

Chapter 01. 시작하기

01.스프링이란?

02.POJO 프로그래밍

03.스프링 기술

04.Spring @MVC 로 시작하기

자바 엔터프라이즈 개발을 편하게 해주는 오픈소스 경량급 애플리케이션 프레임워크

1.2 자바 엔터프라이즈 개발

1.2.1 기업 대상 애플리케이션 개발

- 은행(금융), 네이버, 물류/유통 회사, 병원
- Future Business

1.2.2 환경과 조건

- C/S (network)
- 웹 환경
- 데이터베이스
- 분산환경 (분산객체, 자원 관리, 컴포넌트)

1.2.2 JEE(Java Enterprise Edition)

- Servlet/JSP, JDBC, EJB, RMI, JNDI, JTA, JMS ...

1.3 프레임워크(Framework)

1.3.1 정의

- 소프트웨어를 만드는 데 기본이 되는 골격 코드
- 반제품
- 완전한 애플리케이션 소프트웨어가 아니다.
- 문제 영역(도메인)을 해결하기 위한 잘 설계된 재사용 가능한 모듈
- 확장하여 비즈니스 요구사항에 맞는 완전한 애플리케이션으로 완성이 요구된다.

1.3.2 종류(분류)

- 웹 애플리케이션 프레임워크
Struts, WebWork, Spring MVC
- 데이터베이스 애플리케이션 프레임워크
iBatis(MyBatis), Hibernate, Spring DAO
- 기타(지원) 프레임워크
로깅(Log4J), 빌드/배포(Ant), 단위테스트(JUnit)

1.3 프레임워크(Framework)

1.3.3 애플리케이션 프레임워크

- 특정 계층, 기술, 특정 비즈니스에 국한되지 않은 애플리케이션 전 영역을 포괄
- 개발 전 과정을 빠르고 편리하며 효율적으로 진행하는 것을 목표
- 자바 개발의 폭넓은 간소화
- EJB, Spring

1.3.4 EJB(Enterprise Java Bean)

- Java Bean 이란?
 1. 컴포넌트 기반의 소프트웨어 모델 스펙(1996년 12월 SUN)
 2. 자바 객체를 재사용 가능하게 즉, 컴포넌트화 시킬 수 있는 코딩 방침을 정의
 3. 자바 빈즈 스펙에 맞게 구현된 자바코드를 웹에서 쉽게 이용하기 위해 JSP 표준액션 태그 지원
예) <jsp:useBean>, <jsp:getProperty>, <jsp:setProperty> 지원
 4. 스펙의 일부
 - 디폴트 생성자 존재
 - 프로퍼티 변수는 private, protected로 정의
 - public 접근 지정자를 가지는 setXXX(), getXXX() 메소드 작성
 5. 엔터프라이즈 어플리케이션에서 필요한 보안, 트랜잭션, 분산 컴퓨팅 등의 서비스는 제공 않음

1.3 프레임워크(Framework)

1.3.4 EJB(Enterprise Java Bean)

– Enterprise Java Bean 이란?

1. 1998년 3월 Sun에서 엔터프라이즈급 어플리케이션 개발을 단순화하기 위해 발표한 스펙

2. 다수의 J2EE 서버 개발 벤더에서 EJB 스펙을 구현하여 WAS 제품 출시

예) BEA의 WebLogic, IBM의 WebSphere, TMax의 Jeus 등

3. 보안, 트랜잭션지원, 분산 컴퓨팅 등의 엔터프라이즈 어플리케이션 개발 시 필요한 다양한 서비스를 컨테이너에서 제공하며 개발자는 비즈니스 로직에 전념하도록 지원

4. 컨테이너의 다양한 서비스를 제공받기 위해서는 지켜야 하는 EJB 스펙 자체가 복잡 작성된 코드는 EJB 컨테이너가 없을 경우 사용할 수 없으며 EJB 컨테이너도 벤더에 따라 구현한 내용이 다르고 컨테이너가 변경될 경우 호환이 어렵다

1.3 프레임워크(Framework)

1.3.5 Spring

- 2002년 로드 존슨(Rod Johnson) 이 쓴 「Expert one-on-one:J2EE Design and Development」에서 소개된 소스코드를 기반으로 2003년 2월에 시작된 오픈 소스 프로젝트
- POJO(Plain Old Java Object) 특정클래스를 상속하거나 인터페이스를 구현하지 않는 평범한 자바 클래스(느슨한 Java Bean, Spring Bean)를 이용하며 단순하지만 EJB에서 제공하는 고급 기술을 제공한다.
- 진정한 의미의 자바 개발의 폭 넓은 간소화 를 실현한 프레임워크
- 20여 개의 모듈과 수십만 라인의 복잡하고 방대한 규모
- 불필요하게 무겁지 않다. (EJB와 비교)
- 코드는 단순하고 개발과정은 편리
- 고급 기능을 세련된 방식으로 적용
- 군더더기 없이 깔끔한 기술을 가진 "경량급" 프레임워크
- 비슷한 기술 수준에서 훨씬 빠르고 간편하게 작성이 가능

