

테스트 환경 구축

테스트 환경 개요

- JDK (Java 11)
- Spring Tool Suite or Eclipse with Spring Plugin
- Apache Tomcat (9.x)
- MySQL (MariaDB) or Oracle
- Apache Maven / Gradle

JDK 설치

- 다운로드 → <https://www.oracle.com/java/technologies/downloads/#java11-windows>

The screenshot shows the Oracle Java download page for Java SE Development Kit 11.0.13. At the top, there are two tabs: "Java 8" and "Java 11", with "Java 11" being the active tab and highlighted with a red box. Below the tabs, the section title "Java SE Development Kit 11.0.13" is displayed. A note states: "Java SE subscribers will receive JDK 11 updates until at least September of 2026." Another note specifies: "These downloads can be used for development, personal use, or to run Oracle licensed products. Use for other purposes, including production or commercial use, requires a Java SE subscription or another Oracle license." A link to the "Oracle Technology Network License Agreement for Oracle Java SE" is provided. Under the "Downloads" heading, there are four operating system options: Linux, macOS, Solaris, and Windows, with "Windows" being the active tab and highlighted with a red box. A note below the tabs says: "JDK 11 software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE." A "checksum" link is also present. The main content area displays two download options:

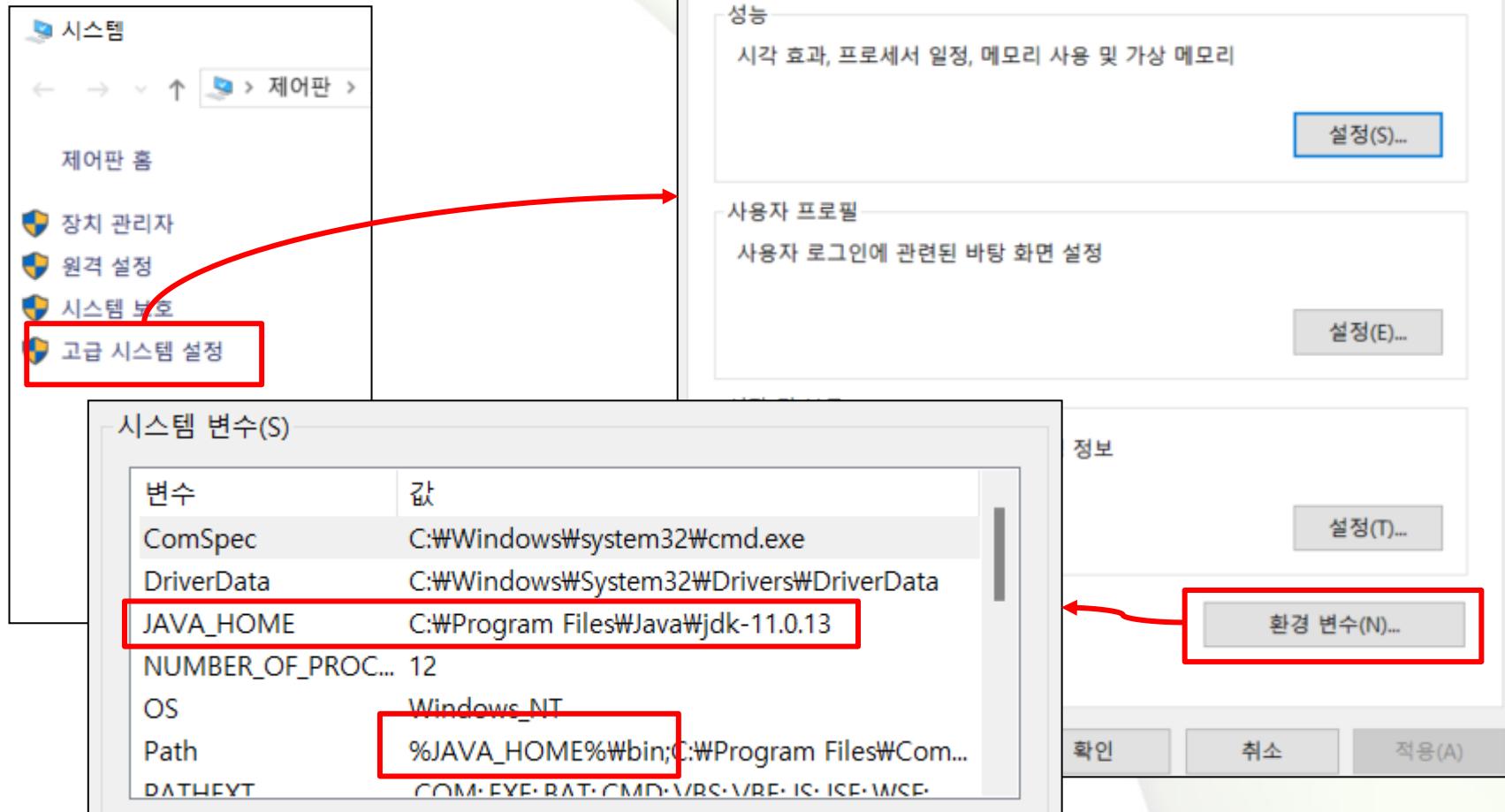
Product/file description	File size	Download
x64 Installer	139.83 MB	jdk-11.0.13_windows-x64_bin.exe
x64 Compressed Archive	157.28 MB	jdk-11.0.13_windows-x64_bin.zip

- 다운로드 완료 후 설치 실행

JDK 설치

▪ 환경변수 등록

- JAVA_HOME
- Path : %JAVA_HOME%\bin



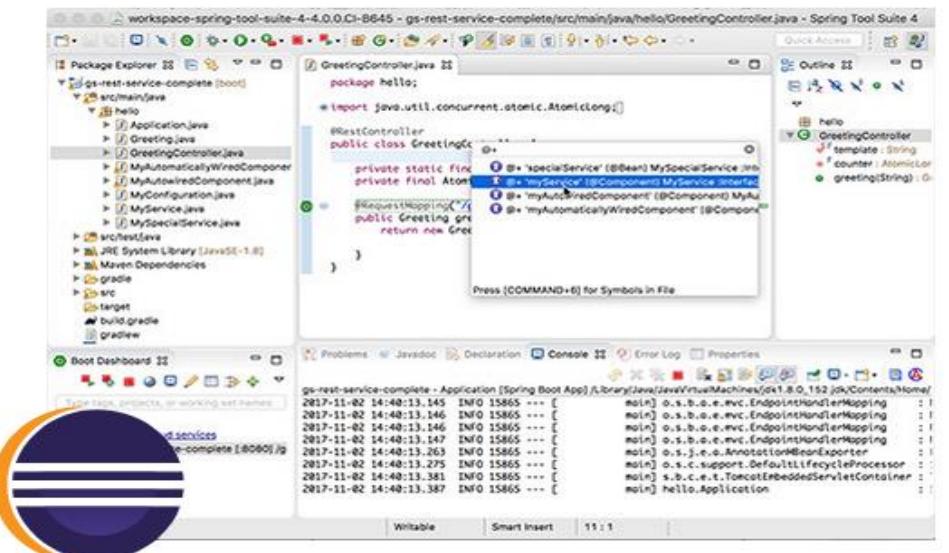
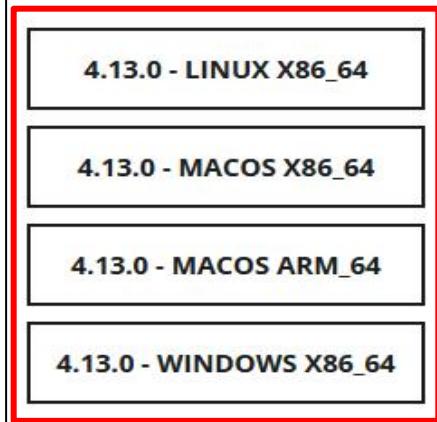
Spring Tool Suite (STS) 설치

- 다운로드 → <https://spring.io/tools>

Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4.

Free. Open source.



- 다운로드 후 실행 → 자동으로 압축 해제
 - .jar 파일 실행 (더블 클릭으로 실행되지 않을 경우 아래 방법으로 명령 실행)
 - 명령 프롬프트에서 java -jar spring-tool-suite-xxx.jar

Eclipse 설치

- 다운로드 → <https://www.eclipse.org/downloads/packages/>

Eclipse IDE for Enterprise Java and Web Developers

509 MB 145,348 DOWNLOADS

Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web Services, JPA and Data Tools, Maven and Gradle, Git, and more.

[Click here to file a bug against Eclipse Web Tools Platform.](#)
[Click here to file a bug against Eclipse Platform.](#)
[Click here to file a bug against Maven integration for web projects.](#)
[Click here to report an issue against Eclipse Wild Web Developer \(incubating\).](#)



Windows x86_64
macOS x86_64 | AArch64
Linux x86_64 | AArch64



 Download

Download from: Korea, Republic Of - Kakao Corp. (https)

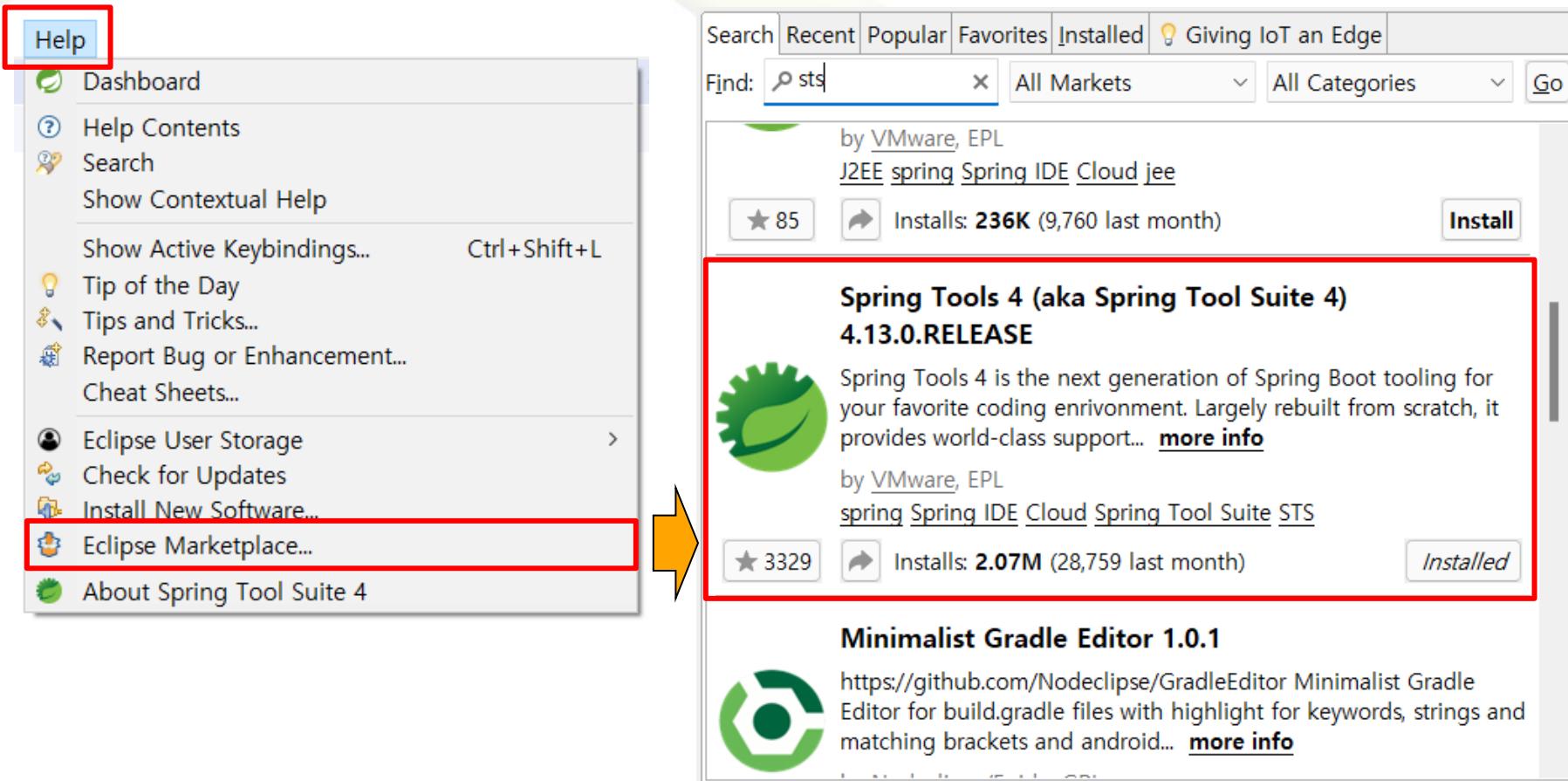
File: [eclipse-jee-2021-12-R-win32-x86_64.zip](#) SHA-512

>> Select Another Mirror

- 다운로드 완료 후 압축 해제

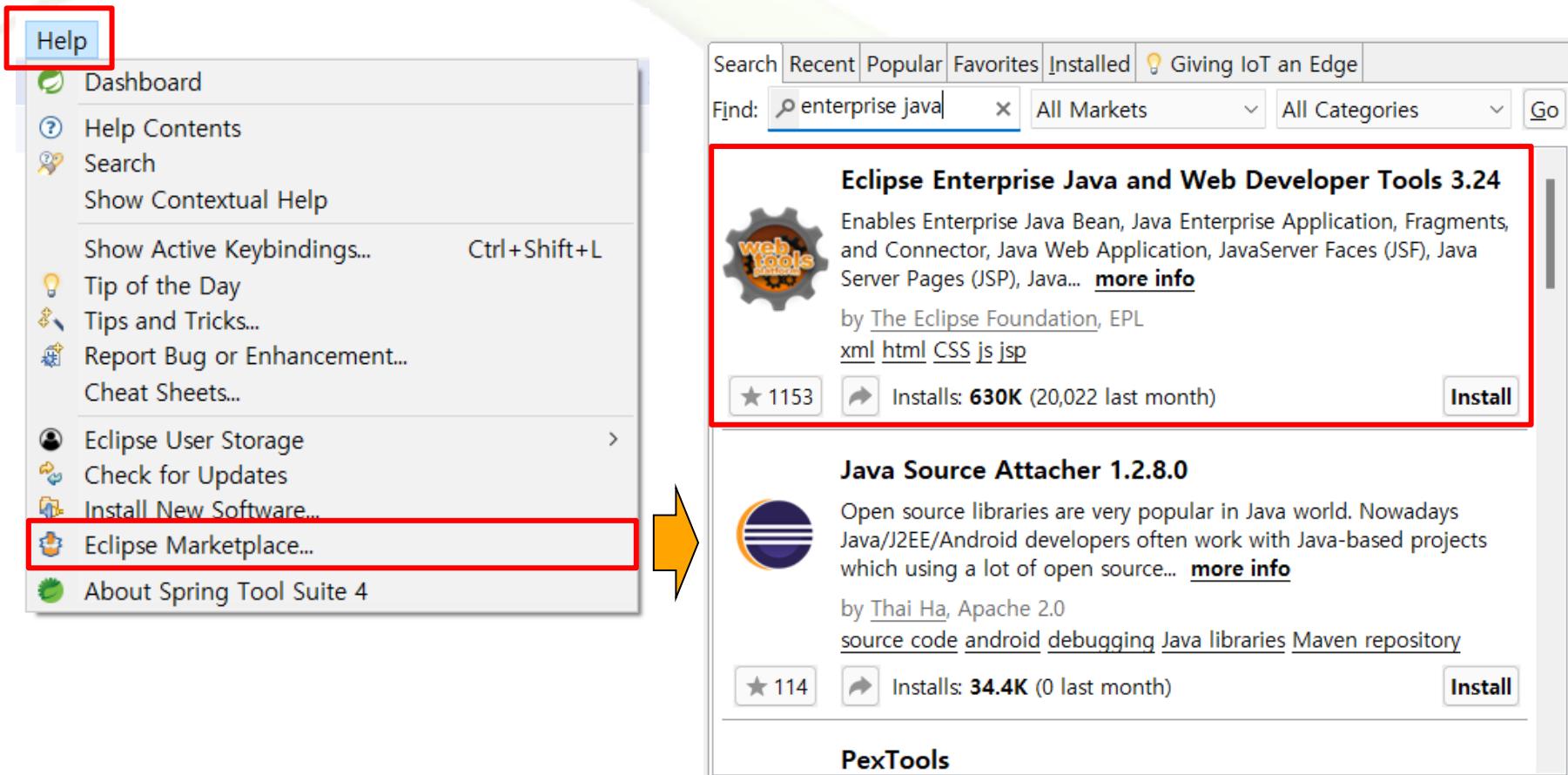
스프링 플러그인 설치 (STS, Eclipse 공통)

- Eclipse Marketplace에서 STS 검색 후 설치
 - Spring Tools 4



스프링 플러그인 설치 (STS)

- Eclipse Marketplace에서 STS 검색 후 설치
 - Eclipse Enterprise Java and Web Developer Tools



Eclipse (STS) 도구 설치

- download → <https://projectlombok.org/download>



Download 1.18.22

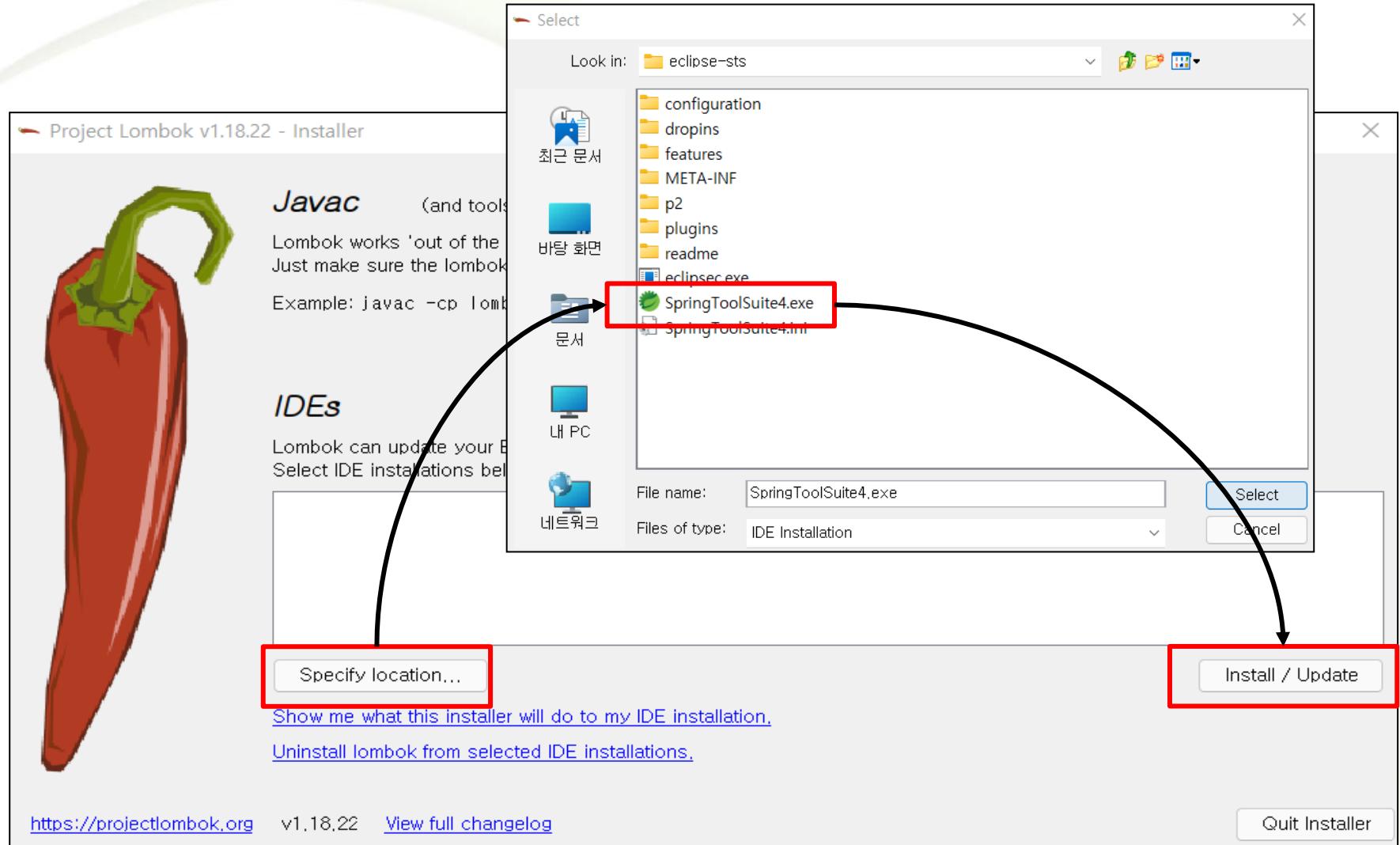
[changelog](#)

[older versions](#)

Feeling adventurous? Download the latest [snapshot](#) release.

