTime = System.nanoTime();
22       System.out.println("[Log]핵심 로직 메소드 호출 후 --> " + methodName+" time check end");
23       System.out.println("[Log] "+ methodName+" 실행 소요 시간 "+(endTime - startTime)+"ns");
24
25       System.out.println(targetName + " 메서드 실행후 리턴된 데이터 : " + obj);
26    }
27
```

XML 기반의 POJO 클래스를 이용한 AOP 구현

aop 네임스페이스가 있는 XML 설정 파일에 Advice 클래스를 빈으로 등록

<aop:config> 태그를 이용해서 Advice를 어떤 포인트컷에 적용할지 지정

설정 파일에 aop 네임스페이스 및 네임스페이스와 관련된 스키마 추가

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4
5     xmlns:aop="http://www.springframework.org/schema/aop"
6
7     xsi:schemaLocation="http://www.springframework.org/schema/beans
8        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
9
10       http://www.springframework.org/schema/aop
11       http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
12
13    <!-- 중략 -->
14
15 </beans>
16
```

XML 기반의 POJO 클래스를 이용한 AOP 구현

aop 네임스페이스가 있는 XML 설정 파일에 Advice 클래스를 빈으로 등록

<aop:config> 태그를 이용해서 Advice를 어떤 포인트컷에 적용할지 지정

<aop:config>태그를 이용하여 AOP관련정보를 설정

```
12
13    <bean id="common" class="spring.study.aop.CommonAdvice" />
14
15    <bean id="data" class="spring.study.aop.DataImpl"></bean>
16
17    <aop:config>
18       <aop:aspect id="commonAdvice" ref="common">
19
20          <aop:pointcut id="aroundPush" expression="within(spring.study.aop.*)"/>
21
22          <aop:around pointcut-ref ="aroundPush" method="around" />
23
24       </aop:aspect>
25    </aop:config>
26
```

spring.study.aop package의 모든 메소드 호출 전후로
공통 로직으로 CommonAdvice의 around()를 호출 의미

빈 객체 사용

```
1  package spring.study.aop;
2
3  import org.springframework.context.ApplicationContext;
4  import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6  public class AopTest {
7
8      public static void main(String[] args) {
9
10         ApplicationContext ctx = new ClassPathXmlApplicationContext("applicationContext.xml");
11         Data data = (Data)ctx.getBean("data");
12
13         data.around("김혜경");
14
15     }
16 }
```

외부 Advice 설정

- 외부에서 정의한 트랜잭션 Advice를 포인트컷과 연결할 수 있는 <aop:advisor>
- <aop:advisor>는 <aop:aspect> 와 달리 외부에서 정의한 어드바이스를 포인트컷으로 연결할 목적으로 사용
- <tx:advice> 태그를 이용하여 선언한 txAdvice 라는 이름의 외부에 정의된 어드바이스를 포인트컷으로 지정하기 위해서 <aop:config> 태그 내에 <aop:advisor> 태그를 사용

```
<aop:config>
    <aop:pointcut id=" businessLogic" expression="within(spring.study.aop.*)"/>
    <aop:advisor pointcut-ref="businessLogic" advice-ref="txAdvice"/>
</aop:config>
<tx:advice id="txAdvice">
    <tx:attributes>
        <tx:method name=="*" propagation="REQUIRED" />
    </tx:attributes>
</tx:advice>
```

애노테이션 기반의 AOP 구현

개발 단계

@Aspect 애노테이션을 이용하여 Advice 클래스를 구현
(Advice를 구현한 메서드와 PointCut 포함)

XML 설정 파일에는 <aop:aspectj-autoproxy /> 설정

애노테이션 기반의 AOP 구현

@Aspect 애노테이션을 이용하여 Advice 클래스를 구현

```
3  import org.aspectj.lang.ProceedingJoinPoint;
4  import org.aspectj.lang.annotation.Aspect;
5  import org.aspectj.lang.annotation.Pointcut;
6  import org.aspectj.lang.annotation.Around;
7  @Aspect
8  public class CommonAdvice {
9      @Pointcut("within(spring.study.aop.*)")
10     private void adviceMethod(){}
11
12     @Around("adviceMethod()")
13     public void around(ProceedingJoinPoint point) throws Throwable{
14         String methodName = point.getSignature().getName();
15         String targetName = point.getTarget().getClass().getName();
16         System.out.println("target 클래스명 : " + targetName + " 및 메소드명 : " + methodName);
17
18         long startTime = System.nanoTime();
19         System.out.println("[Log]핵심 로직 메소드 호출전 --> "+ methodName+" time check start");
20
21         Object obj = point.proceed();
22
23         long endTime = System.nanoTime();
24         System.out.println("[Log]핵심 로직 메소드 호출 후 --> " + methodName+" time check end");
25         System.out.println("[Log] "+ methodName+" 실행 소요 시간 "+(endTime - startTime)+"ns");
26
27         System.out.println(targetName + " 메서드 실행후 리턴된 데이터 : " + obj);
28     }
29 }
```

proxy대상의 실제 메소드 호출

애노테이션 기반의 AOP 구현

```
 8 @Aspect
 9 public class CommonAdvice {
10    @Pointcut("within(spring.study.aop.*)")
11    private void adviceMethod(){}
12
13    @Around("adviceMethod()")
14    public void around(ProceedingJoinPoint point) throws Throwable{
15       //... 중략 ...
16
17       String methodName = point.getSignature().getName();
```

@Aspect
: XML 설정 파일에 <aop:aspect> 태그 설정 없이도 클래스 자체가 어드바이스가 됨
@Pointcut
: 모든 클래스들의 메소드들을 포인트컷으로 설정하는 문법
  이 애노테이션이 적용된 메소드의 리턴 타입은 void여야 하며, 일반적으로 메소드 body에는 특별한 로직의 코드를 갖지 않으며, 이 메소드의 이름을 이용해서 다른 애노테이션에서 해서 활용하게 됨
@Around
: target 객체의 메소드 실행 전, 후 또는 예외 발생 시점에 모두 실행 할수 있게 설정

애노테이션 기반의 AOP 구현

XML 설정 파일에는 <aop:aspectj-autoproxy /> 설정

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4
5     xmlns:aop="http://www.springframework.org/schema/aop"
6
7     xsi:schemaLocation="http://www.springframework.org/schema/beans
8        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
9
10       http://www.springframework.org/schema/aop
11       http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
12
13    <aop:aspectj-autoproxy />
14
15    <bean id="common" class="spring.study.aop.CommonAdvice" />
16
17    <bean id="data" class="spring.study.aop.DataImpl"></bean>
18
19 </beans>
```

<aop:aspectj-autoproxy />
이 태그를 사용하게 되면 @Aspect 애노테이션이 적용된 빈 객체를 Aspect로 사용하게 됨

XML 기반 : 애노테이션 기반 비교해 보기

Advice

```
개발 방법 1
public void afterThrowing(){
        System.out.println("메서드 실행 중 예외 발생, 예외”);
}

개발 방법 2
public void afterThrowing(Throwable ex){
        System.out.println("메서드 실행 중 예외 발생, 예외=" + ex.getMessage());
}
```

설정 파일

```
<aop:aspect id="commonAspect" ref="common">
        <aop:pointcut  id=" exceptionPush" expression="within(spring.study.aop.*)"/>

         <!– 개발 방법 1 -->
        <aop: after-throwing pointcut-ref="exceptionPush" method="afterThrowing”/>
        <!– 개발 방법 2 -->
        <aop: after-throwing pointcut-ref=”exceptionPush” method="afterThrowing”
                                throwing=”ex”/>
</aop:aspect>
```

XML 기반 : 애노테이션 기반 비교해 보기

Advice

개발 방법 1 - Advice 대상 객체의 메소드가 리턴한 값이 없을 경우 설정하는 메소드