1.3.5 Spring

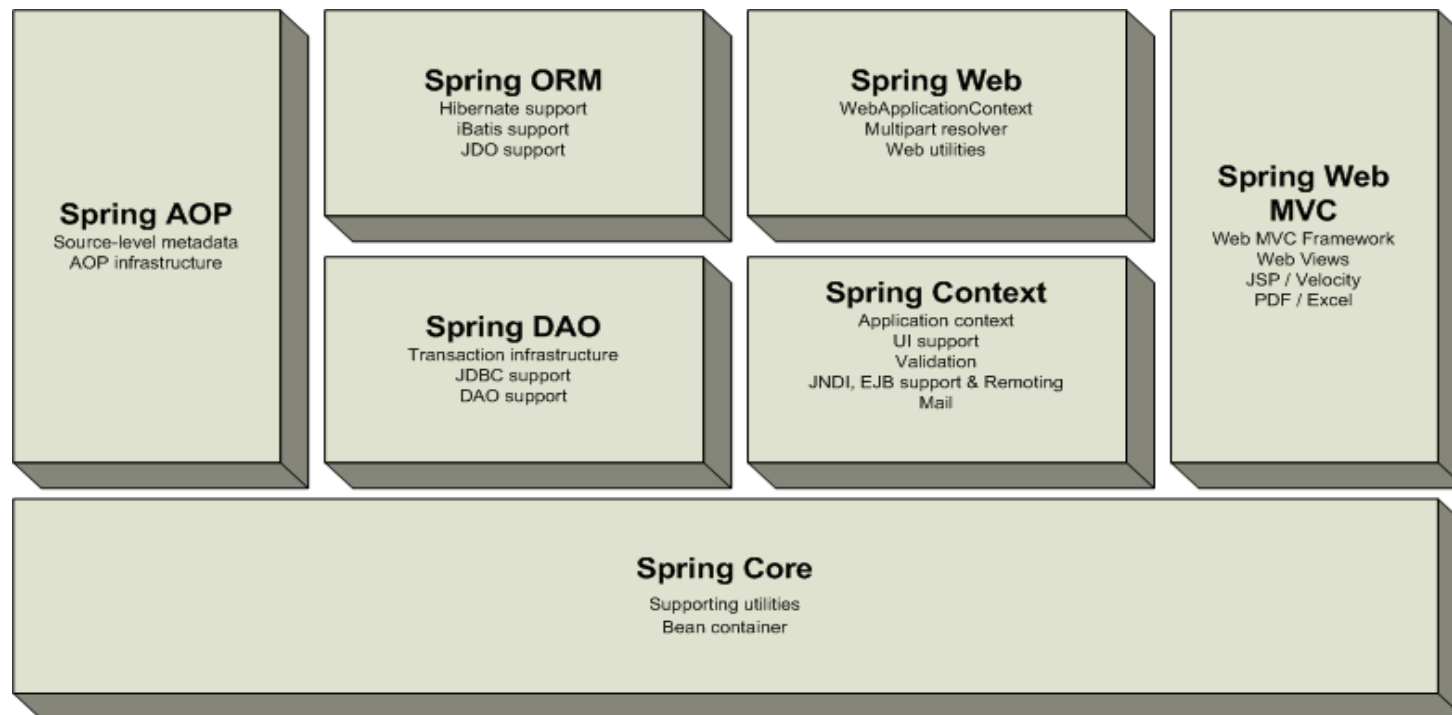
– 컨테이너(Container)

1. EJB의 비즈니스 서비스 컨테이너의 기능은 유지하되 복잡성을 제거한 컨테이너의 필요성
2. 객체들의 라이프사이클을 관리해주는 컨테이너의 기본적인 기능
3. 컨테이너에서 제공하는 API를 상속받거나 구현하여 코드를 작성하는 부분들을 제거
4. 컨테이너를 이루는 파일자체가 몇 메가 밖에 안 되는 작은 사이즈이며 구동에 필요한 시간이 짧고 자체 부하는 무시할 수준이고 컨테이너 내에 객체를 배치하는 복잡한 과정이 짧다.
5. 컨테이너의 필요성
 - 가) 컴포넌트, 객체의 자유로운 삽입이 가능하도록 하기 위한 호출의 독립성
 - 나) 서비스를 설정하거나 찾기 위한 일관된 방법을 제시
 - 다) 싱글톤이나 팩토리를 구현할 필요 없이 단일화된 객체에 대한 접근방법을 제공
 - 라) 비즈니스 객체에 부가적으로 필요한 각종 엔터프라이즈 서비스를 제공

1.3 프레임워크(Framework)

1.3.5 Spring

- 주요 모듈



1.3.5 Spring

- 주요 전략

1. POJO를 이용한 가볍고(lightweight) 비침투적(non-invasive) 개발
2. DI와 인터페이스 지향을 통한 느슨한 결합도(loose coupling)
3. Aspect와 공통 규약을 통한 선언적(declarative) 프로그래밍
4. Aspect와 템플릿(template)을 통한 반복적이고 상투적인(boilerplate) 코드 제거

Chapter 01. 시작하기

01.스프링이란?

02.POJO 프로그래밍

03.스프링 기술

04.Spring @MVC 로 시작하기

2.1 POJO 프로그래밍

2.1.1 POJO 란?

- (P)lain (O)ld (J)ava (O)bject
- 자바 언어와 꼭 필요한 API외에는 특정 규약에 종속되지 않는다.
- 특정 환경에 종속되지 않는다. (기술과 비즈니스 분리)
- 스프링에서는 스프링에 특화된 인터페이스 구현을 요구하지 않음
- 스프링 자체에 의존성이 높은 클래스 확장을 거의 요구 하지 않음

2.1.2 POJO 프로그래밍의 장점

- 스프링의 정수는 엔터프라이즈 개발에서 요구하는 모든 기술을 POJO를 통해 제공
- 비침투적 프로그램이 가능



Chapter 01. 시작하기

01.스프링이란?

02.POJO 프로그래밍

03.스프링 기술

04.Spring @MVC 로 시작하기

3.1 IoC(제어역전) 과 DI(의존관계 주입)

- 1) 스프링의 가장 기본이 되는 기술이자 스프링 핵심 개발 원칙
- 2) **Bean** : 스프링이 제어권을 가지고 직접 만들고 관계를 부여하는 오브젝트
- 3) 스프링 빈은 스프링 컨테이너가 생성과 관계 설정 등을 제어
- 4) IoC(DI) Container = Bean Factory = Application Context

3.2.1 AOP(Aspect Oriented Programming)

- 관점 지향 프로그래밍
- OOP를 더욱 더 OOP 답게 해 주는 (더욱 더 완벽하게 해 주는) 기술
- 관심의 분리 (Separation of Concern)
- 횡단 관심(Crosscutting Concern)과 핵심관심(Core Concern)
- 핵심관심 모듈과 횡단 관심 모듈이 긴밀하게 결합 (핵심 모듈이 필요한 시점에..)
- OOP 문제점 : 중복코드, 지저분한 코드, 생산성 저하, 재활용성의 문제점
- 필요한 시점에 횡단 관심 모듈을 삽입하여 동작하게 하는 기술.
- EJB AOP, JDK Dynamic Proxy, **AspectJ**, **Spring AOP**