Eclipse (STS) 도구 설치

- 다운로드 후 실행 (실행 실패 → 명령행에서 java -jar lombok.jar 실행)



Maven 설치

- 다운로드 → <http://maven.apache.org/download.cgi>

	Link	Checksums
Binary tar.gz archive	apache-maven-3.6.3-bin.tar.gz	apache-maven-3.6.3-bin.tar.gz.sha512
Binary zip archive	apache-maven-3.6.3-bin.zip	apache-maven-3.6.3-bin.zip.sha512
Source tar.gz archive	apache-maven-3.6.3-src.tar.gz	apache-maven-3.6.3-src.tar.gz.sha512
Source zip archive	apache-maven-3.6.3-src.zip	apache-maven-3.6.3-src.zip.sha512

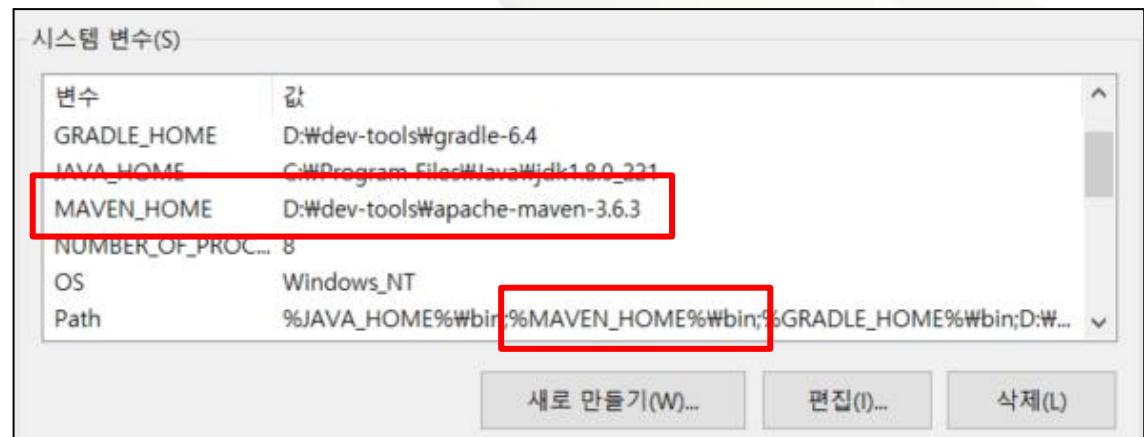
- 다운로드 파일 압축 해제

- 환경 변수 등록

- MAVEN_HOME
- Path : %MAVEN_HOME%\bin

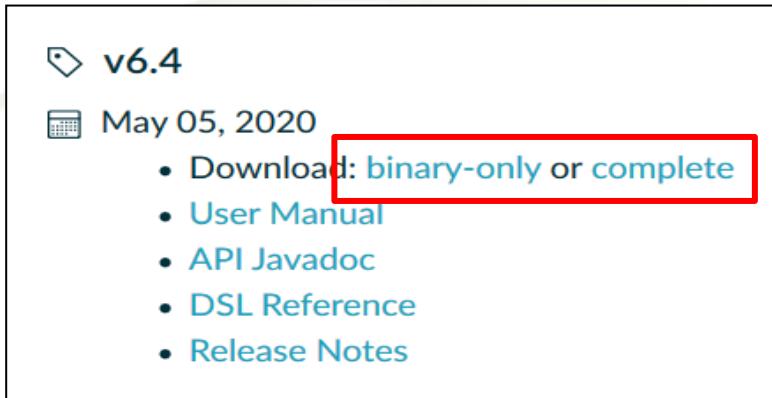
- 명령 프롬프트에서 확인

- mvn -version



Gradle 설치

- 다운로드 → <https://gradle.org/releases/>



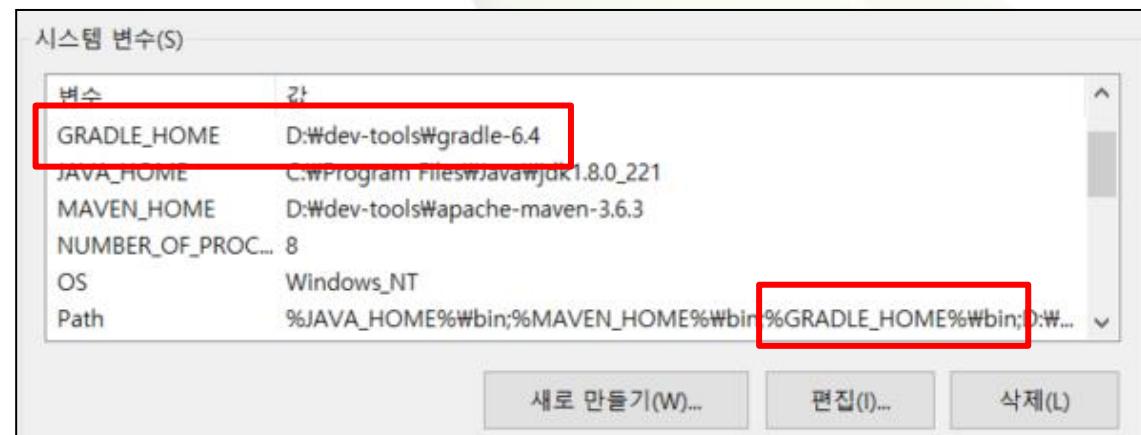
- 다운로드 파일 압축 해제

- 환경 변수 등록

- GRADLE_HOME
- Path : %GRADLE_HOME%\bin

- 명령 프롬프트에서 확인

- gradle -version



Apache Tomcat 설치

- 다운로드 → <https://tomcat.apache.org/download-90.cgi>

The screenshot shows the 'Binary Distributions' section of the Apache Tomcat download page. It lists several download options under the 'Core' category:

- Core:
 - [zip \(pgp, sha512\)](#) (highlighted with a red box)
 - [tar.gz \(pgp, sha512\)](#)
 - [32-bit Windows zip \(pgp, sha512\)](#)
 - [64-bit Windows zip \(pgp, sha512\)](#)
 - [32-bit/64-bit Windows Service Installer \(pgp, sha512\)](#)

- 다운로드 파일 압축 해제
- 설정 변경 (포트번호 변경)
 - 톰캣설치경로\conf\server.xml 편집

The screenshot shows a portion of the server.xml configuration file. The `<Connector>` element is highlighted, specifically the `port` attribute which is set to "8081".

```
<Connector port="8081" protocol="HTTP/1.1"
           connectionTimeout="20000"
           redirectPort="8443" />
```

MySQL 설치

- 다운로드 → <https://dev.mysql.com/downloads/windows/installer/8.0.html>

Windows (x86, 32-bit), MSI Installer <small>(mysql-installer-web-community-8.0.27.1.msi)</small>	8.0.27	2.3M	Download
Windows (x86, 32-bit), MSI Installer <small>(mysql-installer-community-8.0.27.1.msi)</small>	8.0.27	470.2M	Download

- 다운로드 파일 압축 해제
- 설정 변경 (인코딩 등)
 - 설치 경로에 my.ini 파일 생성 및 작성
 - 예시 파일 참고

MySQL 설정

▪ MYSQL_HOME/my.ini

 share	파일 폴더
 my.ini	구성 설정
 LICENSE	파일 271KB

```
[mysql]
default-character-set=utf8mb4

[client]
default-character-set=utf8mb4

[mysqld]

basedir=C:/workspaces/dev-tools/mysql-8.0.27-winx64
datadir=C:/workspaces/dev-tools/mysql-8.0.27-winx64/data

collation-server=utf8mb4_unicode_ci
character-set-server=utf8mb4
skip-character-set-client-handshake

# init-connect='SET NAMES utf8mb4'
# init-connect='SET collation_connection=utf8_general_ci'
```

MySQL 초기화

- MYSQL_HOME/bin에서 명령 실행

```
> mysqld --install  
  
> mysqld --initialize  
  
> net start mysql  
  
> mysql -u root -p  
  
### 패스워드 입력
```

- MYSQL_HOME/data/computer-name.err 파일 확인

```
[Warning] [MY-013746] [Server] A deprecated TLS version TLSv1 is enabled for channel mysql_main  
[Warning] [MY-013746] [Server] A deprecated TLS version TLSv1.1 is enabled for channel mysql_main  
[Note] [MY-010454] [Server] A temporary password is generated for root@localhost: j7,0xF=tbl#()  
[System] [MY-010116] [Server] C:\workspaces\dev-tools\mysql-8.0.27-winx64\bin\mysqld (mysqld 8.0.27) starting  
[System] [MY-013576] [InnoDB] InnoDB initialization has started.
```

- 패스워드 변경

```
mysql> alter user root@localhost identified by "mysql-password"  
  
mysql> exit
```

MariaDB 설치

- 다운로드 →

https://mariadb.org/download/?t=mariadb&p=mariadb&r=10.6.5&os=windows&cpu=x86_64&pkg=msi&m=yongbo

Download MariaDB Server

MariaDB Server is one of the world's most popular open source relational databases and is available in the standard repositories of all major Linux distributions. Look for the package mariadb-server using the package manager of your operating system. Alternatively you can use the following resources:

MariaDB Server **MariaDB Repositories** **Connectors**

MariaDB Server Version
MariaDB Server 10.6.5

Display older releases:

Operating System
Windows

Architecture
x86_64

Package Type
MSI Package

Download **Mirror**
Yongbok.net - South Korea

- 다운로드 완료 후 설치
- 설정 변경 (인코딩 등)
 - 설치 경로에 my.ini 파일 생성 및 작성
 - 예시 파일 참고

MySQL / MariaDB 클라이언트 도구 설치

- 다운로드 → <https://www.heidisql.com/download.php>
- 다운로드 파일 압축 해제

The screenshot shows the download page for HeidiSQL 11.3. The main title is "Download HeidiSQL 11.3, released on 2023-07-10". Below it, there are several download links:

- Installer, 32/64 bit combined (SHA1 checksum)** (highlighted with a red box)
- Portable version (zipped): 32 bit , 64 bit** (highlighted with a red box)
- Sourcecode
- Previous releases

- 설치 경로에서 실행 파일 실행

	파일 풀더	작성자	마지막 업데이트	크기	다운로드 수
	plugins				2
	qpl.txt			18KB	2
	heidisql.exe			9,010KB	2
	libcrypto-1_1-x64.dll			2,639KB	2
	libiconv-2.dll			1,651KB	2
	libintl-8.dll			670KB	2
	libmariadb.dll			1,050KB	2

프로젝트 만들기 실습

- Maven 프로젝트 만들기
- Gradle 프로젝트 만들기
- STS 또는 Eclipse에서 Maven과 Gradle 프로젝트를 만들고 코드 작성 및 실행
- Spring 프로젝트 템플릿을 사용해서 프로젝트를 만들고 코드 작성 및 실행

Introduction to Spring Framework

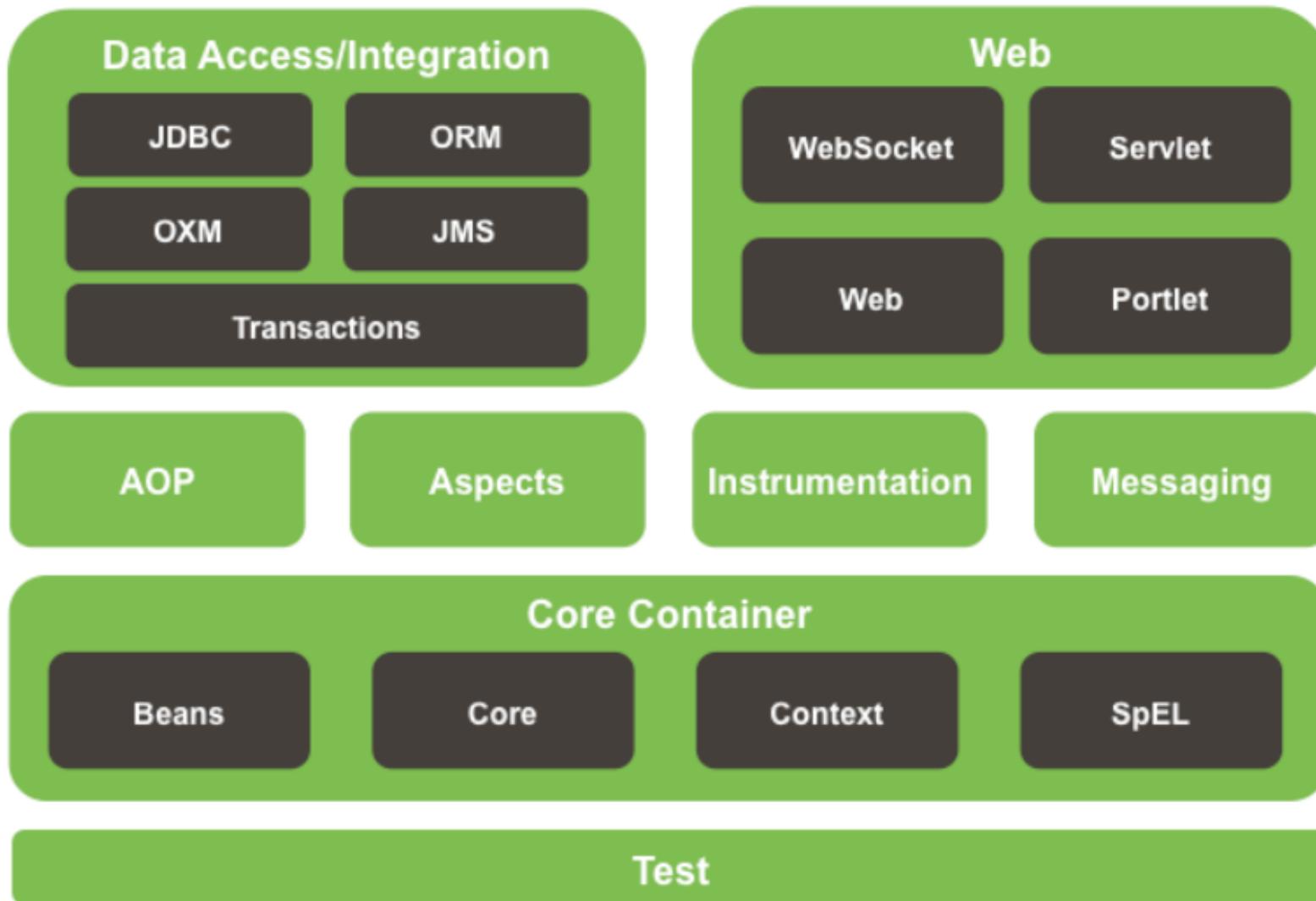
스프링 프레임워크

- 자바 플랫폼을 위한 오픈 소스 경량 프레임워크
 - 엔터프라이즈 애플리케이션 개발의 복잡도를 낮추고 효율성 향상
 - EJB와 같은 과거 기술의 실패 해결을 목표로 출현
- 동적 웹 사이트를 포함하여 엔터프라이즈 애플리케이션 개발을 위한 여러 가지 서비스 제공
- 우리나라 공공기관 웹 서비스 개발에 사용되는 전자정부 표준 프레임워크의 기반 기술.

스프링 프레임워크 기술 요소

- 제어 역전 (IoC) / 의존 관계 주입
 - 객체 의존성 관리의 핵심
 - 스프링 컨테이너가 효과적인 객체 생성 및 의존성 관리 기능 지원
- 관점 지향 프로그래밍 (AOP)
 - 변경 가능한 다수의 위치를 대상으로 하는 관심사를 관리하고 적용하는 기법
 - 프록시를 이용해서 구현
- Portable Service Abstraction (PSA)
 - 특정 인터페이스 구현 또는 클래스 상속에 종속되지 않고,
 - 평범한 자바 클래스를 기반으로 스프링 프레임워크 연동

스프링 프레임워크 - 구성 요소



제어 역전 컨테이너와 의존성 주입

인터페이스 다시 보기

- 의존성

- 클래스의 변경이 다른 클래스에 미치는 영향
- 한 클래스가 다른 클래스의 메서드를 사용할 때 의존성 발생

- 클래스 간의 직접적인 의존성은 유지 보수 측면에서 문제 유발

- 인터페이스는 클래스 간의 의존성을 제거할 수 있는 문법 자원
 - 클래스 간의 직접 호출을 사용하지 않고 인터페이스를 통해 연결

- 인터페이스를 사용하더라도 객체 생성 구문에서 발생하는 의존성은 남음

- 인터페이스는 참조 타입으로만 사용할 수 있고 new 연산자를 통해 인스턴스를 만들 수 없는 타입 → 결국 코드에 클래스 정보가 남아 의존성이 완전하게 제거되지 않음
- 인스턴스 생성 로직을 코드로부터 분리해서 관리할 수 있는 방법 필요

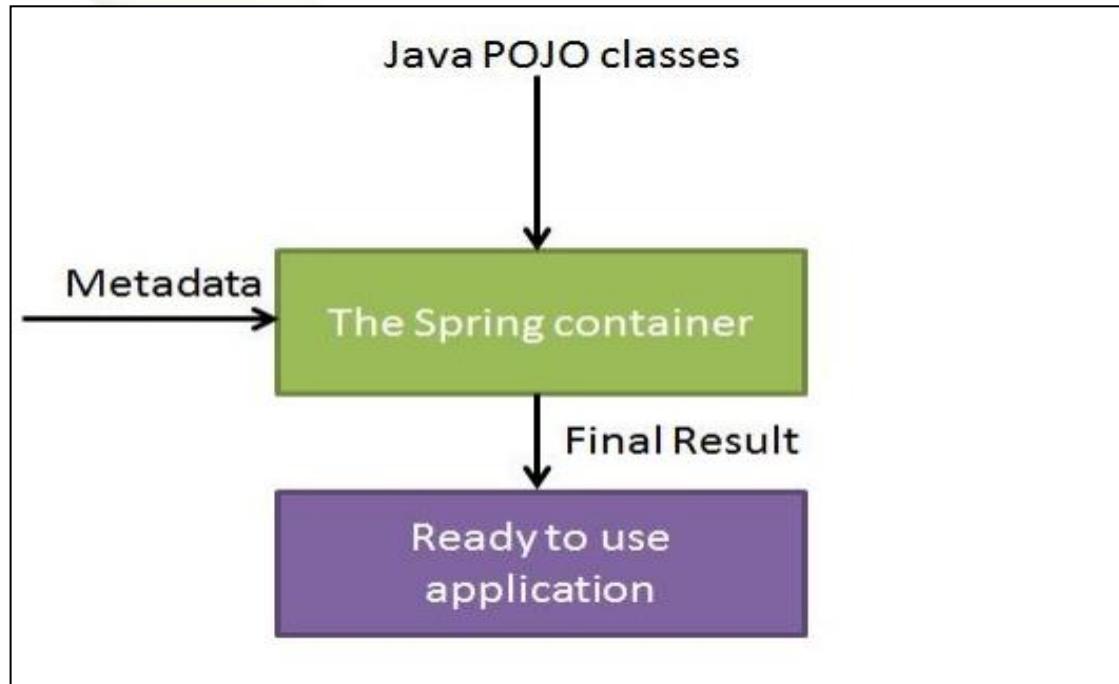
객체 팩토리 구현

리플렉션, 설정 파일 사용

→ 객체 생성 및 관리를 구현 코드로부터 분리

Spring IoC Container

- 스프링 애플리케이션에서는 객체의 생성, 의존성 관리, 사용, 제거 등의 작업을 코드 대신 독립된 컨테이너가 담당



- 구성요소

- 애플리케이션 컨텍스트
- 관리 대상 POJO 클래스 집합 → 스프링 빈
- 설정 메타 정보

IoC 컨테이너 종류

- Spring BeanFactory Container
 - 빈팩토리는 Bean 생성 및 DI 등의 Bean 관리에 집중하는 컨테이너
 - org.springframework.beans.BeanFactory 인터페이스 구현
- Spring ApplicationContext Container
 - 빈팩토리 기능에 다양한 엔터프라이즈 애플리케이션 개발 기능 추가 제공
 - org.springframework.context.ApplicationContext 인터페이스 구현
 - 스프링의 IoC 컨테이너는 일반적으로 애플리케이션 컨텍스트를 의미

BeanFactory 사용

```
public class MessageService {  
    private String message;  
  
    public void setMessage(String message) {  
        this.message = message;  
    }  
  
    public String getMessage() {  
        return String.format("Your  
    }  
}
```

```
<?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:context="http://www.springframework.org/schema/context"  
       xsi:schemaLocation="http://www.springframework.org/schema/beans  
                           https://www.springframework.org/schema/beans/spring-beans.xsd  
                           http://www.springframework.org/schema/context  
                           https://www.springframework.org/schema/context/spring-context.xsd">  
  
    <bean id="messageService" class="com.springexample.ioc.MessageService">  
        <property name="message" value="Hello, Spring IoC !!!!!" />  
    </bean>  
 </beans>
```

```
DefaultListableBeanFactory factory = new DefaultListableBeanFactory();  
XmlBeanDefinitionReader reader = new XmlBeanDefinitionReader(factory);  
reader.loadBeanDefinitions("app-context.xml");
```

```
MessageService messageService = factory.getBean("messageService", MessageService.class);  
  
String message = messageService.getMessage();  
  
System.out.println(message);
```

ApplicationContext 사용



```
GenericXmlApplicationContext context = new GenericXmlApplicationContext("app-context.xml");  
  
MessageService messageService2 = context.getBean("messageService", MessageService.class);  
  
String message2 = messageService2.getMessage();  
  
System.out.println(message2);  
  
context.close();
```

빈 등록 방법

- 설정 XML 파일의 <bean> 태그

```
<bean id="beanName" class="패키지경로.클래스이름">  
    생성자 주입 정보 또는 세터 주입 정보  
</bean>
```

- 자바 코드에 의한 빈 등록 (@Configuration, @Bean)

```
@Configuration public class AnnotatedHelloConfig {  
  
    @Bean public AnnotatedHello annotatedHello() {  
        return new AnnotatedHelloConfig();  
    }  
}
```

- 자동인식을 이용한 빈 등록

```
@Component("bean-name")  
public class AnnotatedHello { ... }
```

```
<context:component-scan base-package="pacakge-name" />
```

XML 설정 방식

```
@ComponentScan(base-packages="package-name")
```

코드 설정 방식

스프링 빈 주요 속성 목록

속성	설명
class	생성, 관리 대상 Bean Class (필수)
id	생성, 관리 대상 Bean 이름
scope	생성된 객체의 유지 범위
constructor-arg	생성자 전달인자 (의존성 주입 도구)
property	setter 메서드 (의존성 주입 도구)
autowiring mode	의존성 주입 자동화 설정 (명시적 설정 없이 의존성 주입)
lazy-initialization mode	객체의 생성 시점 설정 (프로그램 시작 vs 첫 번째 객체 요청)
initialization method	Bean 객체 생성 후 호출될 초기화 메서드
destruction method	컨테이너가 소멸될 때 호출될 메서드

의존성 주입 (DI, Dependency Injection)

- IoC 컨테이너가 객체 간의 의존성 관리를 위해 사용하는 구현 기법
 - 객체가 필요한 곳에 스프링 컨테이너가 객체를 자동으로 할당
 - 객체의 변경에 유연한 코드 구현 가능
 - **교재 75p ~ 80p 참고**
- 의존성 주입 방법
 - 생성자 주입 → 생성자 메서드를 사용해서 객체 할당
 - **교재 80p ~ 84p 참고**
 - 세터 주입 → setXxx(...) 메서드를 사용해서 객체 할당
 - **교재 84p ~ 88p 참고**
 - 기본 데이터 타입 설정 → 객체 의존성 주입과 같은 방법
 - **교재 88p ~ 90p 참고**

다중 설정 파일 사용

- 관리 대상 객체가 많아지면 하나의 설정 파일에 관리하는 것이 어려움
 - 영역별로 객체를 나누고 여러 개의 설정 파일에서 관리
- 구현 방법
 - IoC 컨테이너 (XxxApplicationContext 객체)를 만들 때 여러 개의 설정 파일을 제공 → 각 설정 파일의 객체는 @Autowired 어노테이션으로 서로 의존성 주입 가능
 - **교재 92p ~ 97p 참고**
 - Import 기능을 사용해서 다른 설정 파일의 내용을 포함
 - **교재 97p ~ 98p 참고**

getBean() 메서드 사용

▪ 교재 98p ~ 100p 참고

- IoC 컨테이너로부터 객체를 가져올 때 사용하는 메서드
- getBean("bean-id", bean-class) 형식
 - IoC 컨테이너가 관리하는 객체 중에서 id가 일치하는 객체 반환
 - 해당 bean-id로 관리되는 객체가 없을 경우 오류 발생
 - 반환되는 객체의 타입과 호환되지 않는 bean-class 를 지정한 경우 오류 발생
- getBean(bean-class) 형식
 - IoC 컨테이너가 관리하는 객체 중에서 타입이 bean-class와 호환되는 객체 반환
 - 컨테이너에 해당 타입과 호환되는 객체가 없을 경우 오류 발생
 - 컨테이너에 해당 타입과 호환되는 객체가 두 개 이상이 있을 경우 오류 발생

의존 객체 자동 주입

- IoC 컨테이너가 관리하는 객체 사이의 의존성을 코드를 통해 직접 주입하지 않고 암시적으로 주입하는 기법
- @Autowired, @Resource, @Inject 등의 어노테이션을 통해 암시적 의존성 주입
 - 필드(변수), setter 메서드, 생성자 등에 적용
 - **교재 106p ~ 113p 참고**
 - 호환 가능한 bean이 여러 개인 경우 오류 발생
 - @Qualifier 지정 또는 구체적 타입 지정 등을 통해 오류 해결
 - **교재 115p ~ 121p 참고**
 - 일치하는 bean이 없는 경우 오류 발생
 - required 속성을 false로 지정
 - Nullable 전달인자 또는 Optional 전달인자 사용
 - **교재 121p ~ 125p 참고**

의존 객체 자동 주입

- 일치하는 bean이 없는 경우 오류 발생 (계속)
 - @Autowired에 required=false를 지정한 경우 대상 bean이 없으면 의존성 주입을 수행하지 않음
 - @Nullable, Optional을 지정한 경우 대상 bean이 없으면 null값을 의존성 주입
 - **교재 125p ~ 127p 참고**
- 의존성 자동 주입과 명시적 의존성 주입이 동시에 적용된 경우
 - 의존성 자동 주입 설정이 적용됨
 - 특별한 이유가 없다면 일관된 방법을 사용하는 것이 권장됨
 - **교재 127p ~ 129p 참고**

의존 객체 자동 주입 설정 Annotation

- Spring Annotation
 - @Autowired : 자동 의존성 주입 설정
 - @Qualifier : 의존성 주입 대상 빈을 명시적으로 지정
- JSR-330
 - @Inject (Spring Annotation의 @Autowired)
 - @Named (Spring Annotation의 @Qualifier)
 - @Value : 직접 값 주입
- JSR-250
 - @Resource (Spring Annotation의 @Autowired)
 - @PostConstruct : 스프링 빈 정의의 init-method 속성과 같은 기능
 - @PreDestroy : 스프링 빈 정의의 destroy-method 속성과 같은 기능

Annotation Based Bean Configuration

- 스프링은 Annotation을 이용한 빈 정의 및 의존성 주입 설정 지원
- 컨테이너에 Annotation 기반 빈 설정 활성화 필요
 - annotation-config 설정
 - 이미 등록된 빈의 annotation (@Autowired, @Qualifier 등) 활성화

```
<context:annotation-config/>

<bean id="customerService" class="com.ensoa.order.service.CustomerServiceImpl" />
<bean id="customerRepository" class="com.ensoa.order.repository.CustomerRepositoryImpl"/>
```