```
@AfterThrowing("adviceMethod()")
public void afterThrowing(Throwable ex){
        System.out.println("메서드 실행 중 예외 발생, 예외=" + ex.getMessage());
}
```

개발 방법 2 - Advice 대상 객체의 메소드가 리턴한 값이 있을 경우 설정하는 메소드

```
@ AfterThrowing (pointcut=" adviceMethod ()", throwing="ex")
public void afterThrowing (Throwable ex){
        System.out.println("메서드 실행 중 예외 발생, 예외=" + ex.getMessage());
}
```

설정 파일

개발방법 1 - @Aspect가 선언된 클래스에 @AfterThrowing("adviceMethod()") 애노테이션을 설정하게 되면 target 클래스의 메소드가 예외 발생시 afterThrowing() 메소드는 자동 실행됨

개발 방법 2 – target 객체의 메소드가 발생시킨 예외 객체를 받기 위한 파라미터를 throwing 속성값으로 지정하게 됨. 이 때 어드바이스 클래스의 해당 메소드 파라미터 명과 동일해야 함

포인트컷 표현식 살펴보기

XML 스키마 기반의 POJO 클래스를 이용한 AOP 구현

```
<aop:config>
    <aop:aspect id="commonAdvice" ref="common“>
        <aop:pointcut  id="aroundPush“
                       expression="within(spring.study.aop.*)"/>

        <aop:around pointcut-ref="aroundPush" method="around"/>
    </aop:aspect>
</aop:config>
```

@Aspect 애노테이션 기반의 AOP 구현