3.2 AOP

3.2.2 AOP 개념

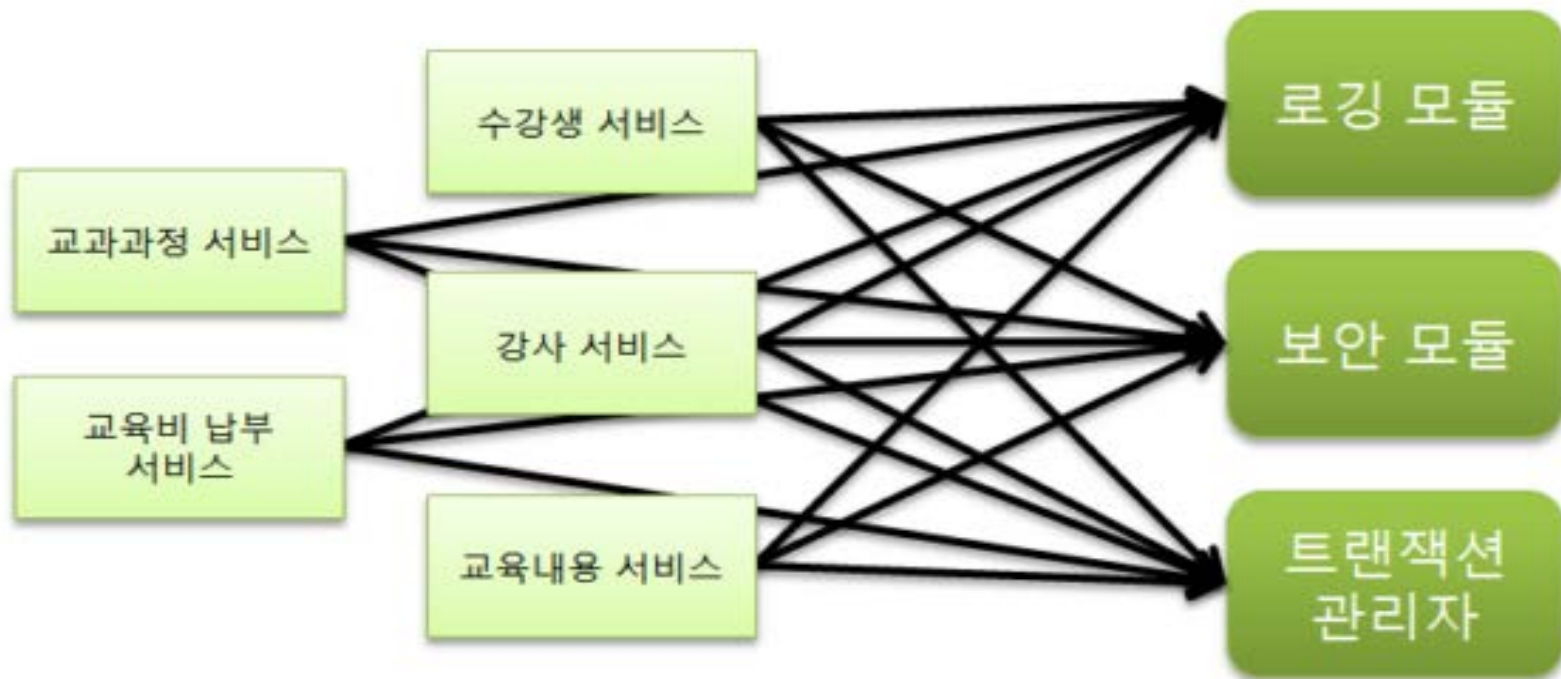
1) 관심의 산재

수강생 관리	강사 관리	교육 관리	...
보안...	보안...	보안...	
로깅...	로깅...	로깅...	
트랙잭션 시작...	트랙잭션 시작...	트랙잭션 시작...	
비즈니스 로직	비즈니스 로직	비즈니스 로직	
트랜잭션 끝...	트랜잭션 끝...	트랜잭션 끝...	...

3.2 AOP

3.2.2 AOP 개념

2) 관심의 모듈화



Chapter 01. 시작하기

01.스프링이란?

02.POJO 프로그래밍

03.스프링 기술

04.Spring @MVC 로 시작하기

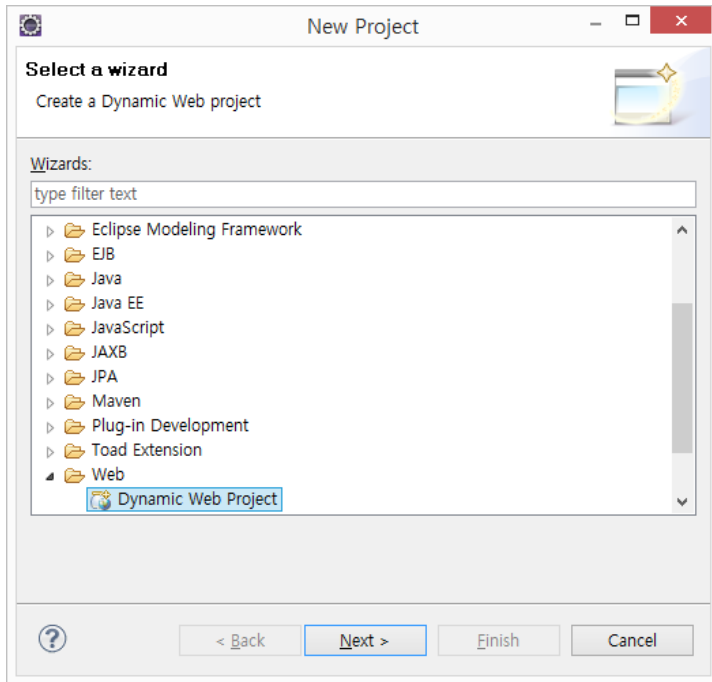
4.1 springex 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

Spring @MVC 기반의 springex을 같이 만들어 봅니다.

IoC/DI를 지원하는 Spring Container에 대해 생각해보고 Spring MVC기반의 웹 어플리케이션의 특징을 살펴 봅니다.

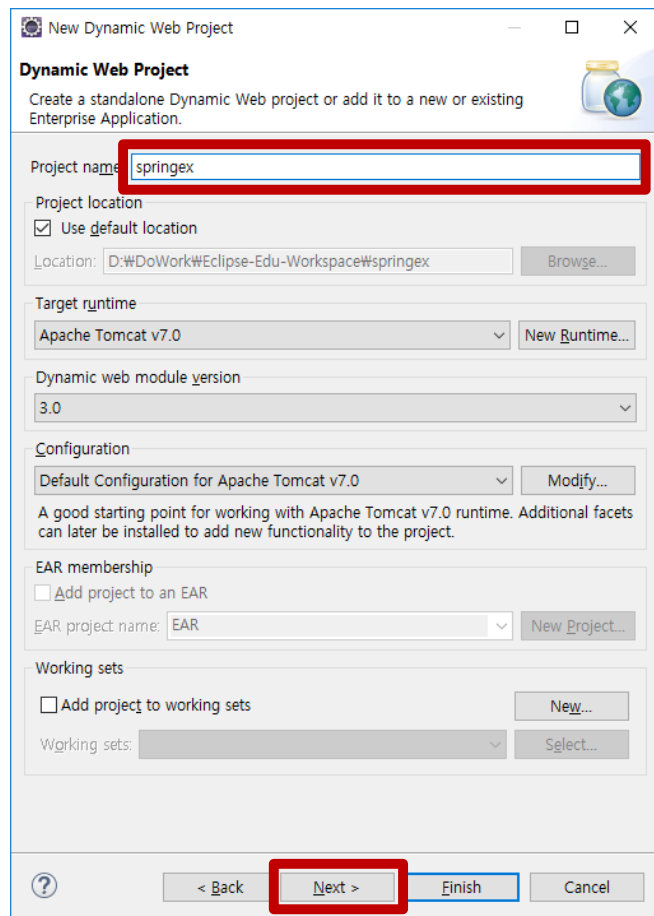
1. Dynamic Web Project 프로젝트 생성. (보통 스프링 프로젝트는 maven 빌드 기반의 maven 프로젝트로 생성)