- component-scan 설정
 - 빈 자동 등록 및 관련 annotation 활성화

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

```
@Configuration
@ComponentScan(basePackages = {"spring"})
```

- component-scan을 사용하는 경우 annotation-config는 불필요
 - **교재 132p ~ 139p 참고**

빈 정의 Annotation

▪ 교재 130p ~ 134p 참고

▪ @Component

- 클래스가 스프링 bean임을 표시하는 범용 Annotation

```
@Component("customerService")
public class CustomerServiceImpl implements CustomerService {
```

▪ @Service

- 업무 로직을 구현하는 서비스 bean을 표시 @Component

```
//Component("customerService")
@Service("customerService")
public class CustomerServiceImpl implements CustomerService {
```

▪ @Repository

- 데이터 접근 논리 구현 bean을 표시하는 @Component

```
//@Component("customerRepository")
@Repository("customerRepository")
public class CustomerRepositoryImpl implements CustomerRepository {
```

▪ @Controller

- Spring MVC 컨트롤러 클래스를 표시하는 @Component

스캔 대상에서 제외 ▪ 교재 135p ~ 138p 참고

- component-scan의 excludeFilters 속성으로 자동 등록 제외 대상 지정
- 필터 종류
 - 정규 표현식 사용 → FilterType.REGEX
 - AspectJ 패턴 사용 → FilterType.ASPECTJ

```
@Configuration
@ComponentScan(basePackages = {"spring", "spring2" },
    excludeFilters = {
        @Filter(type = FilterType.REGEX, pattern="spring\\..*Dao" ),
        @Filter(type = FilterType.ASPECTJ, pattern="spring.*Dao" ),
        @Filter(type = FilterType.ANNOTATION, classes = { ManualBean.class } ),
        @Filter(type = FilterType.ASSIGNABLE_TYPE, classes = { MemberDao.class } )
    })
public class AppCtxWithExclude {
```

```
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.8.13</version>
</dependency>
```

스캔 대상에서 제외 ▪ 교재 135p ~ 138p 참고

- 필터 종류 (계속)
 - Annotation 사용 → FilterType.ANNOTATION

```
@Configuration  
@ComponentScan(basePackages = {"spring", "spring2"},  
    excludeFilters = {  
        @Filter(type = FilterType.REGEX, pattern="spring\\..*Dao"),  
        @Filter(type = FilterType.ASPECTJ, pattern="spring.*Dao"),  
        @Filter(type = FilterType.ANNOTATION, classes = { ManualBean.class }),  
        @Filter(type = FilterType.ASSIGNABLE_TYPE, classes = { MemberDao.class })  
    })  
public class AppCtxWithExclude {
```

```
@ManualBean  
@Component  
public class MemberDao {
```

```
@Retention(RUNTIME)  
@Target(TYPE)  
public @interface ManualBean {  
}
```

스캔 대상에서 제외 ▪ 교재 135p ~ 138p 참고

- 필터 종류 (계속)

- 특정 타입 및 하위 타입 지정 → FilterType.ASSIGNABLE_TYPE

```
@Configuration
@ComponentScan(basePackages = {"spring", "spring2"} ,
    excludeFilters = {
        @Filter(type = FilterType.REGEX, pattern="spring\\..*Dao" ),
        @Filter(type = FilterType.ASPECTJ, pattern="spring.*Dao" ),
        @Filter(type = FilterType.ANNOTATION, classes = { ManualBean.class } ),
        @Filter(type = FilterType.ASSIGNABLE_TYPE, classes = { MemberDao.class } )})
})
public class AppCtxWithExclude {
```

Component Scan 과정에서 발생한 Bean 충돌 처리

- @Component로 등록한 2개 이상의 Bean이 암시적으로 같은 이름을 사용하는 경우 오류 발생 → 명시적으로 bean 이름을 다르게 지정해서 충돌을 피할 수 있음
- 수동으로 등록한 Bean과 @Component로 등록한 Bean이 같은 이름을 사용하는 경우 수동으로 등록한 Bean이 사용됨
- **교재 138p ~ 140p 참고**

IoC Container Life Cycle

```
//컨테이너 초기화
```

```
AnnotationConfigApplicationContext ctx =  
    new AnnotationConfigApplicationContext(AppContext.class);
```

```
//컨테이너 사용
```

```
Greeter g = ctx.getBean("greeter", Greeter.class);  
String msg = g.greet("스프링");  
System.out.println(msg);
```

```
//컨테이너 종료
```

```
ctx.close();
```

- 교재 141p ~ 142p 참고

Spring Bean Life Cycle

- Spring Bean의 생성 → 사용 → 소멸 과정의 중요한 시점마다 컨테이너가 적절한 메서드 호출.
- 이러한 콜백으로 객체의 Life Cycle 관리
- 주로 초기화 및 종료 시점의 이벤트로 활용
- 구현 방법
 - 인터페이스 구현
 - InitializingBean → 초기화, afterPropertiesSet
 - DisposableBean → 종료 처리, destroy
 - 커스텀 메서드
 - **교재 143p ~ 146p 참고**
 - initMethod, destroyMethod 속성 사용

Spring Bean Life Cycle

```
public class Client2 {  
  
    private String host;  
  
    public void setHost(String host) {  
        this.host = host;  
    }  
  
    public void connect() {  
        System.out.println("Client2.connect() 실행");  
    }  
  
    public void send() {  
        System.out.println("Client2.send() to " + host);  
    }  
  
    public void close() {  
        System.out.println("Client2.close() 실행");  
    }  
}
```

```
@Bean initMethod = "connect", destroyMethod = "close")  
public Client2 client2() {  
    Client2 client = new Client2();  
    client.setHost("host");  
    return client;  
}
```

```
<bean id="client2" class="spring.Client2"  
      init-method="connect" destroy-method="close">  
  
    <property name="host" value="host" />  
  
</bean>
```

Spring Bean Scope

- Spring bean에 지정된 scope에 따라 객체의 라이프사이클과 공유 범위가 결정됨.
- Scope 종류

Scope	설명
singleton	컨테이너 단위로 객체를 하나만 생성해서 모든 Bean들이 공유
prototype	객체의 요청이 있을 때마다 새로운 객체 생성
request	웹 애플리케이션의 경우 요청 라이프사이클 범위
session	웹 애플리케이션의 경우 세션 라이프사이클 범위

- IoC 컨테이너는 prototype scope bean의 소멸은 관리하지 않음 → 소멸 처리는 직접 구현

Spring Bean Scope (Singleton)

```
@Scope("singleton")
public Client2 client2() {
    Client2 client = new Client2();
    client.setHost("host");
    return client;
}

<bean id="client2" class="spring.Client2" scope="singleton">
    <property name="host" value="host" />
</bean>
```



```
public static void main(String[] args) throws IOException {
    AbstractApplicationContext ctx =
        new AnnotationConfigApplicationContext(AppCtxWithPrototype.class);

    Client2 client1 = ctx.getBean(Client2.class);
    Client2 client2 = ctx.getBean(Client2.class);
    System.out.println("client1 == client2 : " + (client1 == client2));

    ctx.close();
}
```

true

Spring Bean Scope (prototype)

```
@Scope("prototype")
public Client2 client2() {
    Client2 client = new Client2();
    client.setHost("host");
    return client;
}

<bean id="client2" class="spring.Client2" scope="prototype">
    <property name="host" value="host" />
</bean>
```



```
public static void main(String[] args) throws IOException {
    AbstractApplicationContext ctx =
        new AnnotationConfigApplicationContext(AppCtxWithPrototype.class);

    Client2 client1 = ctx.getBean(Client2.class);
    Client2 client2 = ctx.getBean(Client2.class);
    System.out.println("client1 == client2 : " + (client1 == client2));

    ctx.close();
}
```

false

스프링 AOP

AOP (Aspect Oriented Programming) ■ 교재 158p 참고

- 횡단 관심사 (crosscutting concerns)를 구현하는 도구

횡단 관심사는 시스템의 다른 부분에 의존하거나 영향을 미쳐야 하는 프로그램의 일부분. 이 관심사들은 디자인과 구현 면에서 시스템의 나머지 부분으로부터 깨끗하게 분해되지 못하는 경우가 있을 수 있으며 분산(코드 중복)되거나 얹히는(시스템 간의 상당한 의존성 존재) 일이 일어날 수 있다.

- 동일한 구현을 효과적으로 다수의 객체에 적용할 수 있는 방법

- 여러 객체에 공통으로 적용할 기능을 분리해서 재사용성을 높여주는 프로그래밍 기법
- 핵심 기능과 공통 기능의 구현 분리

- 구현 방법

- 컴파일 시점에 코드에 공통 기능 삽입
- 클래스 로딩 시점에 바이트 코드에 공통 기능 삽입
- 런타임에 프록시 객체를 생성해서 공통 기능 삽입

AOP 주요 용어 ▪ 교재 159p 참고

용어	설명
Advice	Joinpoint에 적용할 코드 실행 시점에 따라 Before Advice, After Advice 등으로 구현
Joinpoint	애플리케이션 실행의 특정 지점 횡단 관심사를 적용하는 구체적인 위치 표시
Pointcut	여러 Joinpoint의 집합으로 Advice를 실행하는 위치 표시
Aspect	Advice와 Pointcut을 조합해서 횡단 관심사에 대한 코드와 그것을 적용할 지점을 정의한 것
Weaving	Aspect를 적용하는 과정 프록시 기반 구현 등의 AOP 구현 방식이 구분되는 기준

스프링 AOP ▪ 교재 160p 참고

▪ 구현 가능한 Advice 종류

종류	설명
Before Advice	메서드 호출 전 공통 기능 수행
After Returning Advice	메서드가 정상적으로 반환한 후 공통 기능 수행
After Throwing Advice	메서드 실행 중 예외가 발생하는 경우 공통 기능 수행
After Advice	예외 발생 여부와 상관 없이 메서드 실행 후 공통 기능 수행
Around Advice	메서드 실행 전, 후 또는 예외 발생 시점에 공통 기능 수행

스프링 AOP

- 완전한 AOP 기능을 제공하지 않음
 - JEE 애플리케이션 구현에 필요한 수준의 기능만 제공
 - 프록시 기반의 동적 AOP 지원
 - 메서드 호출 Joinpoint 지원
- 지원하는 구현 방식
 - XML 설정 기반 POJO 클래스를 이용한 AOP 구현
 - @Aspect, @Pointcut, @Around 등 Annotation 기반 AOP 구현

스프링 AOP 의존성 패키지 정의 (maven build 설정)

- spring-aop와 aspectjweaver 패키지 필요 ▪ 교재 151 ~ 152p 참고

```
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.2.8.RELEASE</version>
    </dependency>
    <dependency>
        <groupId>org.aspectj</groupId>
        <artifactId>aspectjweaver</artifactId>
        <version>1.9.6</version>
    </dependency>
</dependencies>
```

spring-context 패키지에 대한 의존성을 추가하면 spring-aop 패키지 의존성이 자동으로 추가됨

XML 기반 AOP 설정

```
public class ExeTimeAspect {  
    public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {  
        long start = System.nanoTime();  
        try {  
            Object result = joinPoint.proceed();  
            return result;  
        } finally {  
            long finish = System.nanoTime();  
            Signature sig = joinPoint.getSignature();  
            System.out.printf("%s.%s(%s) 실행 시간 : %d ns\n",  
                joinPoint.getTarget().getClass().getSimpleName(),  
                sig.getName(), Arrays.toString(joinPoint.getArgs()),  
                (finish - start));  
        }  
    }  
}
```

Advice 클래스 및 메서드 정의

```
<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-4.3.xsd">
```

AOP Namespace 추가

```
<bean id="exeTimeAdvice" class="aspect.ExeTimeAspect" />
```

AOP 설정 (pointcut, advice, aspect)

```
<aop:config> AOP 설정 영역 지정
```

```
    <aop:aspect id="exeTimeAspect" ref="exeTimeAdvice"> Aspect 설정
```

```
        <aop:pointcut id="publicTarget" expression="execution(public * chap07..*(..))" /> Pointcut 설정
```

```
        <aop:around method="measure" pointcut-ref="publicTarget" /> Advice 설정
```

```
    </aop:aspect>
```

```
</aop:config>
```

Annotation 기반 AOP 설정 • 교재 160 ~ 165p 참고

@Aspect Aspect 설정

```
public class ExeTimeAspect {
```

@Pointcut("execution(public * chap07..*(..))") Pointcut 설정

```
private void publicTarget() {  
}
```

@Around("publicTarget()") Advice 설정

```
public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {  
    long start = System.nanoTime();  
    try {  
        Object result = joinPoint.proceed();  
        return result;  
    } finally {  
        long finish = System.nanoTime();  
        Signature sig = joinPoint.getSignature();  
        System.out.printf("%s.%s(%s) 실행 시간 : %d ns\n",  
            joinPoint.getTarget().getClass().getSimpleName(),  
            sig.getName(), Arrays.toString(joinPoint.getArgs()),  
            (finish - start));  
    }  
}
```

AOP Annotation 활성화

@Configuration

@EnableAspectJAutoProxy

```
public class AppCtx {
```

Aspect bean 등록

@Bean

```
public ExeTimeAspect exeTimeAspect() {  
    return new ExeTimeAspect();  
}
```

XML 설정과 Annotation 비교

Annotation	XML 설정
@Aspect	<aop:aspect>
@Pointcut	<aop:pointcut>
@Before	<aop:before>
@After	<aop:after>
@AfterReturning	<aop:afterReturning>
@AfterThrowing	<aop:afterThrowing>
@Around	<aop:around>

JoinPoint 전달인자

▪ 교재 165 ~ 166p 참고

- Advice 메서드의 JoinPoint 전달인자를 이용해서 호출된 메서드 정보 접근
- Around Advice 메서드는 ProceedingJoinPoint 형식의 전달인자를 필수적으로 사용
- Around Advice 이외의 메서드는 JoinPoint 형식의 전달인자를 선택적으로 사용

```
public class ExeTimeAspect {  
  
    public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {  
        long start = System.nanoTime();  
        try {  
            Object result = joinPoint.proceed();  
            return result;  
        } finally {  
            long finish = System.nanoTime();  
            Signature sig = joinPoint.getSignature();  
            System.out.printf("%s.%s(%s) 실행 시간 : %d ns\n",  
                joinPoint.getTarget().getClass().getSimpleName(),  
                sig.getName(), Arrays.toString(joinPoint.getArgs()),  
                (finish - start));  
        }  
    }  
}
```

프록시 생성 방식

▪ 교재 166 ~ 168p 참고

- 스프링은 Advice가 적용되는 타겟 클래스가 인터페이스를 구현한 경우 타겟 클래스 상속 타입이 아닌 인터페이스 구현 타입의 프록시 객체를 반환
- @EnableAspectJAutoProxy의 proxyTargetClass 속성을 true로 설정하면 스프링은 프록시 객체를 인터페이스 구현 타입이 아닌 타겟 클래스를 상속한 타입으로 구현

```
@Configuration  
@EnableAspectJAutoProxy(proxyTargetClass = true)  
public class AppCtxWithClassProxy {  
    @Bean  
    public ExeTimeAspect exeTimeAspect() {  
        return new ExeTimeAspect();  
    }  
}
```

```
AnnotationConfigApplicationContext ctx =  
    new AnnotationConfigApplicationContext(AppCtxWithClassProxy.class);  
  
RecCalculator cal = ctx.getBean("calculator", RecCalculator.class);  
long fiveFact = cal.factorial(5);  
System.out.println("cal.factorial(5) = " + fiveFact);  
System.out.println(cal.getClass().getName());  
ctx.close();
```

Pointcut 구문 종류

지명자	설명
execution()	메서드를 조인포인트 매치 (메서드 패턴)
within()	타입 범위로 조인포인트 매치 (패키지 또는 클래스 패턴)
bean()	빈 이름으로 조인포인트 매치
target()	특정 타입을 대상으로 조인포인트 매치 (명시적 타입)
this()	AOP 프록시 빈 인스턴스를 대상으로 조인포인트 매치
args()	전달인자가 해당 타입인 메서드에 조인포인트 매치

Pointcut 구문 형식

▪ 교재 168 ~ 169p 참고

▪ 기본형식

- execution(수식어패턴? 리턴타입패턴 클래스이름패턴?메서드이름패턴(전달인자패턴))
- bean(Bean 이름)
- within(클래스 또는 패키지 패턴)

▪ 와일드카드

- ?는 선택적 사용 항목
- *는 All
- ..은 0 or more

▪ 두 개 이상의 Pointcut을 and, or, not, &&, ||, ! 으로 조합 및 수식 가능

Pointcut 구문 사용 사례

▪ 교재 168 ~ 169p 참고

```
execution(public void set*(..))
```

public 접근성, void 반환, 이름이 set으로 시작, 전달인자 0개 이상

```
execution(* com.example.springaop.*.*())
```

com.example.springaop 패키지의 모든 클래스의 전달인자 없는 모든 메서드

```
execution(* com.example.springaop..*.*(..))
```

com.example.springaop 및 하위 패키지 내 모든 클래스의 모든 메서드

```
execution(* com.example.springaop..ClassName.method(..))
```

com.example.springaop 패키지 및 모든 하위 패키지 내의 ClassName
클래스의 모든 오버로딩된 이름이 method인 메서드

Advice 적용 순서 ▪ 교재 169 ~ 175p 참고

- 동일한 PointCut에 여러 개의 Advice가 적용될 경우 호출 순서는 상황에 따라 다를 수 있음.
- 필요한 경우 @Order Annotation으로 @Advice의 적용 순서를 명시적으로 설정할 수 있음

```
@Aspect  
@Order(1)  
public class ExeTimeAspect {
```

```
@Aspect  
@Order(2)  
public class CacheAspect {
```

```
<aop:aspect id="exeTimeAspect" ref="exeTimeAdvice" order="1">  
    <aop:pointcut id="publicTarget" expression="execution(public * chap07..*(..))" />  
    <aop:around method="measure" pointcut-ref="publicTarget" />  
</aop:aspect>  
  
<aop:aspect id="cacheAspect" ref="cacheAdvice" order="2">  
    <aop:pointcut id="cacheTarget" expression="execution(public * chap07..*(long))" />  
    <aop:around method="execute" pointcut-ref="cacheTarget" />  
</aop:aspect>
```

@Pointcut 재사용

```
@Aspect  
public class CacheAspect2 {  
  
    private Map<Long, Object> cache = new HashMap<>();  
  
    @Around("aspect2.CommonPointcut.commonTarget()")  
    public Object execute(ProceedingJoinPoint joinPoint) throws Throwable {
```

```
public class CommonPointcut {  
  
    @Pointcut("execution(public * chap07..*(..))")  
    public void commonTarget() {  
    }  
}
```

```
@Aspect  
public class ExeTimeAspect2 {  
  
    @Around("aspect2.CommonPointcut.commonTarget()")  
    public Object measure(ProceedingJoinPoint joinPoint) throws Throwable {
```

스프링 MVC

Quickstart (프로젝트 생성)

교재 232 ~ 235p 참고

- 웹 프로젝트에 필요한 폴더 (강조된 항목은 웹에 추가되는 폴더)
 - src/main/java
 - src/main/resources
 - **src/main/webapp/WEB-INF/lib**
 - **src/main/webapp/WEB-INF/views**
- pom.xml 파일에 추가되는 항목

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>sp5</groupId>
  <artifactId>sp5-chap09</artifactId>
  <version>0.0.1 SNAPSHOT</version>
  <packaging>war</packaging>
```

```
<dependencies>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>javax.servlet.jsp-api</artifactId>
    <version>2.3.2-b02</version>
    <scope>provided</scope>
  </dependency>

  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
  </dependency>

  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.0.2.RELEASE</version>
  </dependency>
</dependencies>
```

- 톰캣 설치

- 톰캣 다운로드 → 압축풀기
- conf/server.xml 파일 수정 (port : 8080 → 8081)

- 이클립스 톰캣 등록

- 주메뉴 Window → Preferences
- Server → Runtime Environments → Add
- 단계별 마법사에서 다운로드한 톰캣 설치 경로 지정

- 서버 등록

- JavaEE or Spring Perspective → Server Tab → New (서버 등록)

설치 상세 내역은 별도 톰캣 설치 파일 참고

Quickstart (설정)

- web.xml (xml based spring configuration)

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-context.xml</param-value>
</context-param>

<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

루트 웹 애플리케이션 컨텍스트 설정

```
<servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

프론트 컨트롤러 서블릿 설정

```
<servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

서블릿 웹 애플리케이션 컨텍스트 설정

Quickstart (설정)

- web.xml (xml based spring configuration)
 - Root Web Application Context
 - 웹 애플리케이션이 시작될 때 최상위 전역 웹 애플리케이션 컨텍스트 생성
 - 모든 서블릿에서 공통으로 사용되는 빈을 설정하는 용도로 사용
 - Servlet Web Application Context
 - 서블릿이 처음 요청될 때 웹 애플리케이션 컨텍스트 생성
 - 이후 해당 서블릿에 대한 요청이 발생할 때 동작

Quickstart (설정)

교재 239 ~ 240p 참고

- web.xml (code based spring configuration)

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextClass</param-name>
        <param-value>
            org.springframework.web.context.support.AnnotationConfigWebApplicationContext
        </param-value>
    </init-param>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            config.MvcConfig
            config.ControllerConfig
        </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

FrontController Servlet 등록
IoC Container 설정

Quickstart (설정)

- servlet-context.xml 파일 (xml based spring web mvc configuration)

@Controller, @RequestMapping 활성화

```
<!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->
<!-- Enables the Spring MVC @Controller programming model -->
<annotation-driven /> Static path 설정

<!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources -->
<resources mapping="/resources/**" location="/resources/" />

<!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
</beans:bean>

<context:component-scan base-package="chap09" />
```

ViewResolver 설정

Quickstart (설정)

교재 237 ~ 239p 참고

- MvcConfig.java (code based spring web mvc configuration)

Spring Web MVC 설정 활성화 → 내부적으로 다양한 Bean 활성화

```
@Configuration  
@EnableWebMvc  
public class MvcConfig implements WebMvcConfigurer {  
  
    @Override  
    public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {  
        configurer.enable();  
    }  
  
    @Override  
    public void configureViewResolvers(ViewResolverRegistry registry) {  
        registry.jsp("/WEB-INF/view/", ".jsp");  
    }  
  
    @Override  
    public void addResourceHandlers(ResourceHandlerRegistry registry) {  
        registry.addResourceHandler("/resources/**")  
            .addResourceLocations("/resources/")  
            .setCachePeriod(3600)  
            .resourceChain(true);  
    }  
}
```

리소스 처리 서블릿 설정

ViewResolver 설정

Static path 설정

Quickstart (설정)

- web.xml
 - 한글 지원을 위해 Character Encoding Filter 적용

```
<filter>
    <filter-name>characterEncodingFilter</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>
    <init-param>
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>characterEncodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Quickstart (Controller + View 구현)

교재 241 ~ 245p 참고

▪ HelloController.java

```
@Controller  
public class HelloController {  
  
    @GetMapping("/hello")  
    public String hello(Model model,  
                        @RequestParam(value = "name", required = false) String name) {  
  
        model.addAttribute("greeting", "안녕하세요, " + name);  
  
        return "hello";  
    }  
}
```