```
@Pointcut("within(spring.study.aop.*)")
private void aroundPush(){}
```

execution() 표현식

- 가장 대표적인 강력한 포인트컷의 지시자
- 메소드의 signature를 이루는 접근 제한자, 리턴타입, 메소드명, 파라미터 타입, 예외타입 조건을 조합해서 메소드 단위까지 선택 가능한 가장 정교한 포인트컷 구성 가능

execution([**접근제한자 패턴**] **타입패턴** [**타입패턴**.] **이름패턴** (타입패턴 | "..", ...)
[throws 예외 패턴])

public과 같은 접근 제한자

리턴 값의 타입

패키지와 클래스 이름에 대한 패턴
사용할 때 '.'을 두어서 연결해야 함

파라미터의 타입 패턴을 순서대로 넣을 수 있고, 와일드카드를 이용해 파라미터 개수에 상관없는 패턴을 만들 수 있음

execution() 표현식 예시

예	설명
execution(* hello(..))	리턴 타입과 파라미터는 상관 없이 메소드명이 hello인 모든 메소드를 선정하는 포인트컷 표현식
execution(* hello(int, int))	리턴 타입은 상관없이 메소드 이름이 hello 이며, 두개의 int 타입의 파라미터를 가진 모든 메소드를 선정하는 포인트컷 표현식
execution(* *(..))	리턴 타입과 파라미터의 종류, 개수에 상관없이 모든 메소드를 선정하는 포인트컷 표현식

within() 표현식

- 특정 타입에 속하는 메소드를 포인트컷으로 설정할 때 사용
- execution()의 여러 조건 중에서 타입 패턴만을 적용하기 때문에 execution 표현식 문법보다 간단
- 단, 타깃 클래스의 타입에만 적용되며 조인포인트는 target 클래스 안에서 선언된 것만 선정됨
- 선택된 타입의 모든 메소드가 AOP 적용 대상이 됨

예	설 명
within(spring.study.aop...*)	spring.study.aop 및 모든 서브패키지가 포함하고 있는 모든 메소드
within(spring.study.aop.*)	spring.study.aop 패키지 밑의 인터페이스와 클래스에 있는 모든 메소드
within(spring.study.aop.Hello)	spring.study.aop 패키지의 Hello클래스의 모든 메소드

7. Spring MVC를 이용한 웹 요청 처리

- Spring MVC
- 프론트 Controller 패턴
- Spring과 프론트 Controller 패턴
- 캐릭터 인코딩 처리를 위한 필터 설정
- Spring3.0에서의 Controller 구현
- HTML 폼과 자바빈 객체
- 예외 처리

Spring MVC

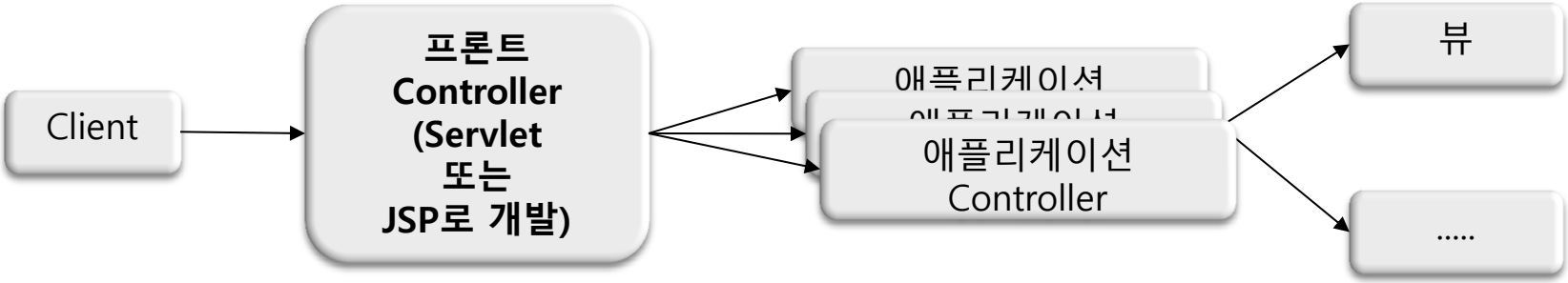
- Model2 구조를 프레임워크 차원에서 제공하는 모듈

- Spring은 DI나 AOP 같은 기능뿐만 아니라 웹 개발을 위한 MVC 프레임워크도 직접 제공하는 서블릿 기반의 MVC 프레임워크

- Spring MVC 프레임워크는 Spring을 기반으로 하고 있기 때문에 Spring이 제공하는 트랜잭션 처리나 DI 및 AOP등을 손쉽게 사용할 수 있다는 장점이 있음

프론트 Controller 패턴 아키텍처

- 프론트 Controller 프로세스



- 클라이언트가 보낸 요청을 받아서 공통적인 작업을 먼저 수행
- 적절한 세부 Controller로 작업 위임
- 클라이언트에게 보낼 뷰를 선택해서 최종 결과를 생성하는 등의 작업 수행
  - 인증이나 권한 체크 처럼 모든 요청에 대하여 공통적으로 처리되어야 하는 로직이 있을 경우 전체적으로 클라이언트의 요청을 중앙 집중적으로 관리하고자 할 경우에 사용

Spring과 프론트 Controller 패턴

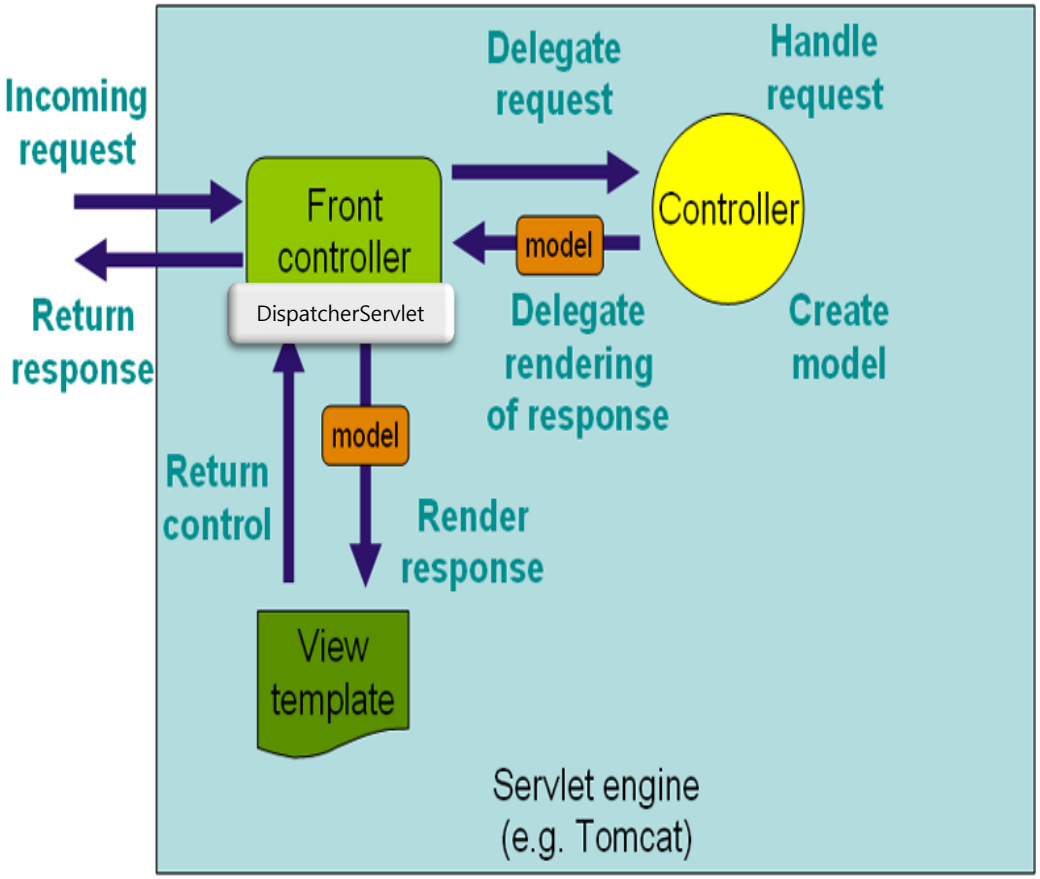
- 프론트 Controller 필요성

  - 다양한 클라이언트의 요청을 제어하려면 많은 수의 controller가 필요하기 때문에 매우 복잡해지고 분산됨
  - 전체 애플리케이션을 통합, 관리하기 위해 프론트 Controller 패턴을 적용
  - JavaEE Core Design Pattern에서 프리젠테이션 계층를 위한 패턴으로 대부분의 MVC 프레임워크들은 이 패턴을 적용해서 구현
  - 중앙 집중형 Controller를 프리젠테이션 계층의 앞단에 놓고 서버로 들어오는 모든 요청을 먼저 받아서 처리하도록 구성하는 것으로 Spring MVC에 적용해서 클라이언트 요청을 관리 하고자 할 때 사용됨
  - 예외가 발생 했을 경우 일관된 방식으로 처리하는 것도 프론트 Controller의 역할

Spring과 프론트 Controller 패턴

- DispatcherServlet API

  - web.xml에 설정
  - 프론트 Controller 패턴 적용
  - client 요청을 전달받음
  - controller나 view와 같은 Spring MVC의 구성 요소를 이용하여 client에게 서비스 제공



Spring과 프론트 Controller 패턴

- 실행 process

1단계 – DispatcherServlet이 HTTP 요청을 받음

2단계 – DispatcherServlet에서 서브 Controller로 HTTP 요청 위임

3단계 – Controller의 모델 생성과 정보를 등록하고 클라이언트의 요청 결과 리턴

4단계 – DispatcherServlet의 모델로 뷰 생성

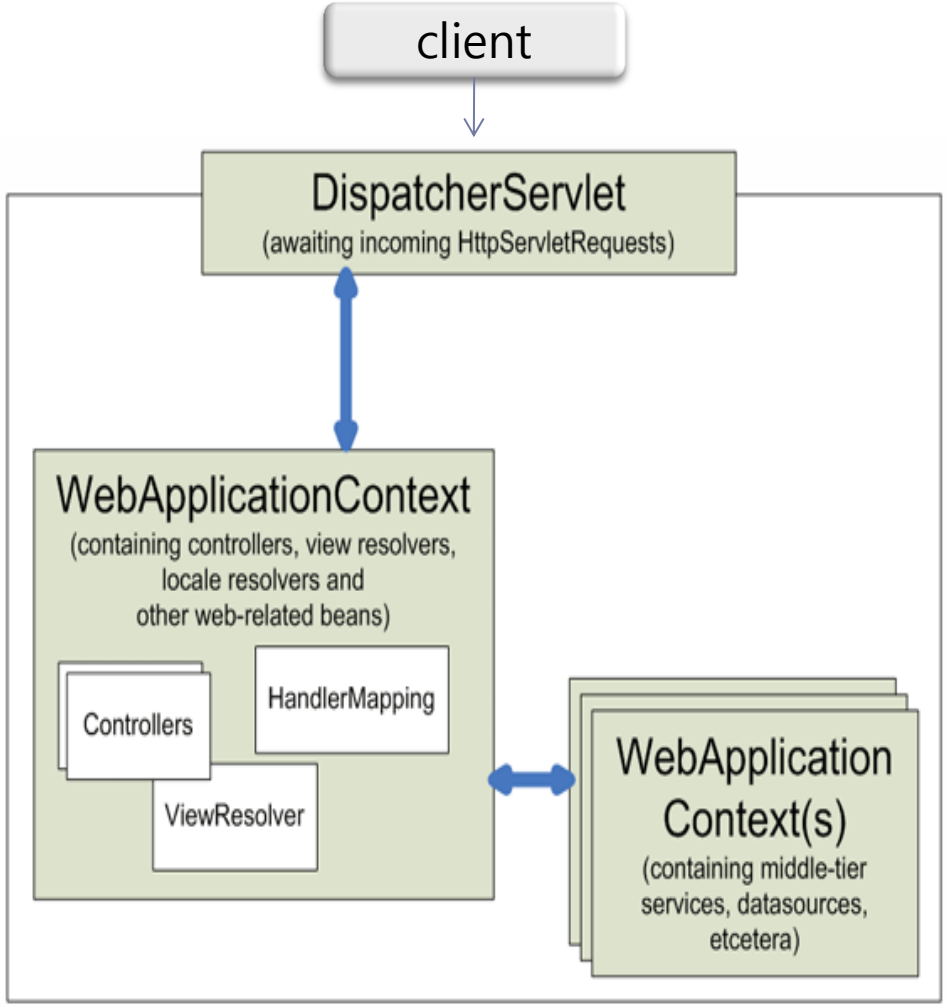
5단계 - HTTP응답 돌려주기

Spring MVC의 주요 구성 요소

구성 요소	설 명
DispatcherServlet	클라이언트의 요청을 받아서 Controller에게 클라이언트의 요청을 전달하고, 리턴한 결과값을 View에게 전달하여 알맞은 응답을 생성
HandlerMapping	URL과 요청 정보를 기준으로 어떤 핸들러 객체를 사용할지 결정하는 로직의 객체 DispatcherServlet은 하나 이상의 핸들러 매핑을 가질 수 있음
Controller	클라이언트의 요청을 처리한 뒤 그 결과를 DispatcherServlet에게 알려 줌 Struts의 Action과 동일한 기능
ModelAndView	Controller가 처리한 결과정보(Model) 및 뷰 선택에 필요한 정보를 보유한 객체
ViewResolver	Controller가 리턴한 뷰 이름을 기반으로 Controller 처리 결과를 생성할 뷰를 결정
View	Controller의 처리 결과 화면을 생성함

Spring MVC의 주요 구성 요소
- Spring MVC의 클라이언트 요청 처리 과정

- client의 요청이 DispatcherServlet에게 전달
- DispactherServlet은 HandlerMapping을 사용하여 client의 요청을 처리할 Controller 객체를 획득
- DispatcherServlet은 controller 객체를 이용하여 client의 요청 처리
- controller는 client의 요청 처리 결과 정보를 담은 ModelAndView 객체를 반환
- DispatcherServlet은 ViewResolver로부터 응답 결과를 생성할 View 객체를 구함
- View는 client에게 전송할 응답을 생성



Spring MVC 웹 애플리케이션 개발 단계

1단계 – client의 요청을 받을 DispatcherServlet 을 web.xml 파일에 설정

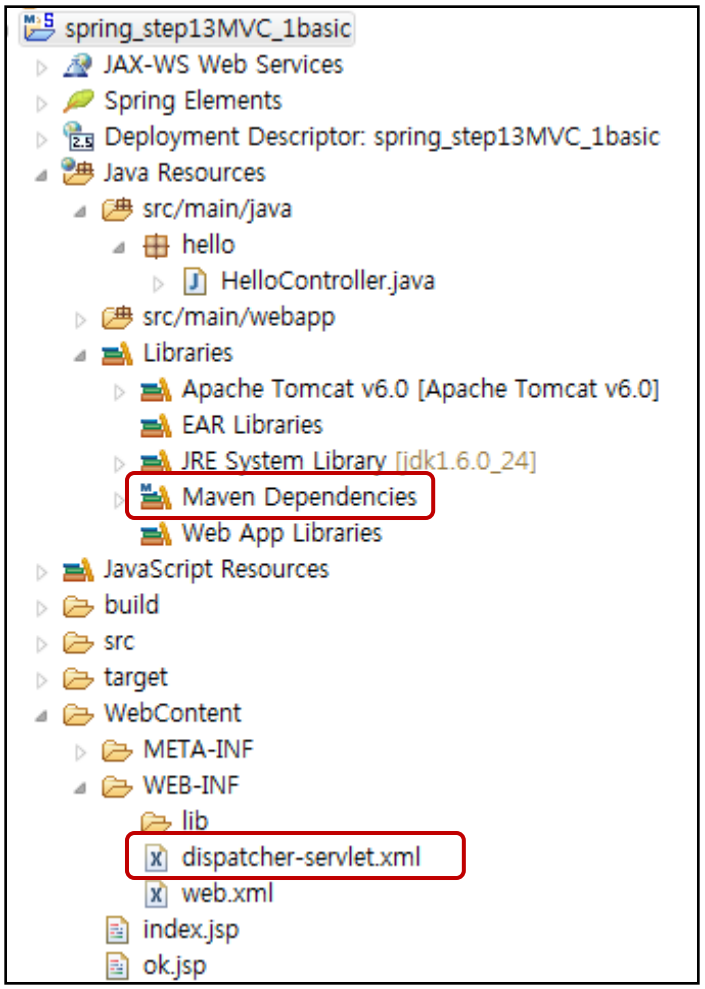
2단계 – client의 요청을 처리할 Controller 작성

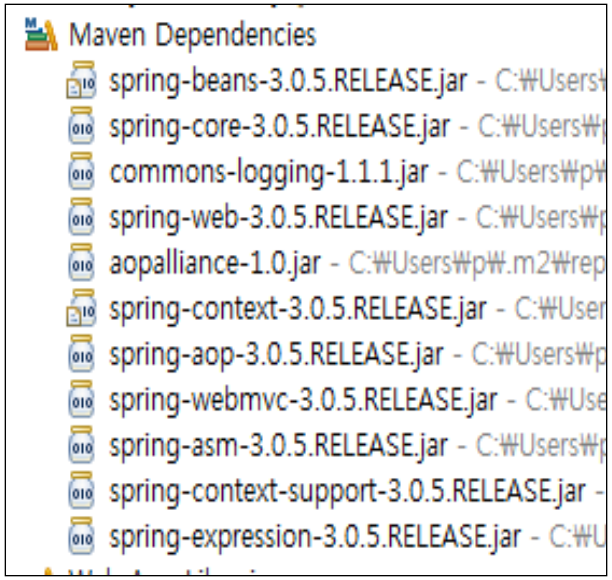
3단계 – Spring 빈으로 controller 등록 & ViewResolver 설정

4단계 – JSP를 이용한 뷰 영역의 코드 작성

5단계 – 실행

Spring MVC 웹 애플리케이션 개발 구조





1. Spring 웹 프로그램 설정 파일
2. web.xml의 <servlet-name> tag의 "데이터값"-servlet.xml이 파일명

Spring MVC 웹 애플리케이션 개발

1단계 – client의 요청을 받을 DispatcherServlet 을 web.xml 파일에 설정