4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

2. project name 은 springex



New Dynamic Web Project

Dynamic Web Project
Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name:

Project location
☒ Use default location
Location:

Target runtime

Dynamic web module version

Configuration

A good starting point for working with Apache Tomcat v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

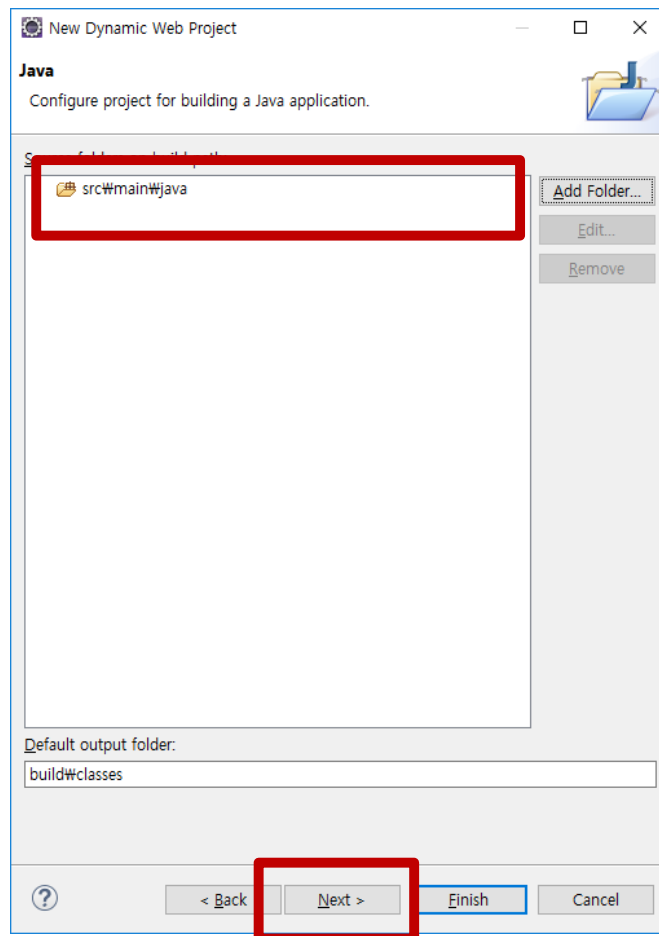
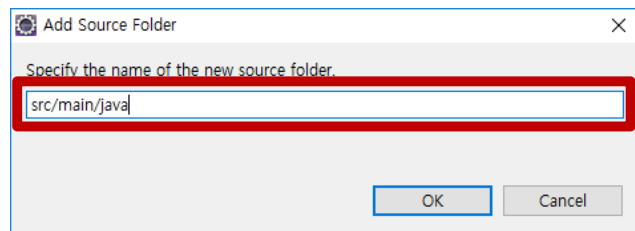
EAR membership
☐ Add project to an EAR
EAR project name:

Working sets
☐ Add project to working sets
Working sets:

4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

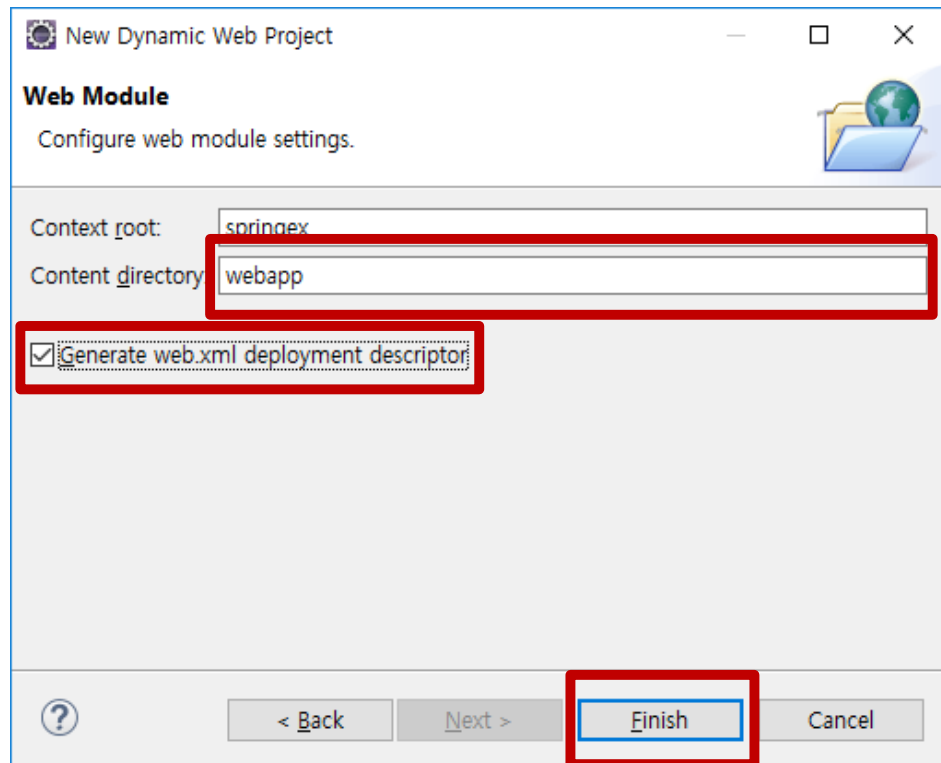
3. maven 웹 애플리케이션 프로젝트는 소스 폴더(src)가 Dynamic Web Project와 다르다. 따라서 지우고 새로 만든다.