▪ hello.jsp

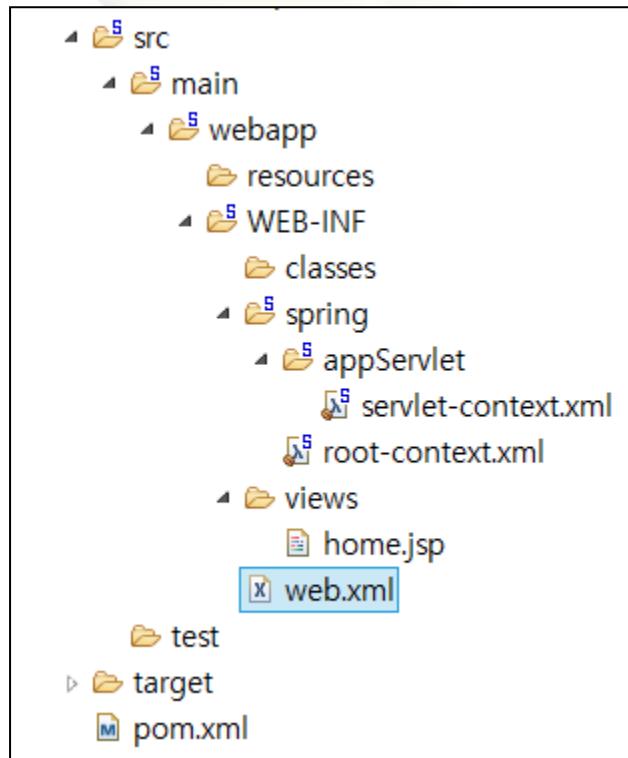
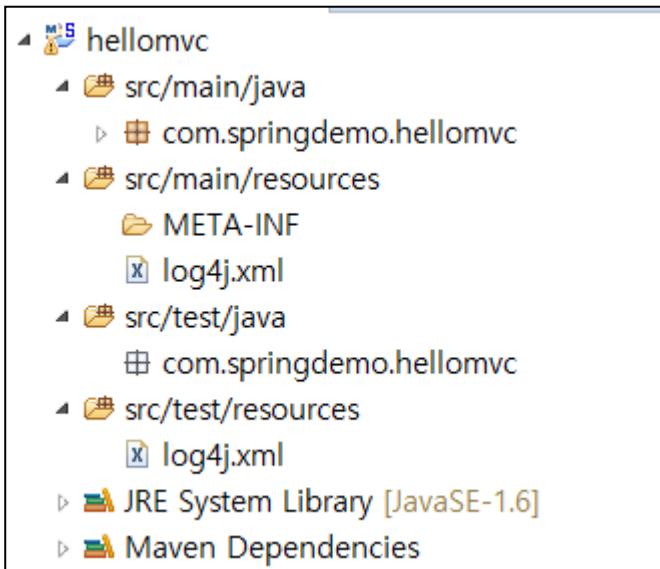
```
<%@ page contentType="text/html; charset=utf-8" %>  
  
<!DOCTYPE html>  
<html>  
    <head>  
        <title>Hello</title>  
    </head>  
    <body>  
        인사말: ${greeting}  
    </body>  
</html>
```

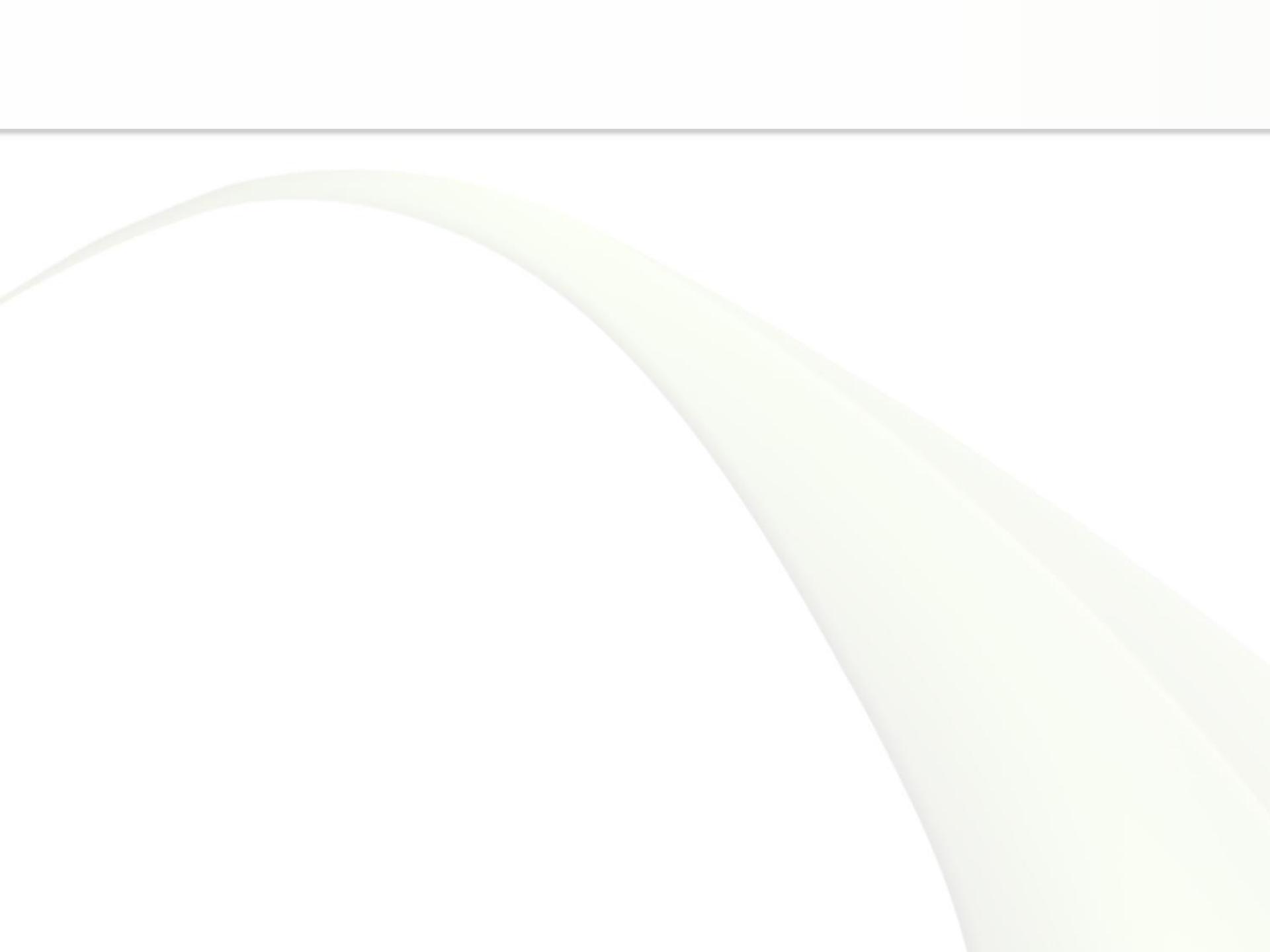
Quickstart (실행)

교재 245 ~ 248p 참고

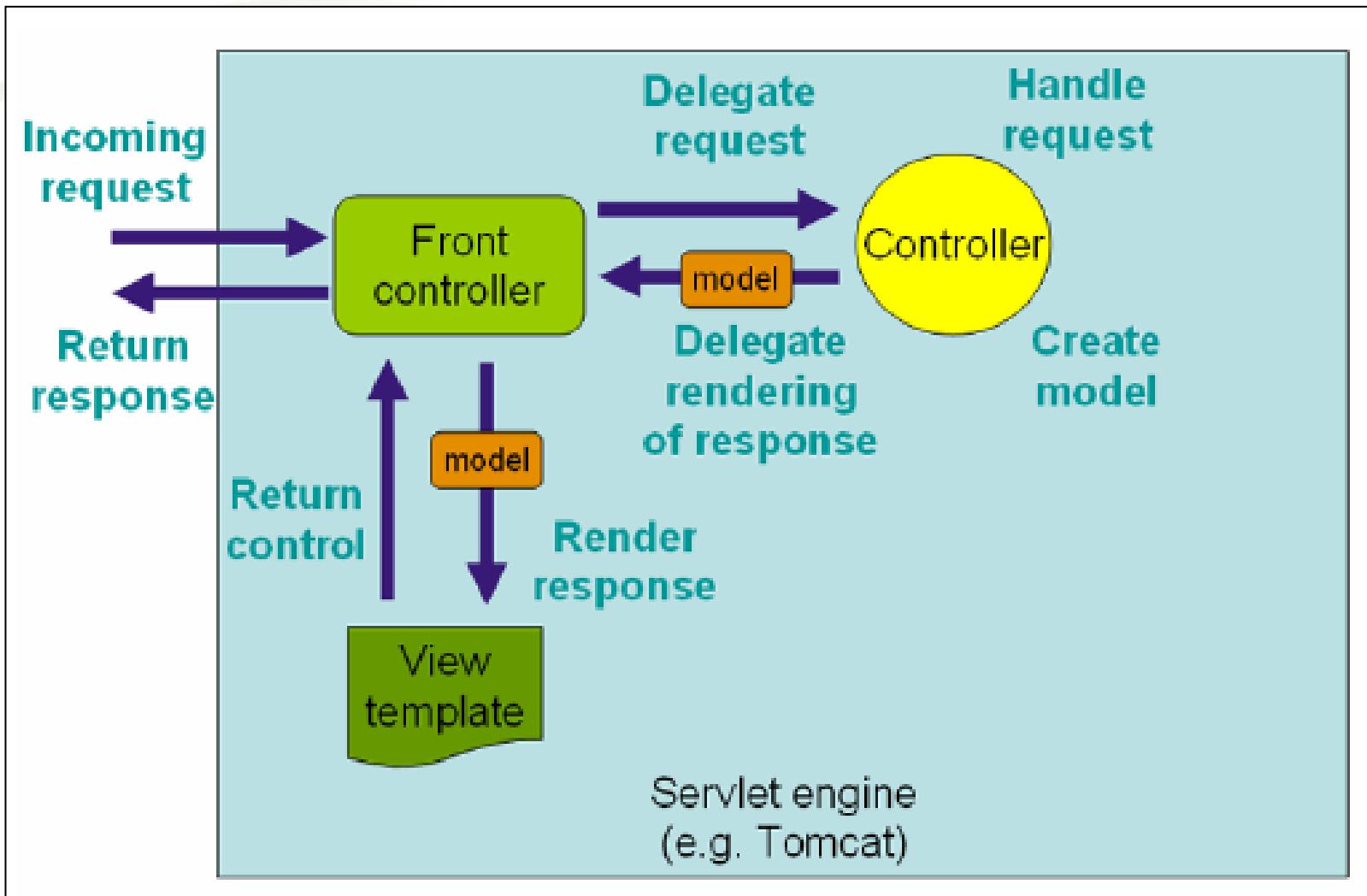
Legacy Template으로 프로젝트 생성

- 프로젝트 만들기
 - File → New → Spring Legacy Project 메뉴 항목 선택
 - 프로젝트 이름 입력 / Templates에서 Spring MVC Project 선택 후 Next
 - 패키지 이름 입력 후 Finish
- 프로젝트 구조 확인

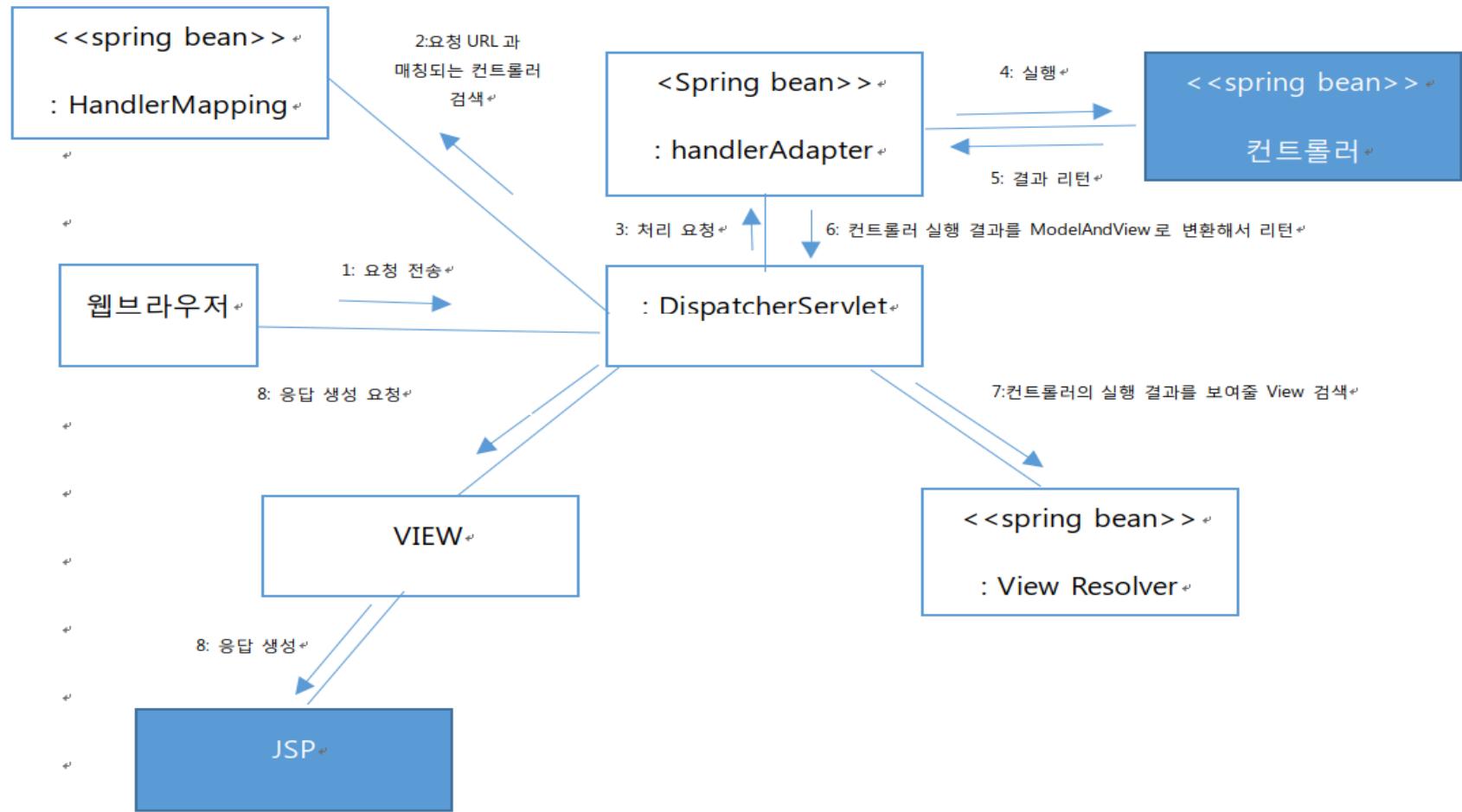




FrontController Based MVC Pattern



Spring MVC 교재 249 ~ 262p 참고



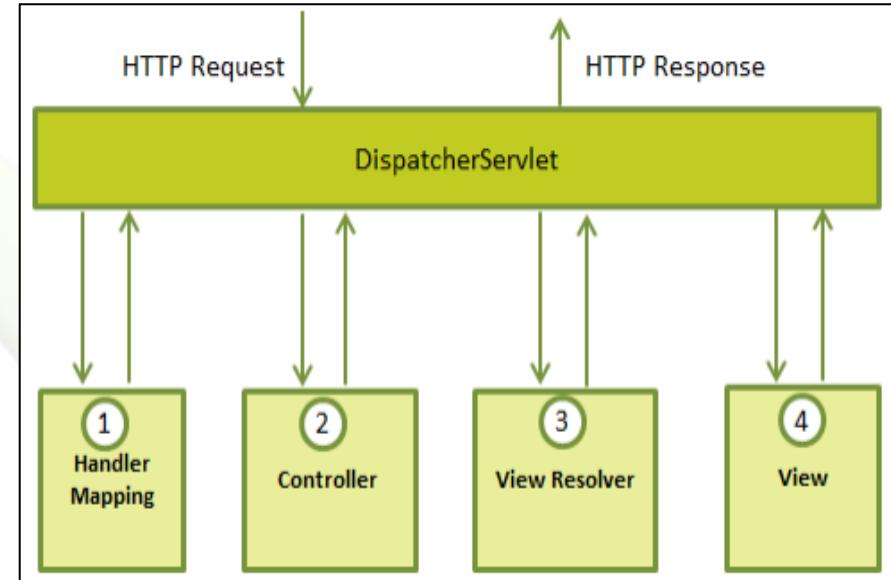
WebApplicationContext

스프링 MVC

- Front Controller Pattern 기반 구현
- DispatcherServlet을 FrontController 구현체로 제공
- DispatcherServlet 생성과 함께 WebApplicationContext 객체 생성
- web.xml 및 스프링 bean 설정 파일의 내용에 따라 동작

요청 처리 과정 요약

- DispatcherServlet 요청 수신
- DispatcherServlet은 HandlerMapping을 통해 적절한 Controller를 선택하고 Controller에 요청 위임
- Controller는 Model 영역의 객체 생성 및 호출 후 Model 객체 생성 / 데이터 할당 / 뷰이름 지정한 후 FrontController에 반환
 - 일반적으로 ModelAndView 형식의 객체 사용해서 결과 반환
- DispatcherServlet은 Controller의 반환 값에 따라 ViewResolver를 이용해서 View를 결정하고 호출
- 호출된 View는 전달된 Model 객체를 이용해서 화면을 구성하고 반환
- DispatcherServlet은 View의 반환 결과를 요청 영역에 응답



스프링 컨트롤러

- DispatcherServlet에서 전달된 개별 요청을 처리하는 객체
- 스프링 IoC 컨테이너에서 관리
- 스프링 3.0 버전부터 Annotation 기반 컨트롤러 클래스 구현 권장

컨트롤러 구현

교재 269 ~ 272p 참고

■ 요청 맵핑

@Controller 어노테이션으로 빈 설정 등록

```
@Controller  
public class RegisterController {  
  
    @RequestMapping("/register/step1")  
    public String handleStep1() {  
        return "register/step1";  
    }  
}
```

@RequestMapping 어노테이션으로 요청과 컨트롤러 맵핑

```
@Controller  
  
@RequestMapping("/register")  
public class RegisterController {  
  
    @RequestMapping("/step1")  
    public String handleStep1() {  
        return "register/step1";  
    }  
}
```

/WEB-INF/views/register.jsp 페이지를 뷰로 사용
(/WEB-INF/views + register/step1 + .jsp)

컨트롤러 구현

교재 273 ~ 274p 참고

■ 전송 방식 제어

- Get, Post 등 전송 방식을 구분해서 요청 컨트롤러 매핑 설정 가능



■ redirect 처리

```
@GetMapping("/register/step2")
public String handleStep2Get() {
    return "redirect:/register/step1";
}
```

요청 맵핑 구성 요소

- web.xml 파일에 등록된 DispatcherServlet의 서블릿 맵핑 url

```
<servlet-mapping>
  <servlet-name>customer</servlet-name>
  <url-pattern>/customer/*</url-pattern>
</servlet-mapping>
```

- @Controller 어노테이션이 지정된 컨트롤러 클래스의 @RequestMapping에 지정된 경로

- 생략되면 @RequestMapping(value="/")

```
@Controller
@RequestMapping(value="/")
public class CustomerController {
```

- 컨트롤러 클래스에 포함된 메서드의 @RequestMapping에 지정된 경로

```
@RequestMapping(value="edit.do", method=RequestMethod.GET)
public String edit(Model model) {
```

- 최종 경로

```
http://.../customer/edit.do
```

컨트롤러 구현

■ 요청 데이터 읽기

- 클라이언트가 전달하는 요청 데이터, 헤더 등의 정보를 수신할 수 있도록 다양한 전달인자 형식을 통해 데이터 전달

■ 요청 데이터 매팅 전달인자 종류

종류	설명
모델	<ul style="list-style-type: none">Model, ModelMap, Map컨트롤러에서 데이터를 저장하고 뷰로 전달되는 용도의 전달인자 (클라이언트가 전달하는 데이터 수신 기능은 없음)
@ModelAttribute	<ul style="list-style-type: none">사용자 정의 객체 모델 지정 (DTO를 이용한 데이터 수신)어노테이션 생략 가능
@PathVariable	<ul style="list-style-type: none">@RequestMapping에 지정된 URL 중 {}에 명시된 경로 변수<code>@RequestMapping("/customer/{name} ") handler(@RequestParam("name") String name) { ...}</code>

컨트롤러 구현

■ 요청 데이터 매팅 전달인자 종류 (계속)

종류	설명
@RequestParam	<ul style="list-style-type: none">개별 Http 요청 패러미터 저장하는 전달인자 지정어노테이션 생략 가능모든 요청을 일괄 수신하기 위해 Map<String, String> 사용
@RequestBody	<ul style="list-style-type: none">Http 요청의 본문을 저장하는 전달인자 지정
HttpServletRequest, HttpServletResponse	<ul style="list-style-type: none">일반 서블릿에 전달되는 요청, 응답 객체
HttpSession	<ul style="list-style-type: none">일반 서블릿에 전달되는 세션 객체
Locale	<ul style="list-style-type: none">Locale Resolver가 결정한 Locale 정보
스트림	<ul style="list-style-type: none">InputStream, Reader, OutputStream, Writer요청 및 응답에 대응하는 저수준 스트림 객체
@RequestHeader	<ul style="list-style-type: none">Http 헤더 정보를 전달인자에 매팅
@Cookievalue	<ul style="list-style-type: none">Http 쿠키 값을 전달인자에 매팅

■ 요청 데이터 읽기

```

<body>
    <h2>약관</h2>
    <p>약관 내용</p>
    <form action="step2" method="post">
        <label>
            <input type="checkbox" name="agree" value="true"> 약관 동의
        </label>
        <input type="submit" value="다음 단계" />
    </form>
</body>

```

```

// @RequestMapping("/register/step2", method = RequestMethod.POST)
@PostMapping("/register/step2")
public String handleStep2(
    @RequestParam(value = "agree", defaultValue = "false") Boolean agree,
    Model model) {
    if (!agree) {
        return "register/step1";
        // return "redirect:register/step1";
    }
    model.addAttribute("registerRequest", new RegisterRequest());
    return "register/step2";
}

```

속성	타입	설명
value	String	요청 파라미터 이름
required	boolean	필수 여부. true인 경우 값이 전달되지 않으면 예외 발생
default	String	값이 전달되지 않으면 사용할 값

컨트롤러 구현

- 커맨드 객체 사용해서 요청 데이터 읽기 → 스칼라 변수에 맵핑

```
<form action="step3">
<p>
    <label>이메일:<br>
    <input type="text" name="email" />
</label>
</p>
<p>
    <label>이름:<br>
    <input type="text" name="name" />
</label>
</p>
<p>
    <label>비밀번호:<br>
    <input type="password" name="password" />
</label>
</p>
<p>
    <label>비밀번호 확인:<br>
    <input type="password" name="confirmPassword" />
</label>
</p>
<input type="submit" value="가입 완료">
</form>
```

```
public class RegisterRequest {
    private String email;
    private String password;
    private String confirmPassword;
    private String name;
```

```
@PostMapping("/register/step3")
public String handleStep3(String email, String password,
                           String confirmPassword, String name) {
    RegisterRequest req = new RegisterRequest();
    req.setEmail(email);
    req.setPassword(password);
    req.setConfirmPassword(confirmPassword);
    req.setName(name);
```

- 커맨드 객체 사용해서 요청 데이터 읽기 → 커맨드 객체에 맵핑

```
<form action="step3">
<p>
    <label>이메일:<br>
    <input type="text" name="email" />
</label>
</p>
<p>
    <label>이름:<br>
    <input type="text" name="name" />
</label>
</p>
<p>
    <label>비밀번호:<br>
    <input type="password" name="password" />
</label>
</p>
<p>
    <label>비밀번호 확인:<br>
    <input type="password" name="confirmPassword" />
</label>
</p>
<input type="submit" value="가입 완료">
</form>
```

```
public class RegisterRequest {
    private String email;
    private String password;
    private String confirmPassword;
    private String name;
```

```
@PostMapping("/register/step3")
public String handleStep3(RegisterRequest regReq) {
    try {
        memberRegisterService.regist(regReq);
        return "register/step3";
    } catch (DuplicateMemberException ex) {
        return "register/step2";
        //return "redirect:/register/step2";
    }
}
```

▪ 중첩 커맨드 객체 맵핑

```
<form method="post">  
  
    <!-- question and response tags will go here -->  
  
    <p>  
        <label>응답자 위치:<br>  
        <input type="text" name="res.location">  
        </label>  
    </p>  
    <p>  
        <label>응답자 나이:<br>  
        <input type="text" name="res.age">  
        </label>  
    </p>  
    <input type="submit" value="전송">  
</form>
```

```
public class Respondent {  
  
    private int age;  
    private String location;  
  
public class AnsweredData {  
  
    private List<String> responses;  
    private Respondent res;
```

```
@PostMapping  
public String submit(@ModelAttribute("ansData") AnsweredData data) {  
    return "survey/submitted";  
}
```