```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/j2ee"
   xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
   xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
                       http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
                       version="2.4">
  <servlet>
      <servlet-name>dispatcher</servlet-name>
      <servlet-class>
         org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
  </servlet>

  <servlet-mapping>

      <servlet-name>dispatcher</servlet-name>

      <url-pattern>*.do</url-pattern>

  </servlet-mapping>

</web-app>
```

client의 요청을 전달받는 서블릿 Controller, 뷰와 같은 SpringMVC 의 구성 요소를 이용하여 client에게 서비스

1. *.do로 요청되는 모든 client 의 요청을 DispatcherServlet이 처리하도록 요청

2. dispatcher-servlet.xml이라는 이름의 Spring 설정 파일을 사용하도록 설정

Spring MVC 웹 애플리케이션 개발

2단계 – client의 요청을 처리할 Controller 작성

```
3  import org.springframework.stereotype.Controller;
4  import org.springframework.web.bind.annotation.RequestMapping;
5  import org.springframework.web.servlet.ModelAndView;
6
7  @Controller
8  public class HelloController{
9
10     @RequestMapping("/hello.do")
11     protected ModelAndView hello() {
12
13         ModelAndView mv = new ModelAndView();
14
15         mv.setViewName("ok");
16
17         mv.addObject("message", "Spring MVC");
18
19         return mv;
20     }
21 }
```

1. ModelAndView
- controller의 처리 결과를 보여줄 뷰와 뷰에서 출력할 모델 지정시 사용되는 객체
2. ok.jsp를 view로 지정
3. "message"라는 이름으로 "Spring MVC" 데이터 추가, ok.jsp에서 사용될 데이터

- @Controller – Spring MVC의 controller 를 구현한 클래스로 지정
- @RequestMapping – /hello.do라는 url값으로의 요청을 처리할 메소드 지정

[http://host:port/contextPath/hello.do]

Spring MVC 웹 애플리케이션 개발

3단계 – Spring 빈으로 controller 등록 & ViewResolver 설정

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans  xmlns="http://www.springframework.org/schema/beans"
3          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4          xsi:schemaLocation="http://www.springframework.org/schema/beans
5          http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
6
7
8     <!--  DispatcherServlet은 Spring 컨테이너에서 controller 객체를 검색하기 때문에
9           Controller를 bean으로 등록해 주어야 함
10    -->
11    <bean name="/helloController" class="hello.HelloController" />
12
13    <!--
14       1. ViewResolver : 컨트롤러의 처리결과를 보여줄 뷰를 선택하는 방식
15       2. JSP를 view 기술로 사용할 경우 InternalResourceViewResolver를 bean으로 등록
16    -->
17    <bean   id="viewResolver"
18            class="org.springframework.web.servlet.view.InternalResourceViewResolver">
19
20       <!-- view 위치 및 이름 설정 : /WEB-INF/뷰이름.jsp -->
21       <property name="prefix" value="/"></property>
22       <property name="suffix" value=".jsp"></property>
23
24    </bean>
25
26 </beans>
```