4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

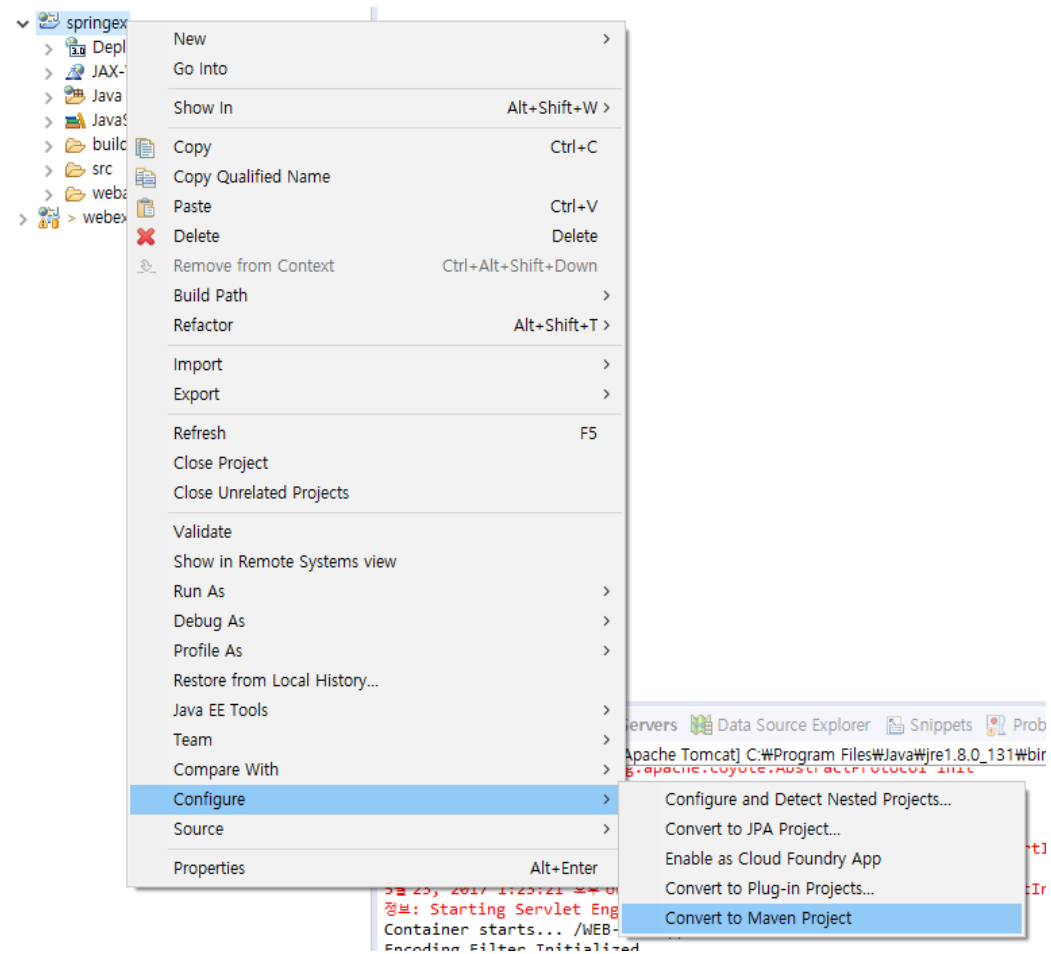
3. maven 웹 애플리케이션 프로젝트에서는 웹 콘텐츠 폴더(WebContent)도 Dynamic Web Project와 다르다. 따라서 지우고 새로 만든다.



4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

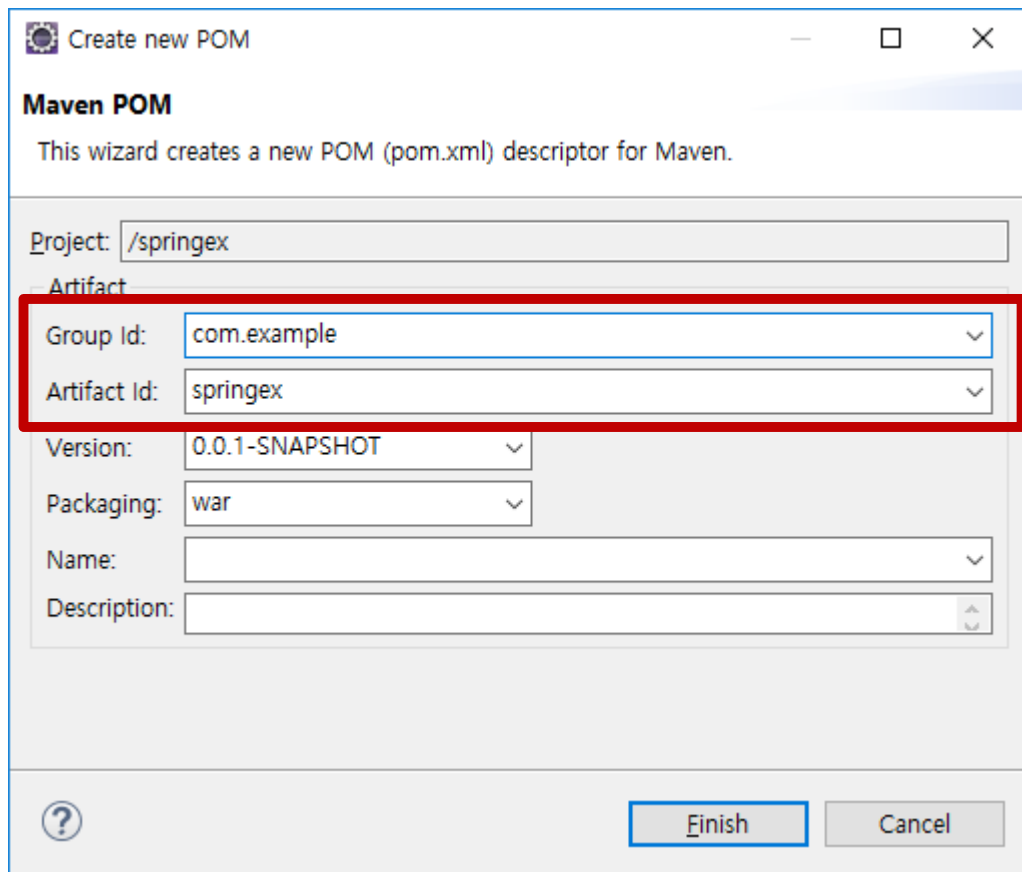
4. Maven 프로젝트로 바꾸기(Convert)



4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

4. Maven 프로젝트로 바꾸기(Convert)



Create new POM

Maven POM

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /springex

Artifact

Group Id: com.example

Artifact Id: springex

Version: 0.0.1-SNAPSHOT

Packaging: war

Name:

Description:

Finish Cancel

4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

5. 프로젝트 의존성(라이브러리) 추가 (pom.xml)

1) Spring Core Library 추가

```
<dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-context</artifactId>  
    <version>4.2.1.RELEASE</version>  
</dependency>
```

2) Spring Web Library 추가

```
<dependency>  
    <groupId>org.springframework</groupId>  
    <artifactId>spring-web</artifactId>  
    <version>4.2.1.RELEASE</version>  
</dependency>
```

4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

5. 프로젝트 의존성(라이브러리) 추가 (pom.xml)

3) Spring MVC Library 추가

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.2.1.RELEASE</version>
</dependency>
```

4) 라이브러리 버전 프로퍼티로 관리 하기

```
<properties>
    <org.springframework-version>4.2.1.RELEASE</org.springframework-version>
</properties>
```

org.springframework-version 프로퍼티를 추가한 후, 각 각 dependency의 버전을 다음과 같이 수정한다.