■ 컬렉션 커맨드 객체 맵핑

```
<form method="post">
<c:forEach var="q" items="${questions}" varStatus="status">
<p>
    ${status.index + 1}. ${q.title}<br/>
    <c:if test="${q.choice}">
        <c:forEach var="option" items="${q.options}">
            <label><input type="radio"
                name="responses[${status.index}]" value="${option}">
            ${option}</label>
        </c:forEach>
    </c:if>
    <c:if test="${! q.choice }">
        <input type="text" name="responses[${status.index}]">
    </c:if>
</p>
</c:forEach>
<!-- nested property tags will go here --&gt;</pre>
```

```
public class Respondent {
    private int age;
    private String location;
```

```
public class AnsweredData {
    private List<String> responses;
    private Respondent res;
```

```
@PostMapping
public String submit(@ModelAttribute("ansData") AnsweredData data) {
    return "survey/submitted";
}
```

컨트롤러 구현

교재 284 ~ 285p 참고

▪ View에서 커맨드 객체 사용

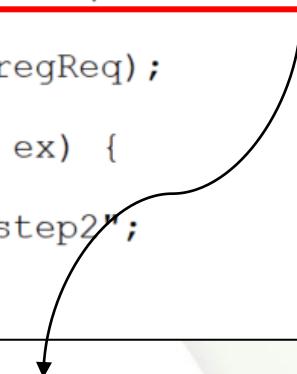
```
@PostMapping("/register/step2")
public String handleStep2(
    @RequestParam(value = "agree", defaultValue = "false") Boolean agree,
    Model model) {
    if (!agree) {
        return "register/step1";
        // return "redirect:register/step1";
    }
    model.addAttribute("registerRequest", new RegisterRequest());
    return "register/step2";
}
```

```
<p>
    <strong>${registerRequest.name}</strong>
    회원 가입을 완료했습니다.
</p>

<p><a href=<c:url value='/main'>>[첫 화면 이동]</a></p>
```

▪ 커맨드 객체의 이름 변경

```
@PostMapping("/register/step3")
public String handleStep3(@ModelAttribute("formData") RegisterRequest regReq) {
    try {
        memberRegisterService.regist(regReq);
        return "register/step3";
    } catch (DuplicateMemberException ex) {
        return "register/step2";
        //return "redirect:/register/step2";
    }
}
```



@ModelAttribute를 사용해서 이름 변경할 경우

→ `request.getParameter("formData")`로 커맨드 객체 접근

@ModelAttribute를 사용해서 이름을 변경하지 않는 경우

→ `request.getParameter("regReq")`로 커맨드 객체 접근

▪ 커맨드 객체와 스프링 폼 연동

- 스프링 폼 태그 라이브러리를 사용해서 커맨드 객체 바인딩

```
@PostMapping("/register/step2")
public String handleStep2(Boolean agree, Model model) {
    model.addAttribute("registerRequest", new RegisterRequest());
    return "register/step2";
}
```

```
<form:form action="step3" modelAttribute="registerRequest">
<p>
    <label>이메일:<br>
        <form:input path="email" />
    </label>
</p>
<p>
    <%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
    <label>이름:<br>
        <form:input path="name" />
    </label>
</p>
<p>
    <label>비밀번호:<br>
        <form:password path="password" />
    </label>
</p>
<p>
    <label>비밀번호 확인:<br>
        <form:password path="confirmPassword" />
    </label>
</p>
<input type="submit" value="가입 완료">
</form:form>
```

컨트롤러 구현

교재 289 ~ 291p 참고

- 컨트롤러 구현 없는 경로 매핑

- 컨트롤러에서 특별한 처리 작업이 없는 단순한 JSP 요청의 경우 컨트롤러를 만들지 않고 JSP를 연결할 수 있음

code based configuration (MvcConfig.java)

```
@Configuration  
@EnableWebMvc  
public class MvcConfig implements WebMvcConfigurer {  
  
    @Override  
    public void addViewControllers(ViewControllerRegistry registry) {  
        registry.addViewController("/main").setViewName("main");  
    }  
}
```

xml based configuration (servlet-context.xml)

```
<view-controller path="/main" view-name="main" />
```

- 요청 처리기 메서드 반환 타입

- 요청 처리기 메서드는 논리적 뷰와 모델을 DispatcherServlet에게 반환
- 반환 형식 종류

종류	설명
자동 추가되는 객체 모델	<ul style="list-style-type: none">Model, ModelMap, Map 타입 전달인자@ModelAttribute 어노테이션이 명시적/암시적 지정된 전달인자
@ModelAttribute	<ul style="list-style-type: none">메서드에 @ModelAttribute 어노테이션이 지정된 반환 타입 (이 때 뷰 이름은 메서드 이름에 일치)
ModelAndView	<ul style="list-style-type: none">뷰와 반환 데이터를 저장할 수 있는 전용 타입
String	<ul style="list-style-type: none">뷰 이름으로 사용될 문자열 (이 때 모델 데이터는 다른 방법으로 전달하도록 구현)
void	<ul style="list-style-type: none">뷰 이름은 메서드 이름에 일치모델 데이터는 다른 방법으로 전달하도록 구현
View	<ul style="list-style-type: none">사용자 정의 구현 내용을 포함하는 뷰 객체 반환
@ResponseBody	<ul style="list-style-type: none">HTTP 응답 메시지 본문을 문자열로 반환

JSTL 뷰 구현 - 컨트롤러와 뷰 사이의 데이터 전달

- Controller에서 ModelAttribute로 설정한 데이터는 JSP에서 사용할 수 있도록 HttpServletRequest 내장 객체에 저장되어 JSP에 전달됨
- 전달된 데이터는 스크립트 코드 블럭의 자바 코드, EL 등을 통해 사용됨

```
@RequestMapping(value="/edit.do", method=RequestMethod.POST)
public String add(
    @Valid @ModelAttribute("customer") CustomerModel model,
    BindingResult bindingResult) {
    if(bindingResult.hasErrors()) {
        return "edit";
    }
    try {
        customerService.saveCustomer(model.buildDomain());
    }
    catch(Exception e) {
        return "forward:/error.do";
    }
    return "result";
}
```

```
<h1>고객 등록 정보</h1>
이름 : ${customer.name} <br>
주소 : ${customer.address} <br>
이메일 : ${customer.email}
```

```
<h1>고객 등록 정보</h1>
<%>
Customer customer =
    (Customer)request.getAttribute("customer");
%>
이름 : <%= customer.getName() %> <br>
주소 : <%= customer.getAddress() %> <br>
이메일 : <%= customer.getEmail() %>
```

JSTL 뷰 구현 - 스프링 지원 품 태그 라이브러리

교재 308 ~ 319p 참고

- Controller에서 전달된 데이터를 JSP의 HTML 요소의 속성에 자동 바인딩 처리

종류	설명
form	HTML 폼 태그를 렌더링하고 다른 폼 태그에 바인딩 경로 제공
input	text 타입의 HTML input 요소를 렌더링
password	password 타입의 HTML input 요소를 렌더링
hidden	hidden 타입의 HTML input 요소를 렌더링
select	HTML select 요소를 렌더링
option	HTML select 요소 안에서 option 요소를 렌더링
options	HTML select 요소 안에서 option 요소의 컬렉션을 렌더링
radiobutton	radiobutton 타입의 HTML input 요소를 렌더링
radioButtons	radiobutton 타입의 HTML input 요소의 컬렉션을 렌더링
checkbox	checkbox 타입의 HTML input 요소를 렌더링
checkboxes	checkbox 타입의 HTML input 요소의 컬렉션을 렌더링
textarea	HTML textarea 요소 렌더링
errors	사용자에게 바인딩 및 검증 에러 표시
label	HTML label 요소를 렌더링하고 input 요소와 연관
button	HTML button 요소를 렌더링

JSTL 뷰 구현 - 스프링 지원 폼 태그 라이브러리

교재 308 ~ 319p 참고

■ 폼 태그 애트리뷰트

종류	설명
path	바인딩할 모델 객체의 속성 지정
readonly	폼 요소의 읽기 전용 여부 지정
disabled	폼 요소 활성화 여부 지정
cssClass	폼 요소에 적용할 css 클래스 지정
cssErrorClass	바인딩 또는 검증 에러 정보를 표시하는 데 사용할 css 클래스 지정
id	Javascript에서 식별자로 사용할 폼 요소의 id 지정

■ 컬렉션 탑입 태그 애트리뷰트

종류	설명
items	렌더링할 모델 객체 컬렉션 속성 지정
itemValue	렌더링할 단일 요소의 값 지정
itemLabel	렌더링할 단일 요소의 제목 지정
multiple	다중 선택 허용 여부 지정

폼 태그 예제

```
@RequestMapping(value="/edit.do", method=RequestMethod.POST)
public String add(
    @Valid @ModelAttribute("customer") CustomerModel model,
    BindingResult bindingResult) {
    if(bindingResult.hasErrors()) {
        return "edit";
    }
    try {
        customerService.saveCustomer(model.buildDomain());
    }
    catch(Exception e) {
        return "forward:/error.do";
    }
    return "result";
}
```

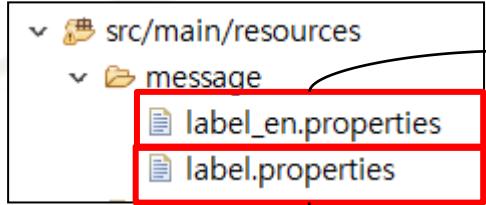
```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<h3>고객 정보 입력</h3>
<fieldset>
<form:form method="post" modelAttribute="customer">
    <form:label path="name" cssErrorClass="error">이름 : </form:label>
    <form:input type="text" path="name" />
    <form:errors path="name" cssClass="error"/> <br>
    <form:label path="address" cssErrorClass="error">주소 : </form:label>
    <form:input type="text" path="address" size="60" />
    <form:errors path="address" cssClass="error"/> <br>
    <form:label path="email" cssErrorClass="error">이메일 : </form:label>
    <form:input type="text" path="email" />
    <form:errors path="email" cssClass="error"/> <br>
    <input type="submit" value="저장" />
</form:form>
</fieldset>
```

스프링 국제화

- 스프링은 표준 JSTL 태그 라이브러리와 동일한 방식으로 리소스 번들과 메시지 포맷을 사용하여 국제화 지원
- 리소스 번들을 쉽게 사용할 수 있도록 추상화 메시지 소스 제공

종류	설명
ResourceBundleMessageSource	리소스 파일을 반드시 /WEB-INF/classes 폴더 또는 resources 폴더에 저장하도록 제약
Reloadable ResourceBundleMessageSource	리소스 파일의 위치를 자유롭게 설정

▪ .properties 리소스 파일 작성



member.register=회원가입

term=약관

term.agree=약관동의

next.btn=다음단계

member.info=회원정보

email=이메일

name=이름

password=비밀번호

password.confirm=비밀번호 확인

register.btn=가입 완료

register.done={0} 님 ({1}), 회원 가입을 완료했습니다.

go.main=메인으로 이동

required=필수항목입니다.

bad.email=이메일이 올바르지 않습니다.

duplicate.email=중복된 이메일입니다.

nomatch.confirmPassword=비밀번호와 확인이 일치하지 않습니다.

```
member.register=register  
term=term  
term.agree=term-agree  
next.btn=next  
  
member.info=member-info  
email=email  
name=name  
password=password  
password.confirm=password-confirm  
register.btn=registration completed  
  
register.done=<strong>{0} ({1})</strong>, registration successfully completed  
  
go.main=move to main  
  
required=required  
bad.email=invalid email  
duplicate.email=duplicated email  
nomatch.confirmPassword=password not matched
```

스프링 국제화 구현 교재 321 ~ 322p 참고

- 메시지 소스 빈 등록 (빈 설정 파일)

- code based configuration

```
@Bean  
public MessageSource messageSource(){  
    ResourceBundleMessageSource ms =  
        new ResourceBundleMessageSource();  
    ms.setBasename("message.label");  
    ms.setDefaultEncoding("UTF-8");  
    return ms;  
}
```

bean name은 반드시 messageSource

- xml based configuration

```
<beans:bean id="messageSource"  
            class="org.springframework.context.support.ResourceBundleMessageSource">  
    <beans:property name="basename" value="message.label" />  
    <beans:property name="defaultEncoding" value="UTF-8" />  
</beans:bean>
```

▪ 스프링 태그 라이브러리를 이용한 메시지 표시

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>

<h2><spring:message code="term" /></h2>
<p>약관 내용</p>
<form action="step2" method="post">
<label>
    <input type="checkbox" name="agree" value="true">
    <spring:message code="term.agree" />
</label>
<input type="submit" value="
```

term=약관
term.agree=약관동의
next.btn=다음단계

member.info=회원정보
email=이메일
name=이름
password=비밀번호
password.confirm=비밀번호 확인
register.btn=가입 완료

register.done={0}님 ({1}), 회원 가입을 완료했습니다.

▪ 스프링 태그 라이브러리를 이용한 메시지 표시

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
```

```
<h2><spring:message code="member.info" /></h2>
<form:form action="step3" modelAttribute="registerRequest">
<p>
    <label><spring:message code="email" /><br>
    <form:input path="email" />
    </label>
</p>
<p>
    <label><spring:message code="name" /><br>
    <form:input path="name" />
    </label>
</p>
<p>
    <label><spring:message code="password" /><br>
    <form:password path="password" />
    </label>
</p>
<p>
    <label><spring:message code="password.confirm" /><br>
    <form:password path="confirmPassword" />
    </label>
</p>
<input type="submit" value=<spring:message code="register.btn" />>
</form:form>
```

member.info=회원정보

email=이메일

name=이름

password=비밀번호

password.confirm=비밀번호 확인

register.btn=가입 완료

▪ 스프링 태그 라이브러리를 이용한 메시지 표시

```
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>
```

```
<head>
    <title><spring:message code="member.register" /></title>
</head>
<body>
    <p>
        <spring:message code="register.done">
            <spring:argument value="${registerRequest.name}" />
            <spring:argument value="${registerRequest.email}" />
        </spring:message>
    </p>
    <p>
        <a href=<c:url value='/main' />>
            [<spring:message code="go.main" />]
        </a>
    </p>
</body>
```

register.btn=가입 완료

register.done={0} 님 ({1}), 회원 가입을 완료했습니다.

go.main=메인으로 이동

데이터 검증 교재 330 ~ 336p 참고

■ Validator, Errors 인터페이스 사용해서 검증 객체 구현

```
public class RegisterRequestValidator implements Validator {
    private static final String emailRegEx =
        "^[_A-Za-z0-9-\\+]+(\\.[_A-Za-z0-9-]*)@" +
        "[A-Za-z0-9-]+(\\.[A-Za-z0-9]+)*(\\.[_A-Za-z]{2,})$";
    private Pattern pattern;

    public RegisterRequestValidator() {}

    public boolean supports(Class<?> clazz) {}

    @Override
    public void validate(Object target, Errors errors) {
        System.out.println("RegisterRequestValidator#validate(): " + this);
        RegisterRequest regReq = (RegisterRequest) target;
        if (regReq.getEmail() == null || regReq.getEmail().trim().isEmpty()) {
            errors.rejectValue("email", "required");
        } else {
            Matcher matcher = pattern.matcher(regReq.getEmail());
            if (!matcher.matches()) {
                errors.rejectValue("email", "bad");
            }
        }
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "name", "required");
        ValidationUtils.rejectIfEmpty(errors, "password", "required");
        ValidationUtils.rejectIfEmpty(errors, "confirmPassword", "required");
        if (!regReq.getPassword().isEmpty()) {
            if (!regReq.isPasswordEqualToConfirmPassword()) {
                errors.rejectValue("confirmPassword", "nomatch");
            }
        }
    }
}
```

요청 데이터 검증 교재 330 ~ 336p 참고

■ 검증 객체 직접 사용

```
@PostMapping("/register/step3")
public String handleStep3(RegisterRequest regReq, Errors errors) {
    new RegisterRequestValidator().validate(regReq, errors);
    if (errors.hasErrors())
        return "register/step2";

    try {
        memberRegisterService.regist(regReq);
        return "register/step3";
    } catch (DuplicateMemberException ex) {
        errors.rejectValue("email", "duplicate");
        return "register/step2";
    }
}
```

요청 데이터 검증 교재 343 ~ 345p 참고

컨트롤러 수준에서 검증 자동화

```
@PostMapping("/register/step3")
public String handleStep3(@Valid RegisterRequest regReq, Errors errors) {
    if (errors.hasErrors())
        return "register/step2";

    try {
        memberRegisterService.regist(regReq);
        return "register/step3";
    } catch (DuplicateMemberException ex) {
        errors.rejectValue("email", "duplicate");
        return "register/step2";
    }
}

@InitBinder
protected void initBinder(WebDataBinder binder) {
    binder.addValidators(new RegisterRequestValidator());
}
```

요청 데이터 검증 교재 341~ 343p 참고

■ 전역 수준에서 검증 자동화

```
@Configuration  
@EnableWebMvc  
public class MvcConfig implements WebMvcConfigurer {  
  
    @Override  
    public Validator getValidator() {  
        return new RegisterRequestValidator();  
    }  
  
    public void configureDefaultServletHandling( )  
    public void configureViewResolvers(ViewResolverRegistry registry) {  
    public void addViewControllers(ViewControllerRegistry registry) {  
    public MessageSource messageSource() {  
  
    }  
}
```

```
@PostMapping("/register/step3")  
public String handleStep3(@Valid RegisterRequest regReq, Errors errors) {  
    if (errors.hasErrors())  
        return "register/step2";  
  
    try {  
        memberRegisterService.regist(regReq);  
        return "register/step3";  
    } catch (DuplicateMemberException ex) {  
        errors.rejectValue("email", "duplicate");  
        return "register/step2";  
    }  
}
```

요청 데이터 검증 교재 345 ~ 349p 참고

▪ Bean Validation을 활용한 검증

```
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>1.1.0.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.4.2.Final</version>
</dependency>
```

```
public class RegisterRequest {
    @NotBlank
    @Email
    private String email;
    @Size(min = 6)
    private String password;
    @NotEmpty
    private String confirmPassword;
    @NotEmpty
    private String name;
```

```
@PostMapping("/register/step3")
public String handleStep3(@Valid RegisterRequest regReq, Errors errors) {
    if (errors.hasErrors())
        return "register/step2";

    try {
        memberRegisterService.regist(regReq);
        return "register/step3";
    } catch (DuplicateMemberException ex) {
        errors.rejectValue("email", "duplicate");
        return "register/step2";
    }
}
```

빈 검증 어노테이션 교재 349 ~ 351p 참고

■ 주요 JSR-303 표준 어노테이션

종류	설명
@Null	필드 값이 null이 아니면 검증 실패
@NotNull	필드 값이 null이면 검증 실패
@NotBlank	문자열 필드의 값이 빈문자열이면 검증 실패
@AssertTrue @AssertFalse	boolean 타입의 필드의 true, false 여부 검증
@DecimalMax @DecimalMin	숫자 타입의 필드의 최소/최대 값 지정 double, float은 정밀도 문제로 지원하지 않음
@Digits(integer=, fraction=)	정수 및 소수점 자릿수 제한
@Future @Past	날짜 타입의 필드가 미래 또는 과거인지 검증
@Max(value=) @Min(value=)	최대 또는 최소 값 지정
@Pattern(regexp=, flag=) @Patterns({@Pattern()})	필드 값이 정규 표현식과 일치하지 않으면 검증 실패
@Size(min=, max=)	배열, 컬렉션, 맵 타입 필드에 포함된 요소 갯수의 최대/최소 값 지정

빈 검증 어노테이션 교재 349 ~ 351p 참고

- hibernate-validator 구현체에서 제공하는 주요 어노테이션

종류	설명
@Length(min=, max=)	문자열 타입 필드의 최소, 최대 길이 지정
@NotEmpty	문자열 타입의 필드가 null 또는 빈문자열이면 검증 실패
@Range(min=, max=)	숫자 또는 숫자 값을 표시하는 문자열의 값 범위 지정
@Valid	관련된 객체를 재귀적으로 검증 컬렉션은 요소들을 재귀적 검증 / 맵은 값 요소들을 재귀적으로 검증
@Email	문자열 타입의 필드가 이메일 주소 명세를 충족하는지 검증
@CreditCardNumber	문자열 타입의 필드가 신용카드 번호인지 검증

요청 데이터 검증

교재 337 ~ 340p 참고

■ 에러 메시지 출력

```
<body>
    <h2><spring:message code="member.info" /></h2>
    <form:form action="step3" modelAttribute="registerRequest">
        <p>
            <label><spring:message code="email" /><br>
            <form:input path="email" />
            <form:errors path="email"/>
            </label>
        </p>
        <p>
            <label><spring:message code="name" /><br>
            <form:input path="name" />
            <form:errors path="name"/>
            </label>
        </p>
        <p>
            <label><spring:message code="password" /><br>
            <form:password path="password" />
            <form:errors path="password"/>
            </label>
        </p>
        <p>
            <label><spring:message code="password.confirm" /><br>
            <form:password path="confirmPassword" />
            <form:errors path="confirmPassword"/>
            </label>
        </p>
        <input type="submit" value="<spring:message code="register.btn" />">
    </form:form>
</body>
```

label.properties file

required=필수항목입니다.
bad.email=이메일이 올바르지 않습니다.
duplicate.email=중복된 이메일입니다.
nomatch.confirmPassword=비밀번호와 확인이 일치하지 않습니다.