Spring MVC 웹 애플리케이션 개발

4단계 – JSP를 이용한 뷰 영역의 코드 작성

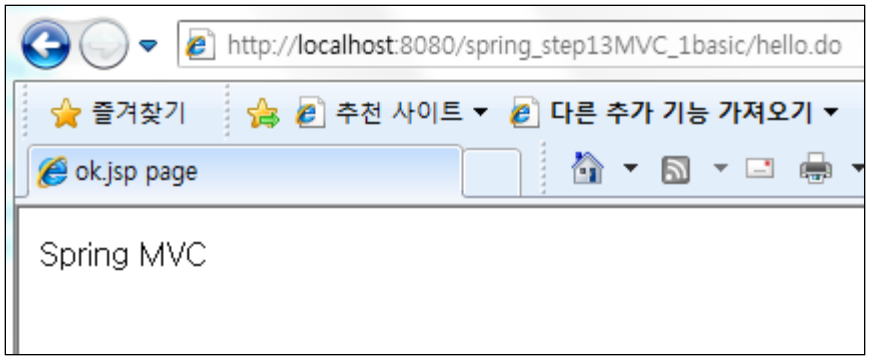
```jsp
<%@ page language="java" import="java.util.*" pageEncoding="EUC-KR"%>
<%
    //url값
    String path = request.getContextPath();
    String basePath = request.getScheme()+"://"+request.getServerName()
                      + ":" + request.getServerPort()+path+"/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>ok.jsp page</title>
    </head>

    <body>
        ${message} <br>
    </body>

</html>
```

controller

```
ModelAndView mv = new ModelAndView();
mv.setViewName("ok");
mv.addObject("message", "Spring MVC");
```

Spring MVC 웹 애플리케이션 개발

5단계 – 실행



캐릭터 인코딩 처리를 위한 필터 설정

- Spring은 servlet 필터를 이용하여 요청 파라미터의 캐릭터 인코딩 설정 가능

```
10    <filter>
11       <filter-name>encodingFilter</filter-name>
12       <filter-class>
13          org.springframework.web.filter.CharacterEncodingFilter
14       </filter-class>
15       <init-param>
16          <param-name>encoding</param-name>
17          <param-value>EUC-KR</param-value>
18       </init-param>
19    </filter>
20    <filter-mapping>
21       <filter-name>encodingFilter</filter-name>
22       <url-pattern>/*</url-pattern>
23    </filter-mapping>
```

Spring3.0에서의 Controller 구현 - @Controller

- @Controller 애노테이션 이용 권장

- controller 구현을 위한 필수 애노테이션
  - @Controller & @RequestMapping

- 개발 방법

1단계 - Controller 클래스에 @Controller 애노테이션 선언

2단계 - client의 요청을 처리할 메소드에 @RequestMapping 애노테이션 적용

3단계- [servletname]-servlet.xml에 controller 클래스를 빈으로 등록

Spring3.0에서의 Controller 구현 - @Controller

- Controller를 위한 핵심 애노테이션

구성 요소	설 명
@Controller	Controller 클래스 정의
@RequestMapping	HTTP 요청 URL을 처리할 Controller 메소드 정의
@RequestParam	HTTP 요청에 포함된 파라미터 참조 시 사용
@ModelAttribute	HTTP 요청에 포함된 파라미터를 모델 객체로 바인딩 @ModelAttribute의 'name'으로 정의한 모델 객체를 다음 뷰에게 사용 가능

Spring3.0에서의 Controller 구현 - @RequestMapping

- 클라이언트의 요청을 처리할 메소드에 선언하는 애노테이션

- 문법
  - 문법 1 - 요청 URL만 선언할 경우
    @RequestMapping("/요청한 URL.do")
  - 문법 2 – 요청 방식을 지정할 경우
    @RequestMapping(value="/요청한 URL.do",
    method=RequestMethod.POST)

- Controller의 처리 결과를 보여줄 view 지정 방법
  - 메소드의 반환 값에 따른 view page지정 방법
  - 방법 1 - ModelAndView인 경우
    : setViewName() 메소드 파라미터로 설정
  - 방법 2 - String인 경우
    : 메소드의 리턴값

Spring3.0에서의 Controller 구현 - @RequestMapping

- 호출하는 URL이 다른 경우

```
 8 @Controller
 9 public class HelloController{
10
11    @RequestMapping("/hello.do")
12    protected ModelAndView hello() {
13        ModelAndView mv = new ModelAndView();
14        mv.setViewName("ok");
15        mv.addObject("message", "Spring MVC");
16        return mv;
17    }
18
19    @RequestMapping(value="/postMethod.do", method=RequestMethod.POST)
20    protected String postMethod() {
21        //...중략...
22        return "postView";
23    }
```

Spring3.0에서의 Controller 구현 - @RequestMapping

- 동일한 URL로 Get/Post 방식 모두 처리할 경우의 설정

```
@Controller
@RequestMapping("/hello.do")
public class HelloController {
	@RequestMapping(method=RequestMethod.GET)
	public ModelAndView getHelloMessage() {
		//...
	}
	@RequestMapping(method=RequestMethod.POST)
	public ModelAndView getHelloMessage() {
		//...
	}
}
```

HTML 폼과 자바빈 객체

- HTML 폼에 입력한 데이터를 자바 빈 객체로 전달하는 방법
  - @RequestMapping 이 적용된 메소드의 파라미터로 자바빈 타입 추가 설정

```
25    post 방식으로 controller 호출해 보기
26    <form method="post" action="postMethod.do">
27       id : <input type="text"  name="id"/><br>
28       이름 : <input type="text" name="name"/><br>
29       age : <input type="text" name="age"/><br>
30       <input type="submit" value="post요청" />
31    </form>
```

```
3  public class User {
4     private String id;
5     private String name;
6     private int age;
7     public User() {}
8     public User(String id, String name, int age) {
9        this.id = id;
10       this.name = name;
11       this.age = age;
12    }
13
```

```
21    @RequestMapping(value="/postMethod.do",
22                    method=RequestMethod.POST)
23    protected String postMethod(User user) {
24       return "postView";
25    }
26
```

1

2

데이터 입력
-> User빈 객체 생성
-> 각 멤버 변수에 입력된 데이터 대입
-> postMethod 메소드의 파라미터로 대입

HTML 폼과 자바빈 객체

- view에서는 Controller의 @RequestMapping 메소드에서 전달받은 자바빈 객체 access 가능
- view 코드
  - 자바빈 객체의 클래스 이름(첫글자 소문자)를 이용하여 access
  - view 코드에서 사용할 자바빈의 이름을 변경하고자 할 경우 @ModelAttribute 애노테이션 사용

controller

```
21  @RequestMapping(value="/postMethod.do",
22                  method=RequestMethod.POST)
23  protected String postMethod(User user) {
24     return "postView";
25  }
26
```

```
3  public class User {
4     private String id;
5     private String name;
6     private int age;
7     public User() {}
8     public User(String id, String name, int age) {
9        this.id = id;
10       this.name = name;
11       this.age = age;
12    }
13
```

view

```
client가 입력한 id값 - ${user.id}
```

@RequestParam 애노테이션 - 파라미터매핑

- HTTP 요청 파라미터를 메소드의 파라미터로 전달받을 때 사용
- http://localhost:8080/spring_step13MVC_1basic/getMethod.do?**id=tester**

```
30  @RequestMapping(value="/getMethod.do",
31                  method=RequestMethod.GET)
32  protected String getMethod(@RequestParam("id") String id) {
33      //...중략
34      return "getView";
35  }
```

Spring에서의 Exception Handling

- 컨트롤러 클래스의 @RequestMapping이 선언된 메소드는 모든 타입의 예외를 발생 시킬 수 있음

- 에러 처리 방법

구성 요소	설 명
HandlerExceptionResolver	HandlerExceptionResolver 인터페이스의 resolveException() 메서드는 발생된 예외 객체를 파라미터로 전달받음
@ExceptionHandler	**@Controller 애노테이션이 적용된 클래스에 @ExceptionHandler 애노테이션이 적용된 메소드 구현**
설정 파일을 이용한 선언적 예외 처리	SimpleMappingExceptionResolver클래스를 이용하기 위해 설정 파일에 설정