```
<version>${org.springframework-version}</version>
```

4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

6. DispatcherServlet에 대한 서블릿 매핑 추가(web.xml)

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

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

4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

7. 웹 애플리케이션 컨텍스트 설정 (/WEB-INF/spring-servlet.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
http://www.springframework.org/schema/mvc/spring-mvc.xsd
    http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop.xsd
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config />
    <context:component-scan base-package="com.example.springex.controller" />

</beans>
```

4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

8. Controller 작성하기

```
@Controller
public class HelloController {

    @RequestMapping( "/hello" )
    public ModelAndView hello( @RequestParam String name ) {

        ModelAndView mav = new ModelAndView();
        mav.addObject( "hello", "Hello " + name );
        mav.setViewName( "/WEB-INF/views/index.jsp" );

        return mav;
    }
}
```

4.1 hellospring 웹 애플리케이션 작성

[실습예제] springex 웹 애플리케이션 작성하기

9. 실행 및 생각해 볼 것 들

- 추가적으로 해준 것들

- 1) pom.xml 구성
- 2) DispatcherServlet 등록(web.xml)
- 3) 서블릿 애플리케이션 컨텍스트 설정 (spring-servlet.xml)
- 4) Controller 작성

- 생략된 것들

- 1) 서블릿 작성
- 2) 파라미터 처리 request.getParameter()
- 3) forwarding

Chapter 02. 자세히 알아보기

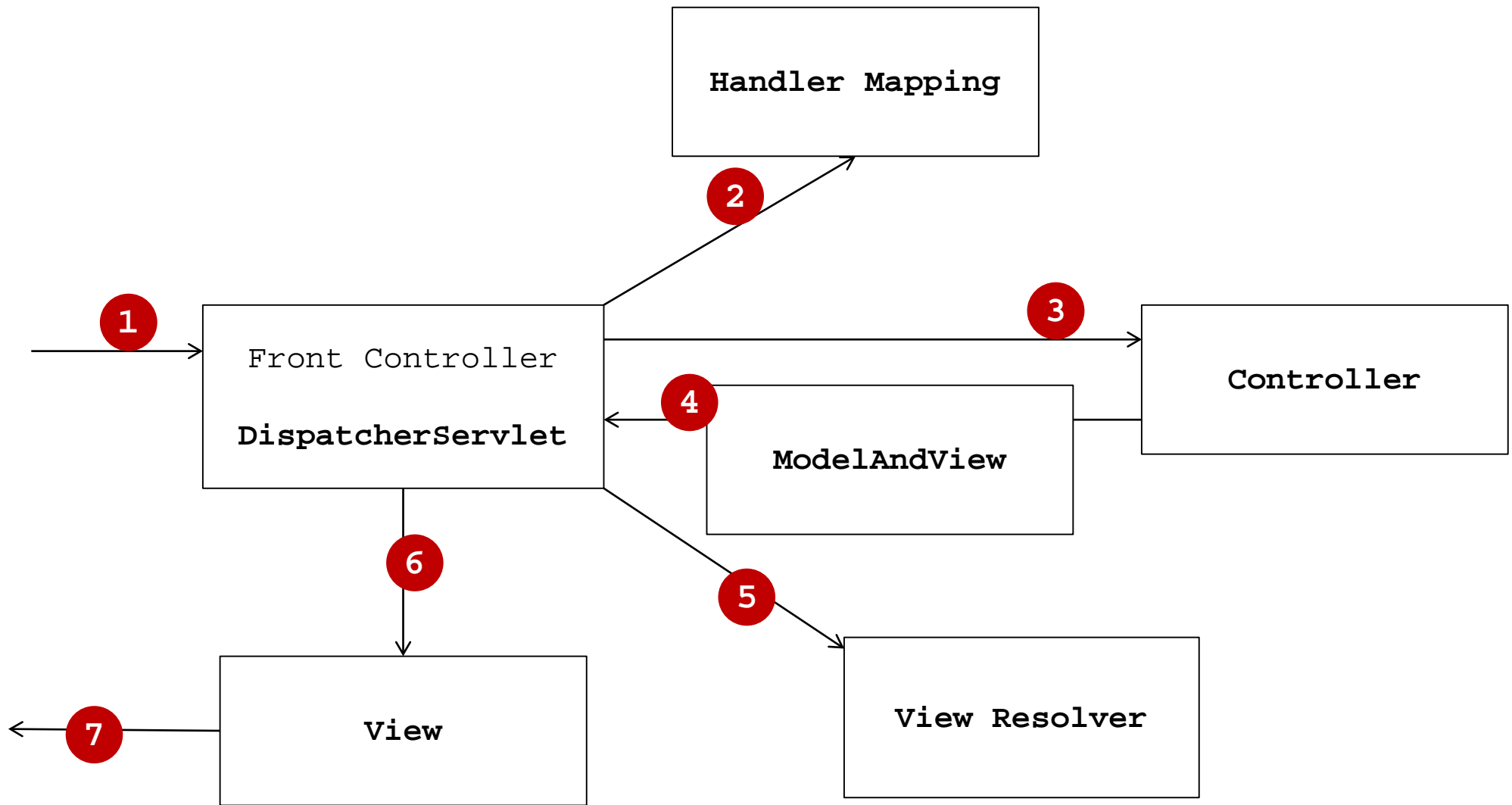
01.DispatcherServlet 과 MVC

02.주요 @ 사용

03.애플리케이션 컨텍스트 생성과정

04.애플리케이션 아키텍처

1.1 DispatcherServlet 과 MVC



1.1 DispatcherServlet 과 MVC

1. 사용자의 요청을 DispatcherServlet이 받는 다.
2. 요청을 처리해야 하는 컨트롤을 찾기 위해 **HandlerMapping**에게 질의를 하고 **HandlerMapping**은 컨트롤 객체에 매핑되어 있는 URL를 찾아낸다.
3. **DispatcherServlet**은 찾은 컨트롤에게 요청을 전달하고 Controller는 서비스 계층의 인터페이스를 호출하여 적절한 비즈니스를 수행한다.
4. 컨트롤러는 비즈니스 로직의 수행결과로 받아낸 도메인 모델 객체와 함께 뷰이름을 **ModelAndView** 객체에 저장하여 반환한다.
5. **DispatcherServlet**은 응답할 **View**를 찾기 위해 **ViewResolver**에게 질의를 한다.
6. **DispatcherServlet**은 찾아낸 **View** 객체에게 요청을 전달한다.