기능 구현

로그인, 패스워드 변경 기능 구현

교재 352 ~ 370p 참고

인터셉터

- 다수의 컨트롤러에 동일한 기능을 적용할 수 있는 도구
 - 컨트롤러의 요청 처리 전/후에 공통 기능 적용 가능
- 필터는 모든 요청을 대상으로 하지만 인터셉터는 컨트롤러의 요청 처리만을 대상으로 적용

인터셉터

- 구현 교재 371 ~ 376p 참고

```
public class AuthCheckInterceptor implements HandlerInterceptor {  
  
    @Override  
    public boolean preHandle(  
        HttpServletRequest request,  
        HttpServletResponse response,  
        Object handler) throws Exception {  
        HttpSession session = request.getSession(false);  
        if (session != null) {  
            Object authInfo = session.getAttribute("authInfo");  
            if (authInfo != null) {  
                return true;  
            }  
        }  
        response.sendRedirect(request.getContextPath() + "/login");  
        return false;  
    }  
}
```

code based configuration

```
@Override  
public void addInterceptors(InterceptorRegistry registry) {  
    registry.addInterceptor(authCheckInterceptor())  
        .addPathPatterns("/edit/**")  
        .excludePathPatterns("/edit/help/**");  
}  
  
@Bean  
public AuthCheckInterceptor authCheckInterceptor() {  
    return new AuthCheckInterceptor();  
}
```

xml based configuration

```
<interceptors>  
    <interceptor>  
        <mapping path="/edit/**" />  
        <exclude-mapping path="/edit/help/**" />  
        <beans:bean class="interceptor.AuthCheckInterceptor" />  
    </interceptor>  
</interceptors>
```

기능 구현

로그인 아이디 기억 기능 구현
→ 쿠키 사용 연습

교재 376 ~ 381p 참고

날짜 형식 변환 교재 384 ~ 390p 참고

```
public class ListCommand {  
  
    @DateTimeFormat(pattern = "yyyyMMddHH")  
    private LocalDateTime from;  
    @DateTimeFormat(pattern = "yyyyMMddHH")  
    private LocalDateTime to;
```

```
@RequestMapping("/members")  
public String list(  
    @ModelAttribute("cmd") ListCommand listCommand,  
    Errors errors, Model model) {  
    if (errors.hasErrors()) {  
        return "member/memberList";  
    }  
    if (listCommand.getFrom() != null && listCommand.getTo() != null) {  
        List<Member> members = memberDao.selectByRegdate(  
            listCommand.getFrom(), listCommand.getTo());  
        model.addAttribute("members", members);  
    }  
    return "member/memberList";  
}
```

컨트롤러 예외 처리 교재 398 ~ 404p 참고

▪ 개별 컨트롤러 수준 예외 처리 구현

```
@Controller
public class MemberDetailController {

    private MemberDao memberDao;

    public void setMemberDao(MemberDao memberDao) {}

    public String detail(@PathVariable("id") Long memId, Model model) {

        @ExceptionHandler(TypeMismatchException.class)
        public String handleTypeMismatchException() {
            return "member/invalidId";
        }

        @ExceptionHandler(MemberNotFoundException.class)
        public String handleNotFoundException() {
            return "member/noMember";
        }
    }
}
```

▪ 전역 컨트롤러 수준 예외 처리 구현

```
@ControllerAdvice("spring")
public class CommonExceptionHandler {

    @ExceptionHandler(RuntimeException.class)
    public String handleRuntimeException() {
        return "error/commonException";
    }
}
```

스프링 MVC FileUpload

FileUpload 구현

- 스프링은 서블릿 3.0 사양에서 제공하는 방식과 Apache Commons FileUpload 컴포넌트를 사용하는 방식 모두 지원
- 서블릿 3.0 방식 설정
 - 파일 업로드 설정 (web.xml)

```
<servlet>
    <servlet-name>product</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/appServlet/product-context.xml</param-value>
    </init-param>
    <load-on-startup>3</load-on-startup>
    <multipart-config>
        <max-file-size>20848820</max-file-size>
        <max-request-size>418018841</max-request-size>
        <file-size-threshold>1048576</file-size-threshold>
    </multipart-config>
</servlet>
```

- 파일 업로드 처리 빈 등록(빈 설정 파일)

```
<beans:bean id="multipartResolver"
    class="org.springframework.web.multipart.support.StandardServletMultipartResolver"/>
```

FileUpload 구현

- Apache Commons FileUpload 컴포넌트 사용 설정
 - 의존성 패키지 등록 (pom.xml)

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.4</version>
</dependency>
```

- 파일 업로드 처리 빈 등록 (빈 설정 파일)

```
<beans:bean id="multipartResolver"
    class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
    <beans:property name="maxUploadSize" value="20848820"/>
</beans:bean>
```

FileUpload 구현

■ 파일 업로드 처리 (공통)

```
@RequestMapping(value="/edit.do", method=RequestMethod.POST)
public String add(@Valid @ModelAttribute("product") ProductModel model,
    BindingResult bindingResult,
    @RequestParam(value="image", required=false) MultipartFile image,
    HttpServletRequest request) {
    if(bindingResult.hasErrors())
        return "edit";
    try {
        if(!image.isEmpty()) {
            if(!image.getContentType().equals("image/jpeg")) {
                throw new ImageUploadException("JPEG 이미지만 선택해주세요.");
            }
            String webRootPath = request.getSession().getServletContext().getRealPath("/");
            String filePath = webRootPath + "resources/" + image.getOriginalFilename();
            File file = new File(filePath);
            FileUtils.writeByteArrayToFile(file, image.getBytes());
            logger.info("업로드 파일 : " + filePath);
        }
    }
    catch(Exception e) {
        bindingResult.reject(e.getMessage());
        return "edit";
    }
    productService.saveProduct(model.buildDomain());
    return "result";
}
```

사용자 정의 View

- InternalResourceViewResolver는 JstlView를 사용해서 응답 컨텐츠 생성
 - JstlView는 일종의 JSP Parser 역할 → ServletContainer이 JSP 처리 기능과 동일
- JSP 기반의 HTML 응답 컨텐츠가 아닌 다른 종류의 응답 컨텐츠를 만들어야 할 경우 적절한 다른 View를 사용하거나 직접 View 클래스 구현 필요
- 사용자 정의 View 클래스 구현 방법
 - View 인터페이스 구현 (render 메서드)
 - AbstractView 추상 클래스 상속 (renderMergedOutputModel)

파일 다운로드

- 파일 다운로드는 HTML이 아닌 컨텐츠 응답
 - 사용자 정의 View를 통해 구현 가능

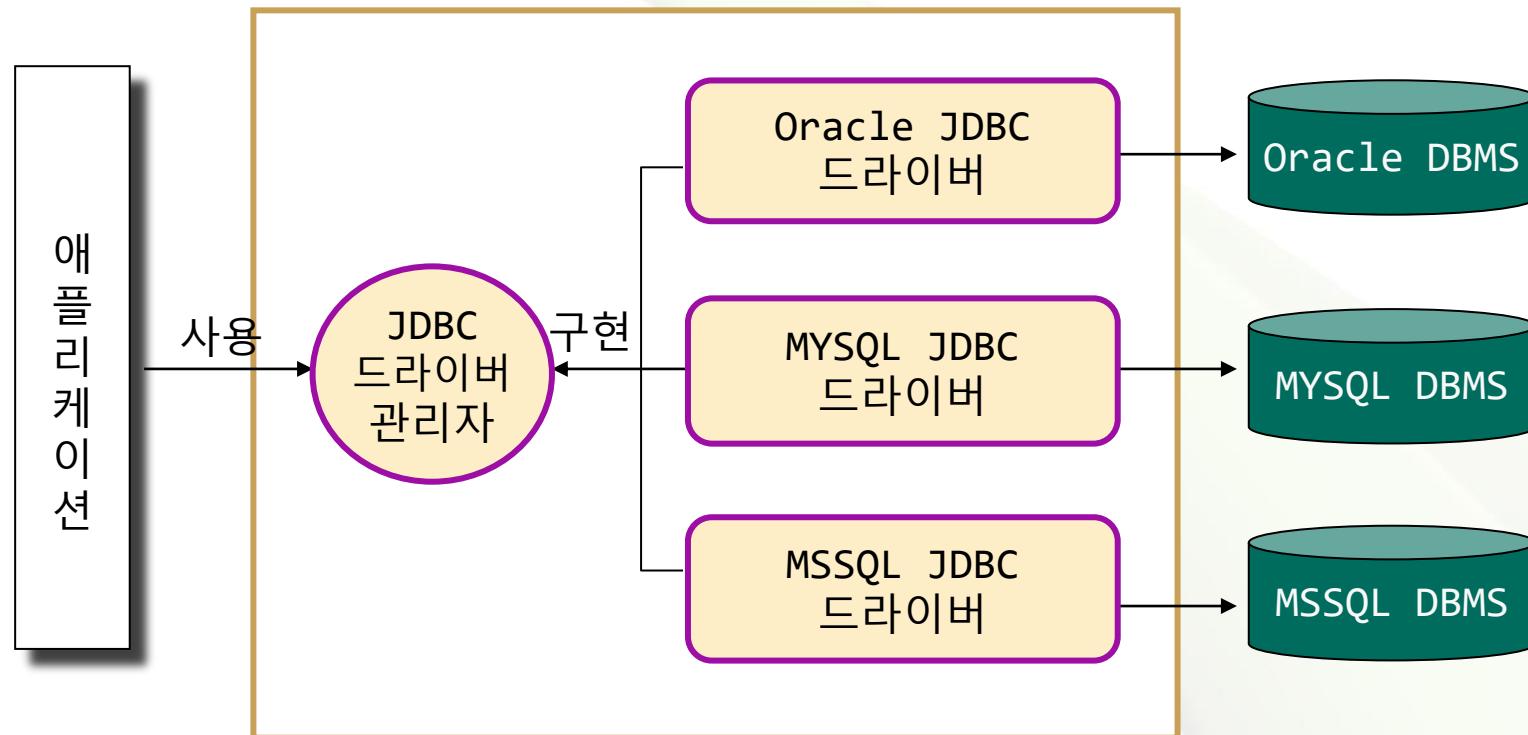
```
public class DownloadView extends AbstractView {  
  
    @Override  
    protected void renderMergedOutputModel(  
        Map<String, Object> params,  
        HttpServletRequest request, HttpServletResponse response) throws Exception {  
  
        //다운로드 처리  
        //1. 브라우저에게 처리할 수 없는 컨텐츠로 알려주어서 다운로드 처리하도록 지정  
        response.setContentType("application/octet-stream");  
  
        //2. 다운로드 처리할 때 필요한 정보 제공  
        response.addHeader( "Content-Disposition", "Attachment;Filename=\"web.xml\" " );  
  
        //3. 파일을 응답스트림에 기록  
        String file2 = request.getSession().getServletContext().getRealPath("/WEB-INF/web.xml");  
        BufferedInputStream istream = new BufferedInputStream(new FileInputStream(file2));  
        BufferedOutputStream ostream = new BufferedOutputStream(response.getOutputStream());  
        while (true) {  
            int data = istream.read();  
            if (data != -1) ostream.write(data);  
            else break;  
        }  
        istream.close();  
        ostream.close();  
    }  
}
```

JDBC와 Spring 데이터베이스 연동 지원

JDBC 개념과 역할

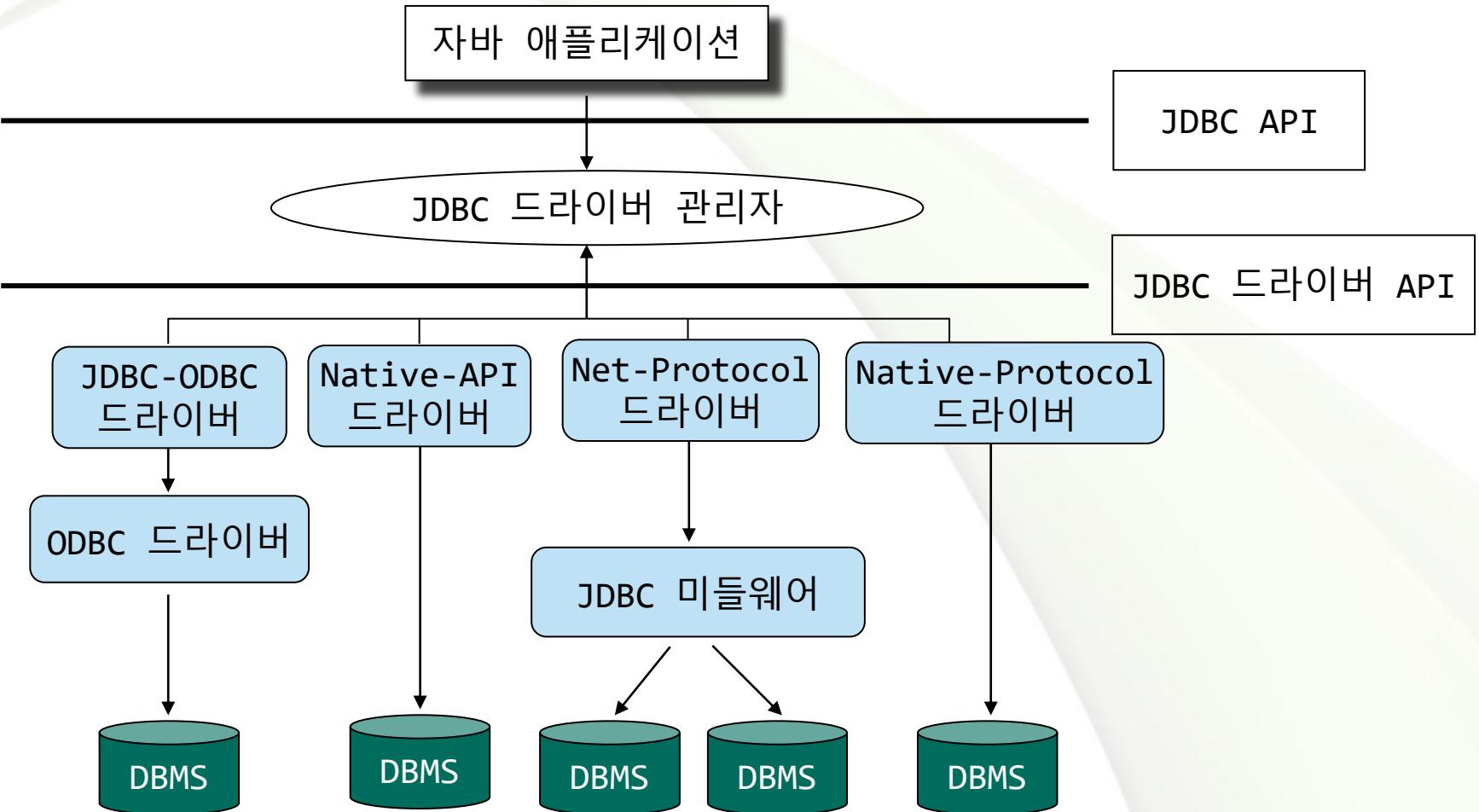
- Java Database Connectivity
- 자바애플리케이션에서 표준화된 데이터베이스 접근 제공.
- 각 데이터베이스 접속에 대한 상세한 정보를 추상화.
- 이론적으로는 개발된 애플리케이션에서 DB 변경시 JDBC 드라이버 교체만으로 가능

JDBC 구성



JDBC 드라이버 유형

- JDBC 드라이버 구성도



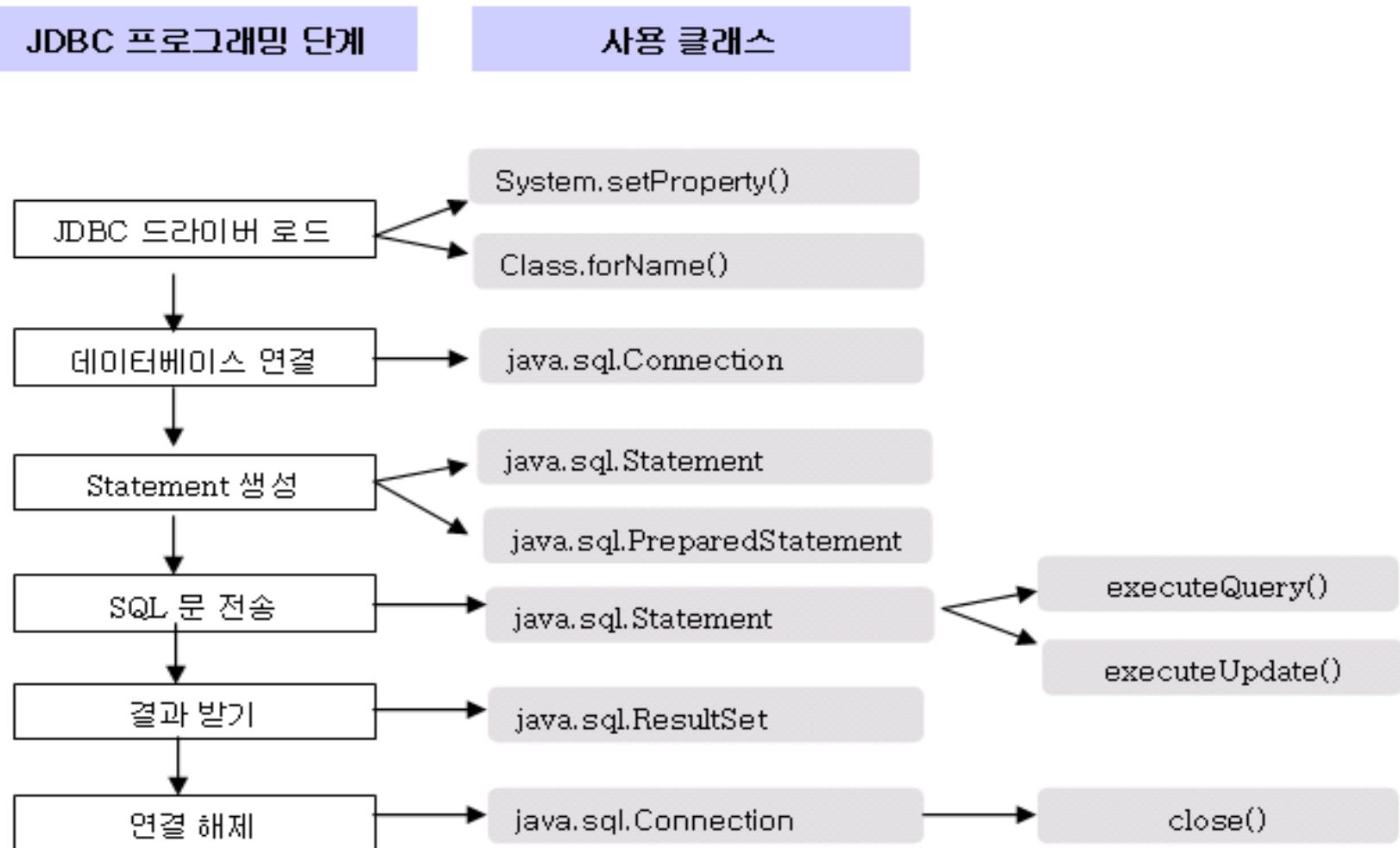
JDBC 드라이버 설치

- JDBC 드라이버 선택
 - JDBC 드라이버는 사용하고자 하는 데이터베이스 벤더 별로 제공 됨
- 설치 디렉터리(다음 중 한 가지를 이용)
 - JDK설치디렉터리\jre\lib\ext\에 복사하는 방법.
 - 톰캣설치디렉터리\common\lib 폴더에 복사하는 방법
 - 이클립스 프로젝트의 WebContent\WEB-INF\lib 폴더에 복사하는 방법
 - Maven을 사용하는 경우 pom.xml 파일에 의존성 패키지 등록

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.33</version>
</dependency>
```

JDBC 프로그래밍 과정

■ JDBC 프로그래밍 단계



JDBC 프로그래밍 단계

- 데이터베이스 드라이버 로드

```
DriverManager.registerDriver(new com.mysql.jdbc.Driver());
```

```
Class.forName("com.mysql.jdbc.Driver");
```

- 데이터베이스 연결

```
Connection conn = DriverManger.getConnection("jdbc-url", "id", "pwd");
```

- JDBC-URL 구성

```
MYSQL → jdbc:mysql://ip:port/db-name
```

```
ORACLE → jdbc:oracle:thin:@ip:port/sid
```

JDBC 프로그래밍 단계

- PreparedStatement 생성 및 쿼리 실행
 - SQL과 SQL 실행에 사용할 데이터를 분리해서 관리하는 방식으로 성능과 관리 면에서 모두 권장 되는 방식

```
PreparedStatement pstmt =  
    conn.prepareStatement("insert into test values(?,?) ");  
pstmt.setString(1,request.getParameter("username"));  
pstmt.setString(2, request.getParameter("email"));  
pstmt.executeUpdate();
```

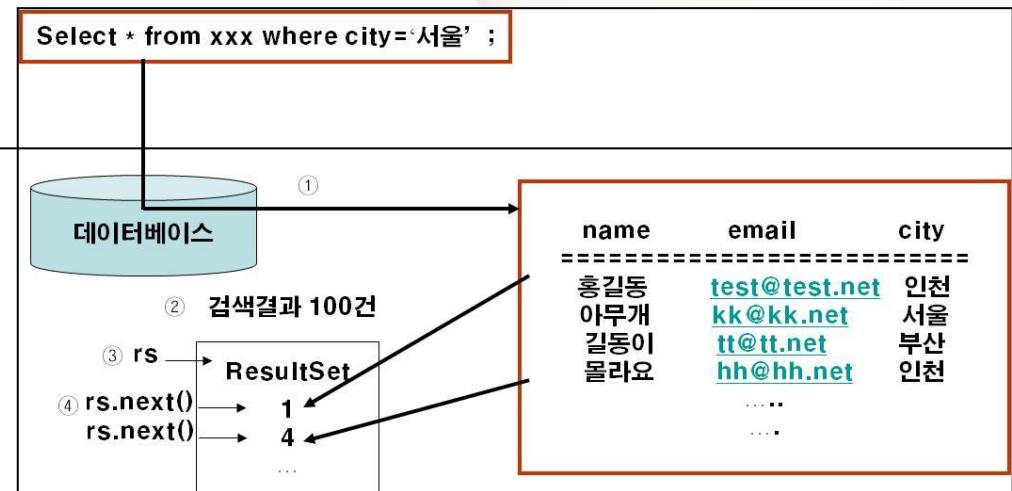
JDBC 프로그래밍 단계

- 결과 받기

```
ResultSet rs = pstmt.executeQuery();
```

- ResultSet은 커서 개념의 연결 포인터로 next()메서드를 통해 다음 행으로 이동

```
ResultSet rs = pstmt.executeQuery();
while(rs.next()) {
    name = rs.getString(1); // or rs.getString("name");
    age = rs.getInt(2); // or rs.getInt("email");
}
rs.close();
```



JDBC 프로그래밍 단계

- 연결해제

- Connection 을 close() 하지 않으면 사용하지 않는 연결이 유지되어 DB 자원 낭비.

```
conn.close();
```

저수준 JDBC 코드의 문제 ▪ 교재 179p 참고

- 구조 코드로 인한 코드량 증가
 - 예외처리 필수
 - 드라이버 등록, 연결 생성, 명령 생성, 명령 실행, 연결 닫기 등의 표준 API 호출
 - 실제 변경되는 내용은 SQL과 전달인자 및 결과 처리 코드
- 데이터 구조 불일치로 인한 효율성 저하 → 객체와 테이블 사이의 호환성 문제
 - 데이터 타입 불일치
 - 관계 불일치
 - 입자성 불일치
 - 상속성 불일치
 - 식별 불일치

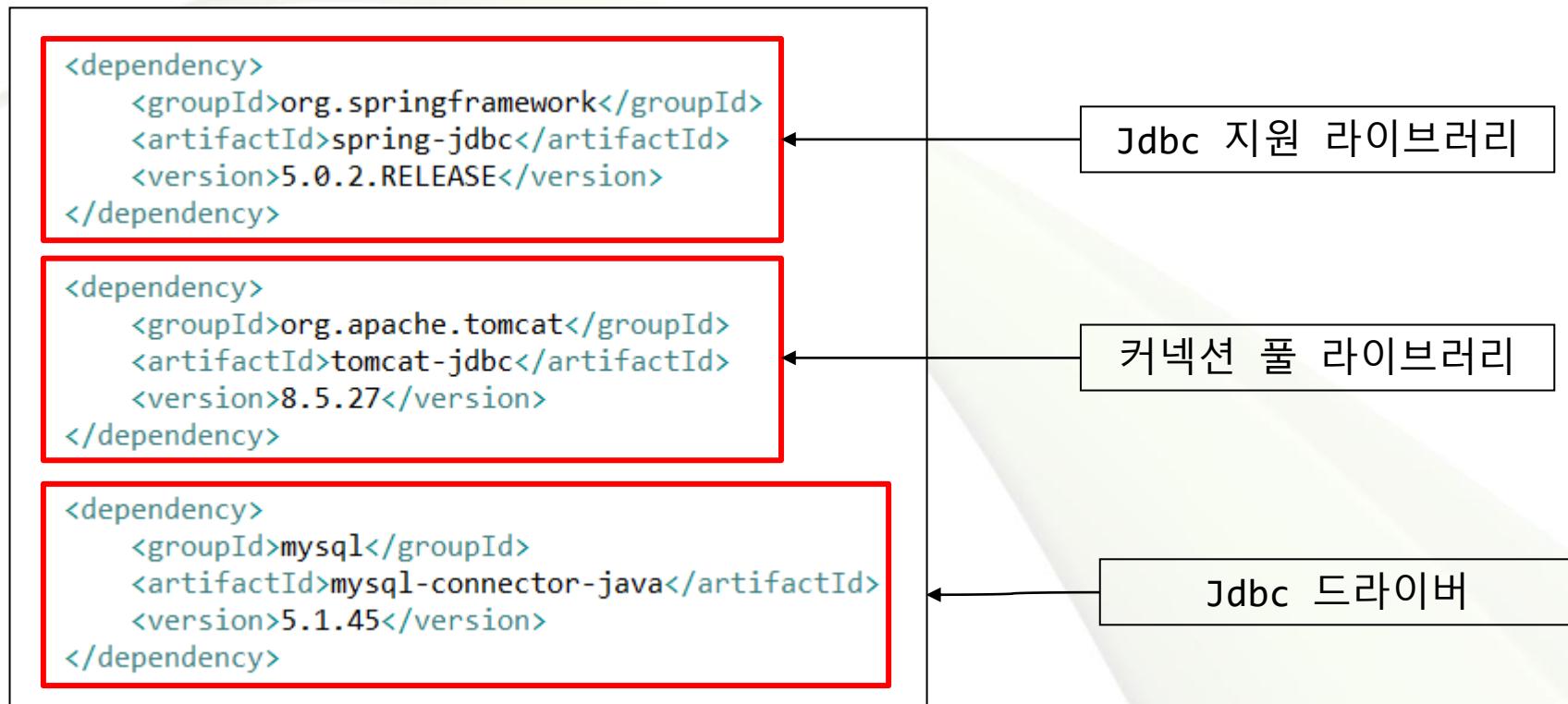
Spring 데이터베이스 연동 지원 ▪ 교재 179 ~ 181p 참고

- 템플릿 클래스를 통한 데이터 접근 지원
 - 템플릿 메서드 패턴과 전략 패턴 결합
 - 동일한 코드의 중복을 제거하고 필요한 최소한의 내용으로 데이터베이스 연동 코드 작성 가능
- 의미 있는 예외 클래스 제공 ▪ 교재 209 ~ 211p 참고
 - 데이터베이스 연동 과정 중에 발생하는 SQLException을 대체하고 오류의 원인을 예측할 수 있는 다양한 예외 클래스 제공
 - MyBatis, JPA, Hibernate 등 사용하는 데이터베이스 연동 기술에 의존하지 않고 동일한 방식으로 예외처리 가능
- 트랜잭션 처리 지원
 - 데이터베이스 연동 기술에 상관 없이 동일한 방식으로 트랜잭션 처리가 가능한 프로그래밍 기법 제공
 - 코드 기반 트랜잭션 및 선언적 트랜잭션 지원