@ExceptionHandler 애노테이션

- submit()에서 예외 발생시 handleException()에서 예외 처리

```
32    @RequestMapping("/member.do")
33    public ModelAndView sumbit(MemberVO member) throws Exception{
34        HashMap map = new HashMap();
35        map.put("point", 1000);
36        map.put("info","basic point");
37
38        service.registerMemberInfo(member, map);
39        ModelAndView mv = new ModelAndView();
40        mv.setViewName("register_ok");
41        mv.addObject("info", member);
42        return mv;
43    }
44
45    @ExceptionHandler(Exception.class)
46    public String handleException(Exception ex){
47        ex.printStackTrace();
48        System.out.println("exception 발생");
49        return "/error";
50    }
```

8. Spring 트랜잭션

- Spring의 트랜잭션
- 선언적 트랜잭션
- 애노테이션 기반 트랜잭션

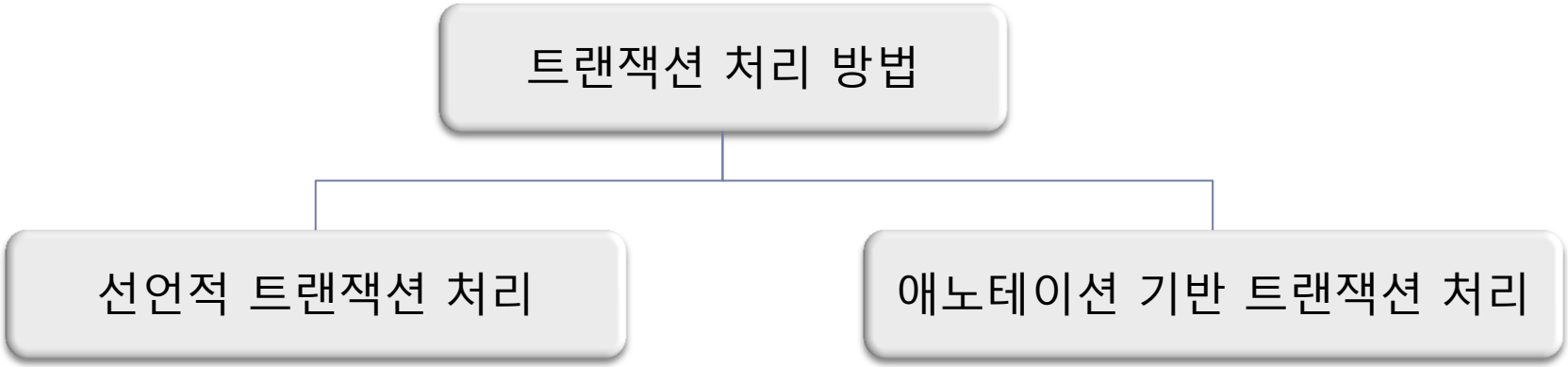
트랜잭션이란?

- 더이상 쪼갤수 없는 최소 단위의 작업
- 여러 단계의 작업이 처리될 경우 전체 로직이 모두 다 성공하거나 실패해야 하는 논리적인 작업의 묶음을 의미
- 트랜잭션 경계 안에서 진행된 작업은 commit을 통한 성공 및 rollback을 통한 취소 작업을 할 수 있음

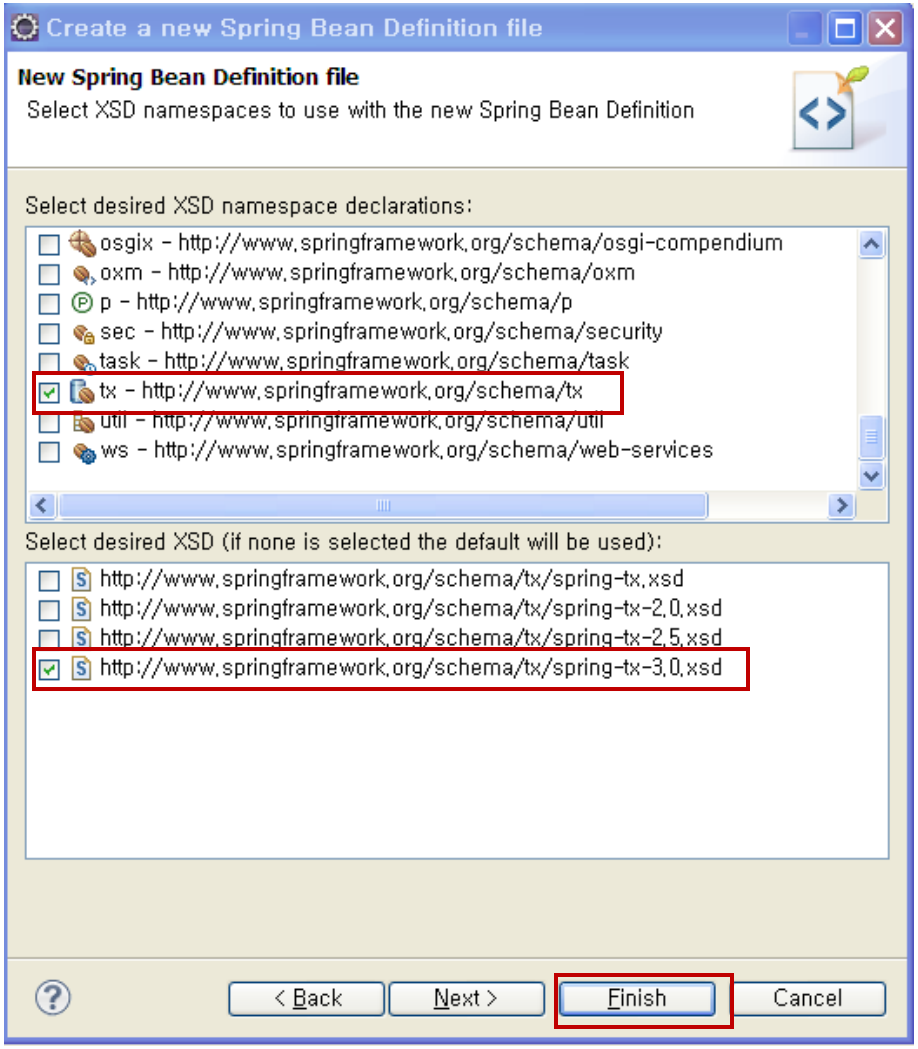
Spring 트랜잭션 적용 방법

- Spring은 데이터베이스 연동 기술과 트랜잭션 서비스 사이의 종속성을 제거하고 Spring이 제공하는 트랜잭션 추상 계층을 이용해서 데이터베이스 연동 기술에 상관없이 동일한 방식으로 트랜잭션 기능을 활용하도록 함

- 트랜잭션 서비스의 종류나 환경이 바뀌더라도 트랜잭션을 사용하는 코드는 그대로 유지할 수 있는 유연성 제공



Spring의 트랜잭션 적용 – 네임스페이스 설정



Spring의 트랜잭션 적용 – 네임스페이스 설정

- tx 네임스페이스를 <beans> 태그에 추가
  - <tx:advice> 태그, <tx:attributes> 태그, <tx:method> 태그를 이용해서 트랜잭션 속성을 정의

```
8     xmlns:tx="http://www.springframework.org/schema/tx"
9
10    xsi:schemaLocation="...
11                http://www.springframework.org/schema/tx
12                http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
13
14    ...">
```

선언적 트랜잭션-트랜잭션을 적용한 설정 파일

```
37    <!-- [트랜잭션 설정]
38       스프링은 다양한 데이터베이스 연동 기술에 상관없이 동일한 방식으로 트랜잭션 처리 지원 -->
39    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
40       <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
41       <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
42       <property name="username" value="scott"/>
43       <property name="password" value="tiger"/>
44    </bean>
45
46    <bean id="transactionManager"
47       class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
48       p:dataSource-ref="dataSource" />
49
50    <tx:advice id="txAdvice" transaction-manager="transactionManager">
51       <tx:attributes>
52          <tx:method name="*" propagation="REQUIRED" />
53       </tx:attributes>
54    </tx:advice>
55
56    <aop:config>
57       <aop:pointcut id="serviceInsertMethod" expression="within(spring.study..*)" />
58       <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceInsertMethod" />
59    </aop:config>
```