Chapter 02. 자세히 알아보기

01. DispatcherServlet 과 MVC

02. 주요 @ 사용

03. 애플리케이션 컨텍스트 생성과정

04. 애플리케이션 아키텍처

2.1 @RequestMapping – 핸들러 매핑

2.1.1 메소드 단독 매핑

```
public class UserController {  
  
    @RequestMapping( "/hello" )  
    public String hello( .... ) { }  
  
    @RequestMapping( "/main" )  
    public String main( .. ) { }  
}
```

2.1 @RequestMapping – 핸들러 매핑

2.1.2 타입 + 메소드 매핑

```
@RequestMapping( "/user" )
public class UserController {

    @RequestMapping( "/add" )
    public String add( .... ) { }

    @RequestMapping( "/delete" )
    public String edelete( ... ) { }
}

@RequestMapping( "/user/add" )
public class UserController {

    @RequestMapping( method = RequestMethod.GET )
    public String form( .... ) { }

    @RequestMapping( method = RequestMethod.POST )
    public String submit( ... ) { }
}
```

2.1 @RequestMapping – 핸들러 매핑

2.1.3 타입 단독 매핑

```
@RequestMapping( "/user/*" )
public class UserController {
    @RequestMapping
    public String add( .... ) {

    }

    @RequestMapping
    public String edit( ... ) {

    }
}
```

/user/add, /user/edit 으로 접근

2.2 @RequestParam – 파라미터 매핑

2.2.1 기본 사용법

- http 요청 파라미터를 메소드 파라미터에 넣어주는 어노테이션

```
public String view( @RequestParam("id") int id, @RequestParam("name") String name ) {  
  
    . . .  
}
```

- RequestParam를 사용했다면 해당 파라미터가 반드시 있어야 한다. 없으면 HTTP 400 – Bad Request 를 받는다.
- 보통 다음과 같이 정보를 더 추가해서 파라미터를 매핑한다.

```
public String view( @RequestParam( value="id", required=false, defaultValue="-1") int id ) {  
  
    . . .  
}
```

2.3.1 사용법

- URL에 쿼리 스트링 대신 URL 패스로 풀어 쓰는 방식

예) /board/view?no=10 -> /board/view/10

```
@RequestMapping( "/board/view/{no}" )  
public String view( @PathVariable("no") int no ) {  
  
    . . .  
  
}
```

2.4 @ModelAttribute

2.4.1 사용법

- 요청 파라미터를 객체에 담을 때 사용

```
public class UserVo{
    long no;
    String name;
    String password;

    . . .

    . . .
}

@RequestMapping( value="/user/join", method=RequestMethod.POST )
public String join( @ModelAttribute UserVo userVo ) {

    userService.join( userVo );

    ....
}
```


2.5 핸들러 메소드의 파라미터

2.5.1 다양한 파라미터

- HttpServletRequest, HttpServletResponse
- HttpSession
- Writer

2.5.2 Model 타입 파라미터

- 모델정보를 담을 수 있는 오브젝트가 전달.

```
public String hello( ModelMap model ) {  
  
    User user = new User( 1, "Spring" );  
    model.addAttribute( "user", user );  
    . . .  
    . . .  
}
```

Chapter 02. 자세히 알아보기

01. DispatcherServlet 과 MVC

02. 주요 @ 사용

03. 애플리케이션 컨텍스트 생성과정

04. 애플리케이션 아키텍처

3.1 웹 애플리케이션 컨텍스트 [Web Application Context]

1. web.xml 서블릿 매핑 설정의 <servlet-name>에 '-servlet.xml' 를 붙힌 이름의 파일을 WEB-INF에서 찾아 컨테이너에 Bean을 생성하고 초기화 한다.

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

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

3.1 웹 애플리케이션 컨텍스트 [Web Application Context]

2. <servlet-name> '-servlet.xml' 설정파일

```
<context:annotation-config />
```

```
<context:component-scan base-package="com.example.controller"/>
```

- Controller 빈을 등록하고 빈의 이름(URL)로 핸들러가 매핑된다.
- @MVC 기반에서 빈의 생성은 어노테이션 기반의 컴포넌트 스캐닝을 통해 생성되고 메서드가 핸들러 매핑과 어댑터의 대상이 된다.

3.1 웹 애플리케이션 컨텍스트 [Web Application Context]

2. <servlet-name> '-servlet.xml' 설정파일

```
<bean id="urlMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping" >
  <property name="mappings">
    <props>
      <prop key="/member">memberController</prop>
    </props>
  </property>
</bean>
```

```
<bean id=" memberController" class= "com.example.controller.MemberController">
```

- 핸들러 어댑터의 대상이 객체이고 객체의 `handleRequest(HttpServletRequest req, HttpServletResponse resp)` 메소드 하나만이 url대상이 된다.

3.2 루트 애플리케이션 컨텍스트 (Root Application Context)

1. 리스너를 등록해 두면, 루트 컨텍스트가 생성되게 되며, 설정 파일은 디폴트로 /WEB-INF/applicationContext.xml 이다.

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

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

2. 서비스 계층, 데이터 액세스 계층을 포함해서 웹 환경과 직접 관련이 없는 모든 빈은 여기에 등록한다.

3.2 루트 애플리케이션 컨텍스트 (Root Application Context)

3. applicationContext.xml 예

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd
http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context.xsd
http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee.xsd
http://www.springframework.org/schema/lang http://www.springframework.org/schema/lang/spring-lang.xsd
http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd
http://www.springframework.org/schema/task http://www.springframework.org/schema/task/spring-task.xsd">

    <context:annotation-config />

    <context:component-scan base-package="com.example.springex">
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Repository" />
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Service" />
        <context:include-filter type="annotation"
            expression="org.springframework.stereotype.Component" />
    </context:component-scan>

</beans>
```

3.1 웹 애플리케이션 컨텍스트 [Web Application Context]

[실습] emailist3

emailist 웹 애플리케이션을 Spring @MVC를 적용해 봅니다.

한글처리를 위해서 다음 필터 설정을 사용합니다.

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

  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>

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


3.1 웹 애플리케이션 컨텍스트 [Web Application Context]

[실습] emailist3

emailist 웹 애플리케이션을 Spring @MVC를 적용해 봅니다.

컨트롤에서 redirect 응답을 위해서는 다음 코드를 참고 합니다.