Spring JDBC 설치

▪ 교재 181 ~ 183p 참고

▪ 의존 패키지 등록 (pom.xml)



Spring DataSource 설정 (연결 설정)

- **DataSource**
 - JDBC API에 정의된 표준 커넥션 풀 인터페이스
- Spring은 템플릿 클래스 및 ORM 프레임워크 연동 클래스를 사용할 경우 DataSource를 통해 Connection 제공
- 제공 방식
 - 커넥션 풀을 이용한 DataSource 설정
 - JNDI를 이용한 DataSource 설정
 - DriverManager를 이용한 DataSource 설정

커넥션 풀을 이용한 DataSource 설정

- Apache Commons DBCP와 같은 라이브러리를 이용해서 DataSource 설정
- 스프링 빈 설정 (tomcat connection pool)

```
@Bean(destroyMethod = "close")
public DataSource dataSource() {
    DataSource ds = new DataSource();
    ds.setDriverClassName("com.mysql.jdbc.Driver");
    ds.setUrl("jdbc:mysql://localhost/spring5fs?characterEncoding=utf8");
    ds.setUsername("spring5");
    ds.setPassword("spring5");
    ds.setInitialSize(2);
    ds.setMaxActive(10);
    ds.setTestWhileIdle(true);
    ds.setMinEvictableIdleTimeMillis(60000 * 3);
    ds.setTimeBetweenEvictionRunsMillis(10 * 1000);
    return ds;
}
```

교재 185p ~ 190p 참고

```
<bean id="dataSource"
      class="org.apache.tomcat.jdbc.pool.DataSource"
      destroy-method="close">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/spring5fs?characterEncoding=utf-8" />
    <property name="username" value="spring5" />
    <property name="password" value="spring5" />
    <property name="initialSize" value="2" />
    <property name="maxActive" value="10" />
    <property name="testWhileIdle" value="true" />
    <property name="minEvictableIdleTimeMillis" value="18000" />
    <property name="timeBetweenEvictionRunsMillis" value="10000" />
</bean>
```

DataSource를 사용해서 커넥션 구하기

- 사용할 클래스에 필드 선언 후 의존성 주입 처리

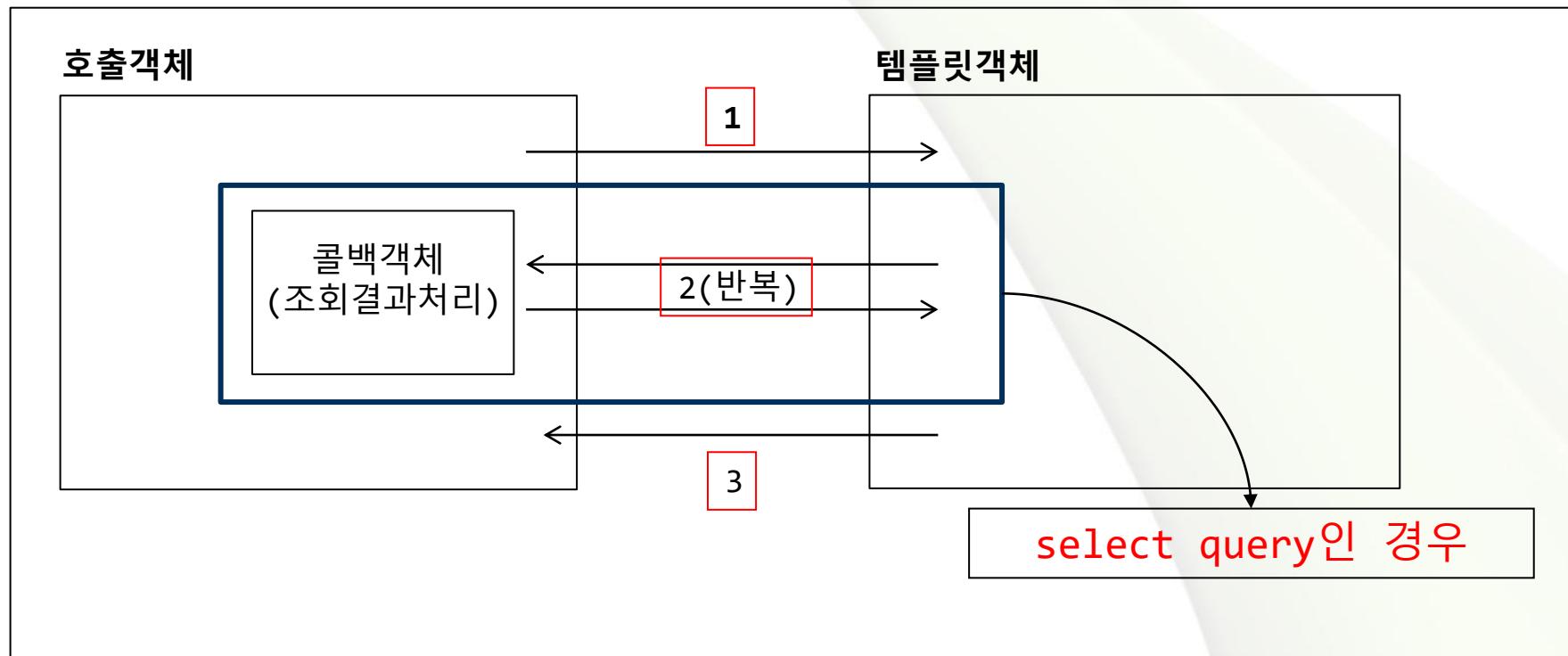
```
public class MyClass implements MyInterface {  
  
    @Autowired  
    private DataSource dataSource;  
  
    public void testMethod() {  
        Connection conn = null;  
        try {  
  
            conn = dataSource.getConnection(); //트랜잭션 활용 불가능  
            conn = DataSourceUtils.getConnection(dataSource); //트랜잭션 활용 가능  
            ...  
  
        } finally {  
  
            JdbcUtils.closeConection(conn); // 트랜잭션 사용하지 않는 경우  
            DataSourceUtils.releaseConnection(conn, dataSource); //트랜잭션 사용  
        }  
    }  
}
```

스프링 데이터베이스 연동 템플릿

- 연결 객체 획득, 예외처리 등 중복 코드를 제거하고 효과적인 데이터베이스 연동 코드 작성을 위해 템플릿 지원
- 종류
 - `JdbcTemplate`
 - SQL 실행을 위한 다양한 메서드 제공
 - 인덱스 기반 전달인자 사용
 - `NamedParameterJdbcTemplate`
 - 인덱스 기반 전달인자가 아닌 이름 기반의 전달인자 사용
 - 이를 위해 `Map`이나 `SqlParameterSource` 등을 사용

템플릿 구조의 작동 원리

- 반복되는 구조를 분리해서 별도의 클래스로 정의하고 변경되는 내용을 전달해서 기능을 처리하는 기법
- SQL, Parameter 매핑 데이터, 조회 결과를 처리할 객체 참조를 전달인자로 제공하면 템플릿의 구조 코드에서 이 전달인자를 사용해서 전체 데이터 연동 코드 수행



스프링 JDBC 템플릿 사용

■ 데이터 조회 (select)

```
public Member selectByEmail(String email) {
    List<Member> results = jdbcTemplate.query(
        "select * from member where EMAIL = ?",
        new RowMapper<Member>() {
            @Override
            public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
                Member member = new Member(
                    rs.getString("EMAIL"),
                    rs.getString("PASSWORD"),
                    rs.getString("NAME"),
                    rs.getTimestamp("REGDATE").toLocalDateTime());
                member.setId(rs.getLong("ID"));
                return member;
            }
        }, email);
    return results.isEmpty() ? null : results.get(0);
}

public Member selectByEmail2(String email) {
    Member result = jdbcTemplate.queryForObject(
        "select * from member where EMAIL = ?",
        new RowMapper<Member>() {
            @Override
            public Member mapRow(ResultSet rs, int rowNum) throws SQLException {
                Member member = new Member(
                    rs.getString("EMAIL"),
                    rs.getString("PASSWORD"),
                    rs.getString("NAME"),
                    rs.getTimestamp("REGDATE").toLocalDateTime());
                member.setId(rs.getLong("ID"));
                return member;
            }
        }, email);
    return result;
}
```

다중 행 조회

단일 행 조회

스프링 JDBC 템플릿 사용

- 데이터 변경 (insert, update, delete)

```
public void insert2(Member member) {  
  
    jdbcTemplate.update(  
        "insert into member (EMAIL, PASSWORD, NAME, REGDATE) " +  
        "values (?, ?, ?, ?)",  
        member.getEmail(),  
        member.getPassword(),  
        member.getName(),  
        Timestamp.valueOf(member.getRegisterDateTime()));  
  
}  
  
public void update(Member member) {  
  
    jdbcTemplate.update(  
        "update member set NAME = ?, PASSWORD = ? where EMAIL = ?",  
        member.getName(), member.getPassword(), member.getEmail());  
  
}
```

스프링 JDBC 템플릿 사용

■ 자동증가컬럼을 포함하는 insert

```
public void insert(Member member) {  
    KeyHolder keyHolder = new GeneratedKeyHolder();  
  
    jdbcTemplate.update(new PreparedStatementCreator() {  
        @Override  
        public PreparedStatement createPreparedStatement(Connection con)  
            throws SQLException {  
            // 파라미터로 전달받은 Connection을 이용해서 PreparedStatement 생성  
            PreparedStatement pstmt = con.prepareStatement(  
                "insert into member (EMAIL, PASSWORD, NAME, REGDATE) " +  
                "values (?, ?, ?, ?)",  
                new String[] { "ID" });  
            // 인덱스 파라미터 값 설정  
            pstmt.setString(1, member.getEmail());  
            pstmt.setString(2, member.getPassword());  
            pstmt.setString(3, member.getName());  
            pstmt.setTimestamp(4,  
                Timestamp.valueOf(member.getRegisterDateTime()));  
            // 생성한 PreparedStatement 객체 리턴  
            return pstmt;  
        }  
    }, keyHolder);  
  
    Number keyValue = keyHolder.getKey();  
    member.setId(keyValue.longValue());  
}
```

Spring MyBatis

MyBatis?

- 객체와 테이블을 매팅하는 ORM과는 다르게 객체와 SQL문을 매팅하는 프레임워크
 - 독자적인 질의 언어를 지원하지 않고 SQL 사용
- 장점
 - 저수준 JDBC 코드의 복잡성 제거
 - 친근한 SQL 기반으로 초기 학습곡선 완만
 - 기존 데이터베이스와 호환성 높음
 - Spring 프레임워크와 통합 기능 제공
 - 성능 우수

MyBatis - 스프링 연동

- 의존성 패키지 등록 (pom.xml)
 - MyBatis 패키지 등록

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.2.8</version>
</dependency>
```

- MyBatis – Spring 연동 패키지 등록

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.2.2</version>
</dependency>
```

MyBatis - 스프링 연동

- 스프링 설정 파일 작성 (/resources/mybatis-config.xml)

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

<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

    <environments default="development">
        <environment id="development">
            <transactionManager type="JDBC" />
            <dataSource type="POOLED">
                <property name="driver" value="com.mysql.jdbc.Driver" />
                <property name="url" value="jdbc:mysql://localhost:3306/demoweb" />
                <property name="username" value="devadmin" />
                <property name="password" value="mysql" />
            </dataSource>
        </environment>
    </environments>

    <mappers>
        <mapper resource="com/example/springmybatis/BoardMapper.xml" />
    </mappers>

</configuration>
```

MyBatis - 스프링 연동

▪ MyBatis 클래스

종류	설명
SqlSessionFactoryBuilder	<ul style="list-style-type: none">SqlSessionFactory를 생성하는 객체mybatis-config.xml 설정 파일 사용
SqlSessionFactory	<ul style="list-style-type: none">SqlSession을 생성하는 객체
SqlSession	<ul style="list-style-type: none">SQL을 실행하는 객체Mapper 파일의 맵핑 정보 사용

▪ MyBatis 연동 스프링 클래스

종류	설명
SqlSessionFactoryBean	SqlSessionFactoryBuilder를 사용해서 SqlSessionFactory 생성 스프링의 FactoryBean 인터페이스 구현 → getObject 메서드 재정의를 통해 SqlSessionFactory 객체 반환
SqlSessionTemplate	SqlSession을 Wrapping한 객체

MyBatis - 스프링 연동

- MyBatis - 스프링 연동 빈 등록
 - SqlSessionFactory

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="classpath:mybatis-config.xml"/>
</bean>
```

- SqlSessionTemplate

```
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>
```

MyBatis 설정 파일

▪ 구성 요소

종류	설명
environments	데이터 소스 및 트랜잭션 관리자 환경 설정
mappers	SQL 매팅 파일 또는 인터페이스 경로 지정
properties	다른 요소에서 재사용 할 수 있도록 설정 정보를 분리 정의
typeAliases	클래스의 전체 경로명 대신 사용할 별명 지정
typeHandlers	Java와 JDBC 타입 사이의 변환을 처리하는 핸들러 설정
settings	MyBatis 행위를 조정하기 위한 값을 설정
objectFactory	MyBatis가 결과 객체인 인스턴스를 생성하기 위해 제공하는 objectFactory를 커스터마이징하는 클래스를 사용하는 경우에 설정 커스터마이징 클래스는 DefaultFactory 클래스 상속
plugins	MyBatis가 매팅을 수행할 때 사용하는 메서드 호출을 가로채기 위한 플러그인을 커스터마이징하는 클래스를 사용하는 경우에 설정 커스터마이징 클래스는 Intercept 인터페이스 구현

MyBatis 매핑 설정 파일

▪ 매핑 설정 파일 형식

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

<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.ensoa.order.entity.mapper.ProductMapper">

    <resultMap id="ProductResult" type="com.ensoa.order.entity.ProductEntity" >..</resultMap>

    <select id="findAll" resultType="ProductEntity">..</select>

    <select id="findById" parameterType="long" resultType="ProductEntity">..</select>

    <update id="update" parameterType="ProductEntity">..</update>

    <delete id="delete" parameterType="long">..</delete>

    <insert id="insert" parameterType="ProductEntity" ..>..</insert>

</mapper>
```

MyBatis를 이용한 SQL 실행

- SqlSessionTemplate 의존성 주입

```
@Repository("productRepository")
public class ProductRepositoryMyBatis implements ProductRepository {
    @Autowired
    private SqlSessionTemplate sessionTemplate;
```

```
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg index="0" ref="sqlSessionFactory" />
</bean>
```

MyBatis를 이용한 SQL 실행

- Mapper Interface 구현 객체 의존성 주입

```
public interface ProductMapper {  
    List<ProductEntity> findAll();  
    List<ProductEntity> findAll(RowBounds rowBounds);  
    ProductEntity findById(long id);  
    void insert(ProductEntity Product);  
    void update(ProductEntity Product);  
    void delete(long id);  
}
```

매핑 설정 파일의 Namespace와 일치하는 패키지. 인터페이스를 만들고 <select>, <insert> 등의 요소에 부여된 id와 일치하는 추상메서드 선언

```
<bean id="productMapper" class="org.mybatis.spring.mapper.MapperFactoryBean">  
    <property name="mapperInterface"  
        value="com.ensoa.order.entity.mapper.ProductMapper" />  
    <property name="sqlSessionFactory" ref="sqlSessionFactory" />  
</bean>
```

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  
    <property name="dataSource" ref="dataSource" />  
    <property name="configLocation" value="classpath:mybatis-config.xml"/>  
</bean>
```

```
@Repository("productRepository")  
public class ProductRepositoryMyBatis implements ProductRepository {  
    @Autowired  
    private ProductMapper mapper;
```

Select Mapping XML

▪ 형식 및 속성

```
<select  
    id="selectPerson"  
    parameterType="int"  
    parameterMap="deprecated"  
    resultType="hashmap"  
    resultMap="personResultMap"  
    flushCache="false"  
    useCache="true"  
    timeout="10000"  
    fetchSize="256"  
    statementType="PREPARED"  
    resultSetType="FORWARD_ONLY">
```

속성	설명
id	구문을 찾기 위해 사용될 수 있는 명명 공간내 유일한 구분자
parameterType	구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭
resultType	이 구문에 의해 리턴되는 기대타입의 패키지 경로를 포함한 전체 클래스명이나 별칭.
resultMap	외부 resultMap 의 참조명.
flushCache	이 값을 true 로 설정하면, 구문이 호출될 때 마다 캐시 제거
useCache	이 값을 true 로 셋팅하면, 구문의 결과를 캐시
timeout	예외가 던져지기 전에 데이터베이스의 요청 결과를 기다리는 최대시간 설정
fetchSize	지정된 수만큼의 결과를 리턴하도록 하는 드라이버 힌트 값.
statementType	STATEMENT, PREPARED 또는 CALLABLE 중 하나를 선택할 수 있다.
resultSetType	FORWARD_ONLY SCROLL_SENSITIVE SCROLL_INSENSITIVE 중 하나를 선택할 수 있다.

Select 맵핑

- 사용 사례 (다중 행 반환 - 단순 맵핑)

```
public class ProductEntity {  
    private long id;  
    private String name;  
    private int price;  
    private String description;
```

```
<select id="findAll" resultType="ProductEntity">  
    SELECT PRODUCT_ID AS ID, NAME, PRICE, DESCRIPTION FROM PRODUCT  
</select>
```

```
public List<ProductEntity> findAll() {  
    List<ProductEntity> products =  
        sessionTemplate.selectList("com.ensoa.order.entity.mapper.ProductMapper.findAll");  
    return products;  
}
```

Select 맵핑

- 사용 사례 (다중 행 반환 - 맵핑 인터페이스 사용)

```
public class ProductEntity {  
    private long id;  
    private String name;  
    private int price;  
    private String description;
```

```
<mapper namespace="com.ensoa.order.entity.mapper.ProductMapper">  
    <select id="findAll" resultType="ProductEntity">  
        SELECT PRODUCT_ID AS ID, NAME, PRICE, DESCRIPTION FROM PRODUCT  
    </select>
```

```
public interface ProductMapper {  
    List<ProductEntity> findAll();  
    List<ProductEntity> findAll(RowBounds rowBounds);  
    ProductEntity findById(long id);  
    void insert(ProductEntity Product);  
    void update(ProductEntity Product);  
    void delete(long id);  
}
```

```
@Override  
public List<ProductEntity> findAll() {  
    List<ProductEntity> products = mapper.findAll();  
    return products;  
}
```

Select 맵핑

- 사용 사례 (단일 행 반환 - 맵핑 인터페이스 사용)

```
public class ProductEntity {  
    private long id;  
    private String name;  
    private int price;  
    private String description;
```

```
<select id="findById" parameterType="long" resultType="ProductEntity">  
    SELECT PRODUCT_ID AS ID, NAME, PRICE, DESCRIPTION FROM PRODUCT WHERE PRODUCT_ID = #{id}  
</select>
```

```
public interface ProductMapper {  
    List<ProductEntity> findAll();  
    List<ProductEntity> findAll(RowBounds rowBounds);  
    ProductEntity findById(long id);  
    void insert(ProductEntity Product);  
    void update(ProductEntity Product);  
    void delete(long id);  
}
```

```
@Override  
public ProductEntity findOne(long id) {  
    ProductEntity product = mapper.findById(id)  
    return product;  
}
```

Insert, Update, Delete Mapping XML

▪ Insert, Update, Delete

```
<insert
    id="insertAuthor"
    parameterType="domain.blog.Author"
    flushCache="true"
    statementType="PREPARED"
    keyProperty=""
    keyColumn=""
    useGeneratedKeys=""
    timeout="20">

<update
    id="insertAuthor"
    parameterType="domain.blog.Author"
    flushCache="true"
    statementType="PREPARED"
    timeout="20">

<delete
    id="insertAuthor"
    parameterType="domain.blog.Author"
    flushCache="true"
    statementType="PREPARED"
    timeout="20">
```

속성	설명
id	구문을 찾기 위해 사용될 수 있는 명명공간내 유일한 구분자
parameterType	구문에 전달될 파라미터의 패키지 경로를 포함한 전체 클래스명이나 별칭
flushCache	이 값을 true 로 셋팅하면 구문이 호출될 때마다 캐시 제거
timeout	예외가 던져지기 전에 데이터베이스의 요청 결과를 기다리는 최대시간을 설정
statementType	STATEMENT, PREPARED 또는 CALLABLE 중 하나를 선택
useGeneratedKeys	(입력(insert, update)에만 적용) 데이터베이스에서 내부적으로 생성한 키(예를 들어, MySQL 또는 SQL Server 와 같은 RDBMS 의 자동 증가 필드)를 받는 JDBC getGeneratedKeys 메서드를 사용하도록 설정하다. 디폴트는 false 이다.
keyProperty	(입력(insert, update)에만 적용) getGeneratedKeys 메서드나 insert 구문의 selectKey 하위 요소에 의해 리턴된 키를 셋팅할 프로퍼티를 지정..
keyColumn	(입력(insert, update)에만 적용) 생성키를 가진 테이블의 칼럼명을 셋팅. 키 칼럼이 테이블이 첫번째 칼럼이 아닌 데이터베이스에서만 필요

Insert 매핑

- 사용 사례 (매핑 인터페이스 사용)

```
public class ProductEntity {  
    private long id;  
    private String name;  
    private int price;  
    private String description;
```

```
<insert id="insert" parameterType="ProductEntity"  
        useGeneratedKeys="true" keyProperty="id">  
    INSERT INTO PRODUCT(NAME, PRICE, DESCRIPTION) VALUES (#{name}, #{price}, #{description})  
</insert>
```

```
public interface ProductMapper {  
    List<ProductEntity> findAll();  
    List<ProductEntity> findAll(RowBounds rowBounds);  
    ProductEntity findByPk(long id);  
    void insert(ProductEntity product);  
    void update(ProductEntity product);  
    void delete(long id);  
}
```

```
@Override  
public void save(ProductEntity product) {  
    mapper.insert(product);  
}
```

Update 매팅

▪ 사용 사례 (매팅 인터페이스 사용)

```
public class ProductEntity {  
    private long id;  
    private String name;  
    private int price;  
    private String description;
```

```
<update id="update" parameterType="ProductEntity">  
    UPDATE PRODUCT  
    SET NAME = #{name}, PRICE = #{price}, DESCRIPTION = #{description}  
    WHERE PRODUCT_ID = #{id}  
</update>
```

```
public interface ProductMapper {  
    List<ProductEntity> findAll();  
    List<ProductEntity> findAll(RowBounds rowBounds);  
    ProductEntity findById(long id);  
    void insert(ProductEntity Product);  
    void update(ProductEntity Product);  
    void delete(long id);  
}
```

```
@Override  
public void update(ProductEntity product) {  
    mapper.update(product);  
}
```

Delete 맵핑

■ 사용 사례 (맵핑 인터페이스 사용)

```
<delete id="delete" parameterType="long">
    DELETE FROM PRODUCT WHERE PRODUCT_ID = #{id}
</delete>
```

```
public interface ProductMapper {
    List<ProductEntity> findAll();
    List<ProductEntity> findAll(RowBounds rowBounds);
    ProductEntity findById(long id);
    void insert(ProductEntity Product);
    void update(ProductEntity Product);
    void delete(long id);
}
```

```
@Override
public void update(ProductEntity product) {
    mapper.update(product);
}
```