선언적 트랜잭션-트랜잭션을 적용한 설정 파일

- DataSourceTransactionManager API
  - Connection의 트랜잭션 API를 이용하여 트랜잭션을 관리해주는 트랜잭션메니저
  - JDBC와 iBatis SqlMap로 만든 DAO에 적용 할 수 있으며, DataSource가 Spring의 빈으로 등록되어 있어야 함

```
<bean id="transactionManager"
   class="org.springframework.jdbc.datasource.DataSourceTransactionManager"
   p:dataSource-ref="dataSource" />
```

선언적 트랜잭션-트랜잭션을 적용한 설정 파일

- 설정 내용
  - 트랜잭션 전파(propagation) 속성에 REQUIRED로 설정되어 있기 때문에 메소드 수행하는데 트랜잭션이 필요하다는 것을 의미
  - 즉, 현재 진행중인 트랜잭션이 존재하면 해당 트랜잭션을 사용하며 존재하지 않을 경우엔 새로운 트랜잭션을 생성하겠다는 의미

```
<tx:advice id="txAdvice" transaction-manager="transactionManager">
   <tx:attributes>
      <tx:method name="*" propagation="REQUIRED" />
   </tx:attributes>
</tx:advice>
```

선언적 트랜잭션-트랜잭션을 적용한 설정 파일

- <aop:config> tag
  - spring.study 패키지의 하위 패키지에 있는 모든 메소드에 <tx:advice> 태그로 설정한 트랜잭션을 적용하도록 설정

- 이미 <tx:method> tag의 name 속성이 "*"로 표기되어 있기 때문에 이 모든 메소드에 대해 트랜잭션을 적용하도록 설정했음

```
<aop:config>
   <aop:pointcut id="serviceInsertMethod" expression="within(spring.study..*)" />
   <aop:advisor advice-ref="txAdvice" pointcut-ref="serviceInsertMethod" />
</aop:config>
```

애노테이션 기반 트랜잭션

- @Transactional 애노테이션을 사용해서 트랜잭션을 설정

```
@Transactional
public void customerInsertMethod(Customer customer){ ... }
```

- @TransactionI 애노테이션이 적용된 Spring 빈에 실제로 트랜잭션을 적용하려면 Spring 설정 파일에 다음 태그 설정

```
<tx:annotation-driven transaction-manager="transactionManager" />
```

- @Transactional이 붙은 클래스나 인터페이스 또는 메소드를 찾아 트랜잭션 어드바이스를 자동 적용해 줌
- transation-manager 속성은 트랜잭션 메니저로 사용할 빈의 이름이 설정됨
- 이처럼 Spring에서의 트랜잭션 처리는 XML 설정 파일 또는 애노테이션만으로도 간편하게 적용 가능

9. Database 연동

- Spring의 데이터베이스 연동 지원
- Spring의 JDBC지원 – Template클래스 이용
- Spring의 JDBC지원 – DaoSupport 클래스 이용
- Spring의 iBatis 연동

Spring의 데이터베이스 연동 지원

- Spring은 JDBC를 비롯하여 ORM 프레임워크를 직접적으로 지원하고 있기 때문에 simple하게 JDBC뿐만 아니라 ORM 프레임워크들과의 연동도 매우 쉬움

- Spring은 JDBC, ORM 프레임워크 등의 다양한 기술을 이용해서 손쉽게 DAO클래스를 구현할 수 있도록 지원

Spring의 데이터베이스 연동 지원 - Template

- Template 클래스 지원
  - 장점
    - 개발자가 중복된 코드를 입력해야 하는 번거로운 작업을 줄일 수 있도록 함
    - try~catch~finally 블록 및 커넥션 관리를 위한 중복 코드 감소

- Template 클래스들
  - JDBC : JdbcTemplate
  - iBatis : SqlMapClientTemplate
  - Hibernate : HibernateTemplate

Spring의 데이터베이스 연동 지원 - DaoSupport

- DaoSupport 클래스 지원
  - DAO에서 기본적으로 필요로 하는 기능을 제공
  - Templage 클래스를 편리하게 사용 할 수 있게 해줌

- 장점
  - DaoSupport 클래스를 상속받아 DAO클래스를 구현한 뒤, DaoSupport 클래스가 제공하는 기능을 사용하여 보다 편리하게 코드를 작성할 있음

- DaoSupport 클래스들
  - JDBC : JdbcDaoSupport
  - iBatis : SqlMapClientDaoSupport
  - Hibernate : HibernateDaoSupport

Spring의 데이터베이스 연동 지원-Spring의 예외 처리 정책

- Spring은 데이터베이스 처리 과정에서 발생한 예외가 왜 발생했는지를 좀 더 구체적으로 확인 할 수 있도록 하기 위해 데이터베이스 처리와 관련된 예외 클래스를 제공
- 데이터베이스 처리 과정에서 SQLException이 발생하면 Spring이 제공하는 예외 클래스 중 알맞은 예외 클래스로 변환해서 예외를 발생 시킴
- Spring의 모든 예외 클래스들은 DataAccessException을 상속
  - 예 : BadSqlGrammerException,
    DataRetrievalFailureException
- Spring이 제공하는 템플릿 클래스를 사용하면 데이터베이스 연동을 위해 사용하는 기술에 독립적으로 동일한 방식으로 예외 처리가 가능

DataSource 설정

- Spring의 Connection 제공 방식
  - DataSource를 통해서 제공
  - 설정 파일에 DataSource 설정 필수

- Spring은 다음과 같은 3가지 설정 방식을 제공
  1. 커넥션 풀을 이용한 DataSource 설정
  2. JNDI를 이용한 DataSource 설정
  3. DriverManager를 이용한 DataSource 설정

DataSource설정 – 커넥션 풀 이용

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    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="dataSource"
            class="org.apache.commons.dbcp.BasicDataSource"
             p:driverClassName=" oracle.jdbc.driver.OracleDriver"
             p:url=" jdbc:oracle:thin:@localhost:1521:xe"
             p:username="scott"
             p:password="tiger" />
    ...

</beans>
```

Spring의 JDBC 지원 – Template 클래스이용

- 데이터 베이스 연동을 위한 Connection에 관련된 코드 및 예외 처리 등 반복적인 코드를 제거 할 수 있음

- Spring은 3개의 Template 클래스를 제공

API	설 명
JdbcTemplate	기본적인 JDBC 템플릿 클래스로서 JDBC를 이용해서 데이터에 대한 access를 지원하는 템플릿 클래스
NamedParameterJdbcTemplate	PreparedStatement에서 index 기반의 파라미터가 아닌 이름을 가진 파라미터를 사용 할 수 있도록 지원하는 템플릿 클래스
SimpleJdbcTemplate	JdbcTemplate 와 NamedParameterJdbcTemplate 클래스 기능을 하나도 합쳐놓은 클래스 이름 기반의 파라미터 설정과 인덱스 기반의 파라미터 설정 모두 다 지원

JdbcTemplate 클래스 특징

- SQL 실행을 위한 메소드 제공
- try~catch~finally 블록 및 커넥션 관리를 위한 중복 코드 삭제로 인한 코드량 감소 및 개발용이
- 주요 API

API	설 명
RowMapper	query() 메소드로 실행시킨 결과를 객체 목록으로 가져올 때에는 RowMapper를 이용 ResultSet에서 값을 가져와 원하는 타입으로 매핑할때 사용됨
query() queryForList() queryForObject() queryForInt() queryForLong()	조회를 위한 메소드들

JdbcTemplate 클래스 활용을 위한 설정

- JdbcTemplate클래스는 DataSource를 필요로 함
- JdbcTemplate 클래스를 빈으로 설정하고 미리 설정한 DataSource 빈을 dataSource 프로퍼티로 전달

applicationContext.xml

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

<bean id="jdbcTemplate"
        class="org.springframework.jdbc.core.JdbcTemplate"
        p:dataSource-ref="dataSource" />
...
```