자동 증가 컬럼 처리

- Generate Key 설정 : 컬럼 값 자동 생성 처리

자동 증가 컬럼

```
<insert id="insertAuthor" useGeneratedKeys="true"  
    keyProperty="id">  
    insert into Author (username,password,email,bio)  
    values (#{username},#{password},#{email},#{bio})  
</insert>
```

생성된 데이터로 설정

```
<insert id="insertAuthor">  
    <selectKey keyProperty="id" resultType="int" order="BEFORE">  
        select CAST(RANDOM()*1000000 as INTEGER) a from SYSIBM.SYSDUMMY1  
    </selectKey>  
    insert into Author  
        (id, username, password, email, bio, favourite_section)  
    values  
        (#{id}, #{username}, #{password}, #{email}, #{bio}, #{favouriteSection,jdbcType=VARCHAR})  
</insert>
```

ResultMap

- 매핑 설정 파일에 작성한 SQL과 객체의 필드 사이의 이름 불일치와 같은 문제를 해결하기 위해 사용자 정의 매핑 설정인 ResultMap 사용

```
<resultMap id="ProductResult" type="com.ensoa.order.entity.ProductEntity">
    <id property="id" column="product_id"/>
    <result property="name" column="name"/>
    <result property="price" column="price"/>
    <result property="description" column="description"/>
</resultMap>

<select id="findAll" resultType="ProductEntity">
    SELECT PRODUCT_ID, NAME, PRICE, DESCRIPTION FROM PRODUCT
</select>
```

Parameter 형식

- 단일 값 전달인자

```
<select id="selectUsers" resultType="User">
    select id, username, password
    from users
    where id = #{id}
</select>
```

- 객체 전달인자

```
<insert id="insertUser" parameterType="User">
    insert into users (id, username, password)
    values (#{id}, #{username}, #{password})
</insert>
```

- HashMap 전달인자

```
<insert id="insertUser" parameterType="hashMap">
    INSERT INTO user (id, username, password)
    VALUES (#{id}, #{username}, #{password})
</insert>
```

SQL 재사용

- <sql> 요소로 재사용 가능한 SQL 선언

```
<sql id="userColumns"> id,username,password </sql>
```

```
<select id="selectUsers" resultType="map">
    select <include refid="userColumns"/>
    from some_table
    where id = #{id}
</select>
```

1 : 1 관계 매핑

- 데이터베이스 테이블 간의 1 : 1 관계를 객체에 매핑하기 위해 다음과 같은 방법 사용

방식	설명
포함 방식	<ul style="list-style-type: none">결과 맵에 포함하는 객체의 필드를 모두 포함하는 방식
중첩 결과 방식	<ul style="list-style-type: none"><association>요소를 사용포함하는 객체에 대한 별도의 ResultMap을 참조하는 방식
중첩 SELECT 방식	<ul style="list-style-type: none"><association>요소를 사용포함하는 객체를 조회하는 <select> 요소를 참조하는 방식

1 : 1 관계 매핑

- 포함 방식

```
public class OrderItemEntity {  
    private long id;  
    private int amount;  
    private ProductEntity product;
```

```
public class ProductEntity {  
    private long id;  
    private String name;  
    private int price;  
    private String description;
```

```
<resultMap id="OrderItemResultEmbedded" type="OrderItemEntity">  
    <id property="id" column="order_item_id" />  
    <result property="amount" column="amount"/>  
    <result property="product.id" column="product_id"/>  
    <result property="product.name" column="name"/>  
    <result property="product.price" column="price"/>  
    <result property="product.description" column="description"/>  
</resultMap>  
  
<select id="findAll" resultMap="OrderItemResultEmbedded">  
    SELECT ORDER_ITEM_ID, AMOUN, ORDER_ID,  
          PRODUCT.PRODUCT_ID, NAME, PRICE, DESCRIPTION  
    FROM ORDER_ITEM  
    LEFT INNER JOIN PRODUCT  
    ON ORDER_ITEM.PRODUCT_ID = PRODUCT_PRODUCT_ID  
</select>
```

1 : 1 관계 매핑

▪ 중첩 결과 방식

```
public class OrderItemEntity {  
    private long id;  
    private int amount;  
    private ProductEntity product;
```

```
public class ProductEntity {  
    private long id;  
    private String name;  
    private int price;  
    private String description;
```

```
<resultMap id="OrderItemResultNested" type="OrderItemEntity">  
    <id property="id" column="order_item_id" />  
    <result property="amount" column="amount"/>  
    <association property="product" column="product_id"  
        resultMap="com.ensoa.order.entity.mapper.ProductMapper.ProductResult"/>  
</resultMap>  
<select id="findAll" resultMap="OrderItemResultNested">  
    SELECT ORDER_ITEM_ID, AMOUN, ORDER_ID,  
        PRODUCT.PRODUCT_ID, NAME, PRICE, DESCRIPTION  
    FROM ORDER_ITEM  
    LEFT INNER JOIN PRODUCT  
    ON ORDER_ITEM.PRODUCT_ID = PRODUCT_PRODUCT_ID  
</select>
```

```
<mapper namespace="com.ensoa.order.entity.mapper.ProductMapper">  
    <resultMap id="ProductResult" type="com.ensoa.order.entity.ProductEntity" >  
        <id property="id" column="product_id"/>  
        <result property="name" column="name"/>  
        <result property="price" column="price"/>  
        <result property="description" column="description"/>  
</resultMap>
```

1 : 1 관계 매핑

- 중첩 select 방식

```
public class OrderItemEntity {  
    private long id;  
    private int amount;  
    private ProductEntity product;
```

```
public class ProductEntity {  
    private long id;  
    private String name;  
    private int price;  
    private String description;
```

```
<resultMap id="OrderItemResult" type="OrderItemEntity">  
    <id property="id" column="order_item_id" />  
    <result property="amount" column="amount"/>  
    <association property="product" column="product_id"  
        select="com.ensoa.order.entity.mapper.ProductMapper.findById"/>  
</resultMap>  
<select id="findAll" resultMap="OrderItemResult">  
    SELECT * FROM ORDER_ITEM  
</select>
```

```
<mapper namespace="com.ensoa.order.entity.mapper.ProductMapper">  
    <select id="findById" parameterType="long" resultType="ProductEntity">  
        SELECT PRODUCT_ID AS ID, NAME, PRICE, DESCRIPTION FROM PRODUCT WHERE PRODUCT_ID = #{id}  
    </select>
```

1 : Many 관계 매핑

- 데이터베이스 테이블 간의 1 : Many 관계를 객체에 매핑하기 위해 다음과 같은 방법 사용

방식	설명
중첩 결과 방식	<ul style="list-style-type: none"><collection>요소를 사용포함하는 객체에 대한 별도의 ResultMap을 참조하는 방식
중첩 SELECT 방식	<ul style="list-style-type: none"><collection>요소를 사용포함하는 객체를 조회하는 <select> 요소를 참조하는 방식

동적 SQL

- 애플리케이션 실행 시점에 수행할 SQL문을 결정하는 방식
 - 검색 조건을 사용자 선택하는 경우
 - Update 수행 시 선택적으로 특정 컬럼만 변경하는 경우
- 종류

종류	구문
if	<if test="name != null"> </if>
choose when otherwise	<choose> <when test="condition == 'one'"> </when> <when test="condition == 'one'"> </when> <otherwise> </otherwise> </choose>

종류	구문
where	<where> <if test="name != null"></if> </where>
trim	생략
foreach	<foreach item="item" collection="items" open="("seperator=", " close=")"> </foreach>
set	<set> <if test="name != null"></if> </set>

동적 SQL 예제

```
<select id="find"
    parameterType="com.ensoa.order.domain.CustomerSearch" resultMap="CustomerResult">
    SELECT * FROM CUSTOMER
    <trim prefix="WHERE" prefixOverrides="AND / OR">
        <where>
            <if test="name != null">
                NAME LIKE #{name}
            </if>
            <if test="address != null">
                AND ADDRESS LIKE #{address}
            </if>
            <if test="email != null">
                AND EMAIL LIKE #{email}
            </if>
        </where>
    </trim>
</select>
<update id="update" parameterType="CustomerEntity">
    UPDATE CUSTOMER
    <set>
        <if test="name != null">NAME = #{name}, </if>
        <if test="address != null">ADDRESS = #{address}, </if>
        <if test="email != null">EMAIL = #{email} </if>
    </set>
    WHERE CUSTOMER_ID = #{id}
</update>
```

기타 구문

- 저장 프로시저 호출

- 형식 : { CALL 저장프로시저이름(전달인자목록)}

```
<select id="findBySp" parameterType="string" resultMap="CustomerResult"
        statementType="CALLABLE">
    { CALL GET_CUSTOMERS(#{name, mode=IN, jdbcType=VARCHAR}) }
</select>
```

- 페이지 조회

```
@Override
public List<CustomerEntity> findAll(Pageable page) {
    RowBounds rowBounds = new RowBounds(page.getIndex(), page.getSize());
    List<CustomerEntity> customers = mapper.findAll(rowBounds);
    return customers;
}
```

어노테이션을 이용한 매팅

- 매퍼 인터페이스에 적용
- 종류

종류	매팅 구문
@Select	<select></select>
@Insert	<insert></insert>
@Update	<update></update>
@Delete	<delete></delete>
@Options	<insert useGeneratedKeys="true" keyProperty="prop"></insert> 저장프로시저 호출
@SelectKey	<selectKey></selectKey>
@Results	<resultMap></resultMap>
@Result	<id></id> 또는 <result></result>
@One, @Many	1:1 또는 1:Many 관계 매팅
@SelectProvider @InsertProvider @UpdateProvider @DeleteProvider	동적 SQL 매팅

어노테이션을 이용한 매팅

- Insert, update, delete, select

```
@Select("SELECT PRODUCT_ID AS ID, NAME, PRICE, DESCRIPTION FROM PRODUCT")
List<ProductEntity> findAll();

@Insert("INSERT INTO PRODUCT(NAME, PRICE, DESCRIPTION) " +
        "VALUES (#{name}, #{price}, #{description})")
@Options(useGeneratedKeys=true, keyProperty="prodId")
void insert(ProductEntity Product);

@Update("UPDATE PRODUCT " +
        "SET NAME = #{name}, PRICE = #{price}, DESCRIPTION = #{description} " +
        "WHERE PRODUCT_ID = #{id}")
void update(ProductEntity Product);

@Delete("DELETE FROM PRODUCT WHERE PRODUCT_ID = #{id}")
void delete(long id);
```

자동증가컬럼

- ResultMap

```
@Select("SELECT PRODUCT_ID AS ID, NAME, PRICE, DESCRIPTION " +
        "FROM PRODUCT WHERE PRODUCT_ID=#{id}")
@Results( {
    @Result(id=true, property="id", column="product_id"),
    @Result(property="name", column="name"),
    @Result(property="price", column="price"),
    @Result(property="description", column="description")
})
ProductEntity findById(long id);
```

어노테이션을 이용한 매팅

▪ ResultMap

```
@Select("SELECT PRODUCT_ID AS ID, NAME, PRICE, DESCRIPTION " +  
       "FROM PRODUCT WHERE PRODUCT_ID=#{id}")  
@Results( {  
    @Result(id=true, property="id", column="product_id"),  
    @Result(property="name", column="name"),  
    @Result(property="price", column="price"),  
    @Result(property="description", column="description")  
})  
ProductEntity findById(long id);
```

▪ 관계 Cardinality

```
@Select("SELECT * FROM ORDERS WHERE ORDERS.ORDER_ID = #{orderId}")  
@Results( {  
    @Result(id=true, property="id", column="order_id" ),  
    @Result(property="orderDate", column="order_date"),  
    @Result(property="customer", column="customer_id",  
           one=@One(select="com.ensoa.order.entity.mapper.CustomerMapper.findById")),  
    @Result(property="items", column="order_id",  
           many=@Many(select="com.ensoa.order.entity.mapper.OrderItemMapper..findById"))  
})  
OrderEntity findById(long id);
```

어노테이션을 이용한 매팅

■ 동적 SQL

```
public String find(final CustomerSearch customerSearch) {  
    return new SQL() {  
        SELECT("*");  
        FROM("CUSTOMER");  
        if(customerSearch.getName() != null) {  
            WHERE("NAME LIKE #{name}");  
        }  
        if(customerSearch.getAddress() != null) {  
            WHERE("ADDRESS LIKE #{address}");  
        }  
        if(customerSearch.getEmail() != null) {  
            WHERE("EMAIL LIKE #{email}");  
        }  
    }.toString();  
}
```

```
@SelectProvider(type=CustomerSqlProvider.class, method="find")  
@ResultMap("com.ensoa.order.entity.mapper.CustomerMapper.CustomerResult")  
List<CustomerEntity> find(CustomerSearch customerSearch);
```

■ 저장 프로시저 호출

```
@Select(value= "[CALL GET_CUSTOMERS(#{name, mode=IN, jdbcType=VARCHAR})]")  
@ResultMap("com.ensoa.order.entity.mapper.CustomerMapper.CustomerResult")  
@Options(statementType = StatementType.CALLABLE)  
List<CustomerEntity> findBySp(String name);
```

Spring Transaction

Transaction

- 한 개 이상의 물리적인 동작으로 구성된 논리적인 작업 단위
- 트랜잭션에 포함된 물리적인 작업이 모두 성공하거나 실패하도록 관리
- ACID 특성
 - 원자성 (Atomicity)
 - 일관성 (Consistency)
 - 격리성 (Isolation)
 - 영속성 (Durability)

JDBC transaction

- Connection 객체의 autoCommit 속성과 commit(), rollback() 메서드를 사용해서 구현

```
Connection conn = null;
PreparedStatement pstmt = null;

try {
    Class.forName("com.mysql.cj.jdbc.Driver");

    conn = DriverManager.getConnection(
        "jdbc:mysql://localhost:3306/demo?serverTimezone=UTC", "devuser", "mariadb");

    conn.setAutoCommit(false); //executeUpdate 실행 후에 자동으로 commit 하지 마세요

    // execute query

    conn.commit(); //마지막 commit 또는 rollback 실행 후에 처리된 모든 명령을 commit
    System.out.println("계좌 이제 성공");

} catch (Exception ex) {

    try { conn.rollback(); } catch (Exception ex2) {} //마지막 commit 또는 rollback 실행 후에 처리된 모든 명령을 rollback
    System.out.println("계좌 이제 실패");

} finally {
    try { conn.setAutoCommit(true); } catch (Exception ex) {}
    try { pstmt.close(); } catch (Exception ex) {}
    try { conn.close(); } catch (Exception ex) {}
}
```

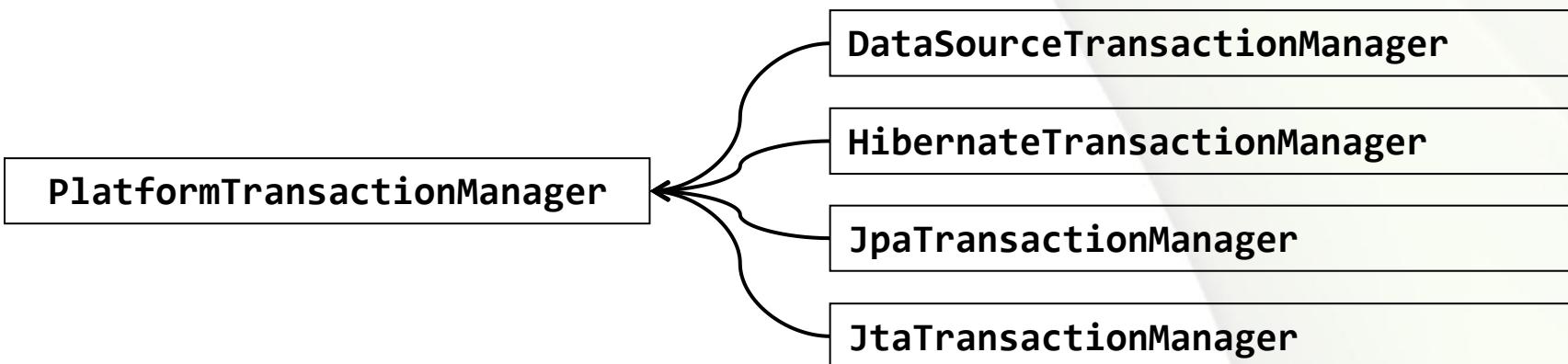
Spring 트랜잭션 지원

- 반복적인 트랜잭션 관리 코드 없이 간단한 코드로 트랜잭션 관리 가능하도록 지원
- 프로그래밍 방식
 - TransactionTemplate 클래스 사용
- AOP 방식
 - <tx:advice>와 <aop:advisor> 설정을 사용하는 선언적 트랜잭션 방식
- 어노테이션 방식
 - @Transactional 어노테이션을 사용하는 선언적 트랜잭션 방식
 - 내부에서는 AOP를 사용해서 트랜잭션 관리

Spring 트랜잭션 관리자

- 스프링 프레임워크는 데이터 연동 기술에 따라 특화된 트랜잭션 관리자 제공
 - 모든 트랜잭션 관리자는 PlatformTransactionManager 인터페이스 구현
- 종류

트랜잭션 관리자	설명
DataSourceTransactionManager	MyBatis와 같은 JDBC 기반 트랜잭션 관리
HibernateTransactionManager	Hibernate 프레임워크 기반 트랜잭션 관리
JpaTransasctionManager	JPA 기반 트랜잭션 관리
JtaTransactionManager	분산 트랜잭션 지원



트랜잭션 관리자 구성

- Annotation 사용

```
@Bean  
public PlatformTransactionManager transactionManager() {  
    DataSourceTransactionManager tm = new DataSourceTransactionManager();  
    tm.setDataSource(dataSource());  
    return tm;  
}
```

- XML 기반 설정

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

트랜잭션 구현 (XML 설정 기반)

▪ 빈 선언 및 의존성 주입

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

<bean id="txTemplate"
      class="org.springframework.transaction.support.TransactionTemplate">
    <property name="transactionManager" ref="txManager" />
</bean>

<bean id="memberDao" class="spring.MemberDao">
    <constructor-arg ref="dataSource" />
</bean>

<bean id="changePwdSvc" class="spring.ChangePasswordService">
    <property name="memberDao" ref="memberDao" />
    <property name="txTemplate" ref="txTemplate" />
</bean>
```

트랜잭션 구현 (XML 설정 기반)

▪ 트랜잭션 적용

```
public void changePassword(String email, String oldPwd, String newPwd) {  
    txTemplate.execute(new TransactionCallback<Void>() {  
  
        @Override  
        public Void doInTransaction(TransactionStatus txStatus) {  
  
            try {  
                Member member = memberDao.selectByEmail(email);  
                if (member == null)  
                    throw new MemberNotFoundException();  
  
                member.changePassword(oldPwd, newPwd);  
  
                memberDao.update(member);  
  
            } catch (Exception ex) {  
                txStatus.setRollbackOnly();  
            }  
  
            return null;  
        }  
    });  
}
```

트랜잭션 구현 (Annotation 기반)

```
@Configuration  
@EnableTransactionManagement  
public class AppCtx {  
  
    @Bean(destroyMethod = "close")  
    public DataSource dataSource() {  
        DataSource ds = new DataSource();  
        ds.setDriverClassName("com.mysql.jdbc.Driver");  
        ds.setUrl("jdbc:mysql://localhost:3306/spring5fs?characterEncoding=utf8");  
        ds.setUsername("spring5");  
        ds.setPassword("spring5");  
        ds.setInitialSize(2);  
        ds.setMaxActive(10);  
        ds.setTestWhileIdle(true);  
        ds.setMinEvictableIdleTimeMillis(60000 * 3);  
        ds.setTimeBetweenEvictionRunsMillis(10 * 1000);  
        return ds;  
    }  
  
    @Bean  
    public PlatformTransactionManager transactionManager() {  
        DataSourceTransactionManager tm = new DataSourceTransactionManager();  
        tm.setDataSource(dataSource());  
        return tm;  
    }  
  
    @Transactional(rollbackFor = ArithmeticException.class)  
    public void changePassword(String email, String oldPwd, String newPwd) {  
        Member member = memberDao.selectByEmail(email);  
        if (member == null)  
            throw new MemberNotFoundException();  
  
        member.changePassword(oldPwd, newPwd);  
  
        memberDao.update(member);  
    }  
}
```

@Transactional과 프록시

▪ 교재 220 ~ 223p 참고

- @Transactional 어노테이션을 이용해서 트랜잭션을 처리하면 내부적으로 AOP를 사용해서 트랜잭션 처리
- 과정
 - @EnableTransactionManagement 적용 → @Transactional이 적용된 bean을 찾아서 알맞은 프록시 객체 생성
 - @Transactional이 적용된 메서드를 호출하면 PlatformTransactionManager를 사용해서 트랜잭션 시작
 - 정상적으로 처리되면 자동으로 commit 처리
 - 예외가 발생하면 자동으로 rollback 처리
 - 단, RuntimeException 및 RuntimeException을 상속한 예외에 대해서만 자동으로 rollback 처리
 - 다른 종류의 예외에 대해서도 rollback을 처리하려면 @Transactional에 rollbackFor 속성에 예외 지정

트랜잭션 속성 ▪ 교재 223 ~ 227p 참고

▪ 속성 종류

속성 이름	설명
전파 행위	트랜잭션 영역 설정
분리 수준	한 트랜잭션이 다른 트랜잭션에 영향 받는 정도 설정
읽기 전용	데이터 읽기 최적화 적용
타임아웃	트랜잭션 타임아웃 시간 설정
롤백 규칙	롤백이 발생하는 예외 지정

트랜잭션 속성 ■ 교재 223 ~ 227p 참고

■ 전파 행위

설정 값	설명
MANDATORY	반드시 트랜잭션 안에서 메서드 실행. 트랜잭션이 없으면 예외 발생
NESTED	트랜잭션 안에서 실행될 경우 중첩된 트랜잭션으로 실행. 트랜잭션이 없는 경우 PROPAGATION_REQUIRED와 동일
NEVER	트랜잭션 없이 실행. 트랜잭션이 있으면 예외 발생
NOT_SUPPORTED	트랜잭션 없이 실행. 기존 트랜잭션이 있는 경우 트랜잭션을 호출한 메서드가 종료될 때까지 보류
REQUIRED	기존 트랜잭션이 있는 경우 기존 트랜잭션 안에서 실행되고, 없는 경우 새 트랜잭션 생성
REQUIRED_NEW	항상 새로운 트랜잭션 생성. 기존 트랜잭션이 있는 경우 기존 트랜잭션은 보류
SUPPORTED	기존 트랜잭션이 있으면 트랜잭션 안에서 실행. 기존 트랜잭션이 없으면 트랜잭션 없이 실행

트랜잭션 속성

▪ 교재 223 ~ 227p 참고

▪ 분리 수준

설정 값	설명
DEFAULT	현재는 ISOLATION_READ_COMMITTED와 동일
READ_UNCOMMITTED	트랜잭션에서 변경된 데이터를 커밋하기 전에 다른 트랜잭션에서 읽을 수 있음
READ_COMMITTED	트랜잭션에서 변경된 데이터를 커밋하기 전에 다른 트랜잭션에서 읽을 수 없음
REPEATABLE_READ	트랜잭션 내에서 여러 번의 읽기 수행이 항상 동일한 값을 반환하도록 보장
SERIALIZE	모든 트랜잭션은 한 번에 하나씩만 실행

트랜잭션 설정 가이드

- 트랜잭션은 서비스 레이어에서 시작
- 삽입, 삭제, 변경 기능을 수행하는 메서드에 기본 값인 TRANSACTION_REQUIRED를 지정해서 기존 트랜잭션에 참여하거나 또는 새 트랜잭션을 시작하도록 설정
 - 읽기 메서드에는 트랜잭션이 필요 없으므로 읽기 최적화 지정
- 웹 애플리케이션의 컨트롤러에 트랜잭션 설정을 피하는 것이 권장됨
- Repository 클래스에는 트랜잭션이 필요한 메서드에 TRANSACTION_SUPPORTED를 지정해서 기존 트랜잭션에 참여하도록 설정
- RuntimeException 에 대해서만 롤백 하도록 기본 설정 유지
 - 직접 예외 처리할 경우 자동으로 Rollback되지 않으므로 UnexpectedRollbackException 예외에서 명시적인 롤백 처리