JdbcTemplate 클래스를 활용한 DAO 클래스 개발

DAO작성시 JdbcTemplate 클래스를 주입 받을 수 있도록 프로퍼티와 setter메소드 또는 생성자를 구현

select 쿼리 결과 집합을 매핑할 커스텀 RowMapper를 작성

JdbcTemplate 클래스가 제공하는 메소드를 이용해서 CRUD 메소드 구현

DAO 클래스를 빈으로 설정하고 미리 설정한 JdbcTemplate 빈을 전달 받도록 설정

JdbcTemplate 클래스를 이용한 DAO 클래스

UserDAOImpl1.java

```
public class UserDAOImpl implements UserDAO {

    private JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
       this.jdbcTemplate = jdbcTemplate;
    }

    public List<User> findUserList() {
       String sql = "SELECT userid, password, name, email FROM userinfo";
      UserRowMapper rowMapper = new UserRowMapper();
      List<User> userList = jdbcTemplate.query(sql, rowMapper);
      return userList;
    }
}
```

RowMapper 인터페이스 구현 클래스 개발

UserRowMapper.java

```
public class UserRowMapper implements RowMapper {

	public Object mapRow(ResultSet rs, int rowNum) throws SQLException {
		User user = new User();
		user.setPassword(rs.getString("PASSWORD"));
		user.setName(rs.getString("NAME"));
		user.setEmail(rs.getString("EMAIL"));
		user.setUserId(rs.getString("USERID"));
		return user;
	}

}
```

JdbcTemplate 클래스 이용

applicationContext.xml

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

<bean id="jdbcTemplate"
       class="org.springframework.jdbc.core.JdbcTemplate"
       p:dataSource-ref="dataSource" />

<bean id="userDao" class="myspring.jdbc.UserDAOImpl"
       p:jdbcTemplate-ref="jdbcTemplate" />
...
```

DaoSupport 클래스 이용

- DaoSupport 클래스는 각각의 템플릿 클래스 별로 존재
- 각 DaoSupport 클래스는 템플릿 객체를 구할 수 있는 메소드를 제공

1. JdbcDaoSupport
   - JdbcTemplate지원

2. NamedParameterJdbcDaoSupport
   - NamedParameterJdbcTemplate 지원

3. SimpleJdbcDaoSupport
   - SimpleJdbcTemplate 지원

JdbcDaoSupport 클래스 특징

- DAO작성시 JdbcDaoSupport 클래스를 상속 받음
- JdbcDaoSupport 클래스가 제공하는 getJdbcTemplate() 메소드를 이용해서 JdbcTemplate 객체를 얻어내어 DAO 클래스 작성
- 즉, JdbcTemplate객체를 주입받을 생성자나 setter메소드가 필요 없음
- select 쿼리 결과 집합을 매핑할 커스텀 RowMapper 작성
- DAO 클래스를 빈으로 설정하고 미리 설정한 DataSource 빈을 전달 받도록 설정

JdbcDaoSupport 클래스 이용

UserDAOImpl2.java

```
public class UserDAOImpl2 extends JdbcDaoSupport implements UserDAO {

        public List<User> findUserList() {

        String sql = "SELECT userid, password, name, email FROM userinfo";
        UserRowMapper rowMapper = new UserRowMapper();
        List<User> userList = getJdbcTemplate().query(sql, rowMapper);
        return userList;

      }

}
```

JdbcTemplate 와 비교
- 생략된 코드

```
private JdbcTemplate jdbcTemplate;

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
  this.jdbcTemplate = jdbcTemplate;
}
```

JdbcDaoSupport 클래스 이용

applicationContext.xml

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

<bean id="userDao2" class="myspring.jdbc.UserDAOImpl2"
       p:dataSource-ref="dataSource"/>
...
```

JdbcTemplate 와 비교
- 생략된 코드

```
<bean id="jdbcTemplate"
        class="org.springframework.jdbc.core.JdbcTemplate"
        p:dataSource-ref="dataSource" />
```

Spring의 iBatis지원 API-SqlMapClientTemplate

- 내부적으로 iBatis의 SqlMapClient 사용하고 있기 때문에 대부분의 메소드와 동일한 이름의 파라미터 타입, 리턴 타입을 정의하고 있음
- 단, SQLException을 발생시키는 대신 Spring이 제공하는 예외를 발생 시킴
- Spring 설정 파일에서 쉽게 설정하도록 돕는 SqlMapclientFactoryBean 클래스를 제공
- 이 클래스를 사용하면 SqlMapClient를 Spring의 빈으로 설정할 수 있음
- SqlMapClient를 사용할 때의 단점인 반복적인 코드를 제거함

Spring의 iBatis지원 API-SqlMapClientDaoSupport

- SqlMapClientTemplate 클래스를 DAO 클래스에서 좀 더 쉽게 사용할 수 있도록 제공하는 클래스

- 설정 파일 내용
  - 이 클래스를 상속받은 클래스는 sqlMapClientTemplate 프로퍼티를 통해서 SqlMapClientTemplate 객체를 전달 받음
- 자바 코드
  - SqlMapClientDaoSupport 클래스가 제공하는 getSqlMapClientTemplate() 메소드를 이용해서 SqlMapClientTemplate 객체 획득
- 장점
  - SqlMapClientTemplate 객체를 주입받을 생성자나 setter메소드가 필요 없음[코드 간소화]
  - DAO 클래스를 빈으로 설정하고 미리 설정한 SqlMapClient빈을 전달 받도록 설정

Spring의 iBatis지원

MemberDaoImpl.java

```
3  import java.sql.SQLException;
4  import java.util.HashMap;
5
6  import org.apache.commons.logging.Log;
7  import org.apache.commons.logging.LogFactory;
8  import org.springframework.orm.ibatis.support.SqlMapClientDaoSupport;
9
10 import kicox.vo.MemberVO;
11
12 public class MemberDaoImpl extends SqlMapClientDaoSupport implements MemberDao{
13    private Log log = LogFactory.getLog(getClass());
14
15    public MemberDaoImpl(){ }
16
17    public void registerMember(MemberVO vo) throws SQLException {
18       getSqlMapClientTemplate().insert("member.insertMember", vo);
19    }
20    public void registerMemberPoint(HashMap map) throws SQLException {
21       getSqlMapClientTemplate().insert("memberpoint.insertMemPoint", map);
22    }
23 }
```

Spring의 iBatis지원

applicationContext.xml

```
29    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
30       <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
31       <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
32       <property name="username" value="scott"/>
33       <property name="password" value="tiger"/>
34    </bean>
35
36    <bean id="sqlMapClient" class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
37       <property name="configLocation" value="WEB-INF/SqlMapConfig.xml" />
38       <property name="dataSource" ref="dataSource" />
39    </bean>
40
41    <bean id="mdao" class="spring.dao.MemberDaoImpl">
42       <property name="sqlMapClient">
43          <ref bean="sqlMapClient" />
44       </property>
45    </bean>
```

SqlMapConfig.xml

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE sqlMapConfig
3     PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
4     "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">
5
6  <sqlMapConfig>
7     <settings useStatementNamespaces="true"/>
8     <sqlMap resource="config/member.xml"/>
9     <sqlMap resource="config/memberpoint.xml"/>
10 </sqlMapConfig>
```

참고 문헌

- Spring3.0 프로그래밍[최범균]