```
@RequestMapping( "/user/join", method=RequestMethod.POST )

public String join( @ModelAttribute UserVo userVo ) {
    userService.join( userVo );

    return "redirect:/user/joinsuccess";
}
```

Maven POM에 다음 mysql jdbc driver dependency를 추가 합니다.

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.38</version>
</dependency>
```

3.1 웹 애플리케이션 컨텍스트 [Web Application Context]

[실습] emailist3

Maven POM에 다음 jstl dependency를 추가 합니다.

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

Chapter 02. 자세히 알아보기

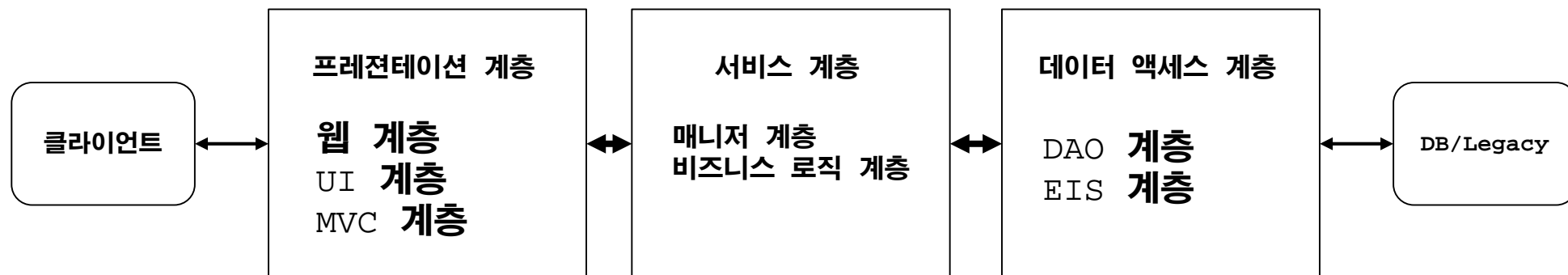
01.DispatcherServlet 과 MVC

02.주요 @ 사용

03.애플리케이션 컨텍스트 생성과정

04.애플리케이션 아키텍처

4.1 애플리케이션 아키텍처



스프링에서는...

1. 3계층은 스프링을 사용하는 엔터프라이즈 애플리케이션에서 가장 많이 사용되는 구조
2. 스프링 주요 모듈과 기술을 보면 3계층 구조에 맞게 설계
3. 논리적 개념이므로 언제든지 상황과 조건에 따라 달라 질 수 있다.

4.2 애플리케이션 아키텍처 예제

[예제] mysite

1. 비즈니스 분석 (사용자 스토리 도출)

- (1) 사용자는 회원가입을 한다.
- (2) 사용자는 로그인을 한다.
- (3) 사용자는 로그아웃을 한다.

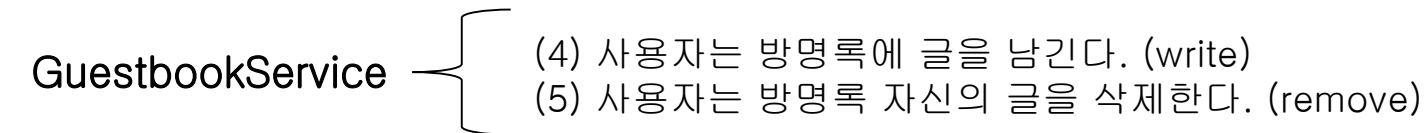
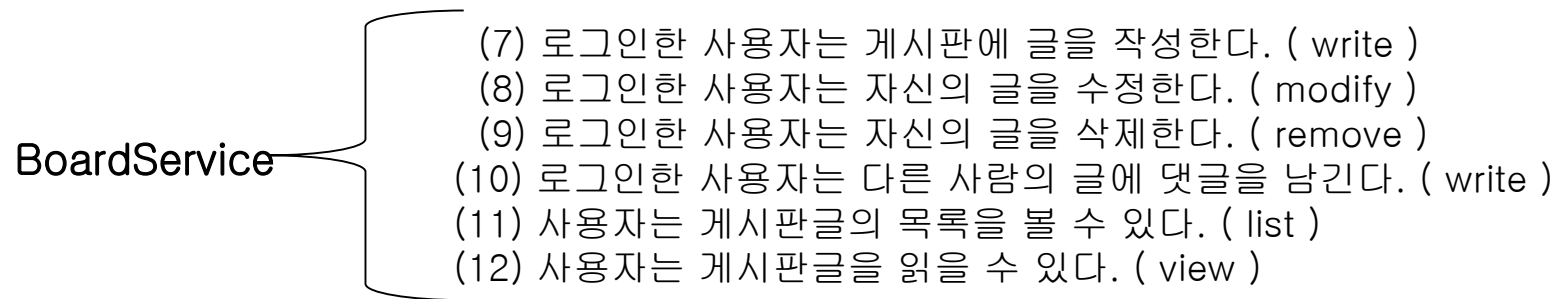
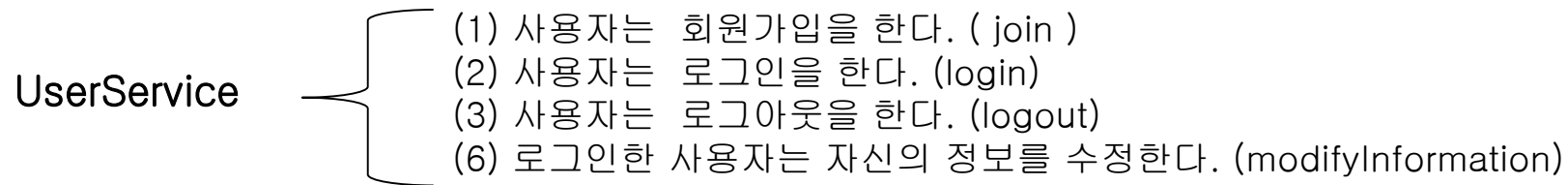
- (4) 사용자는 방명록에 글을 남긴다.
- (5) 사용자는 방명록 자신의 글을 삭제한다.

- (6) 로그인한 사용자는 자신의 정보를 수정한다.
- (7) 로그인한 사용자는 게시판에 글을 작성한다.
- (8) 로그인한 사용자는 자신의 글을 수정한다.
- (9) 로그인한 사용자는 자신의 글을 삭제한다.
- (10) 로그인한 사용자는 다른 사람의 글에 댓글을 남긴다.
- (11) 사용자는 게시판글의 목록을 볼 수 있다.
- (12) 사용자는 게시판글을 읽을 수 있다.

4.2 애플리케이션 아키텍처 예제

[예제] mysite

1. 서비스 정의



4.2 애플리케이션 아키텍처 예제

[예제] mysite

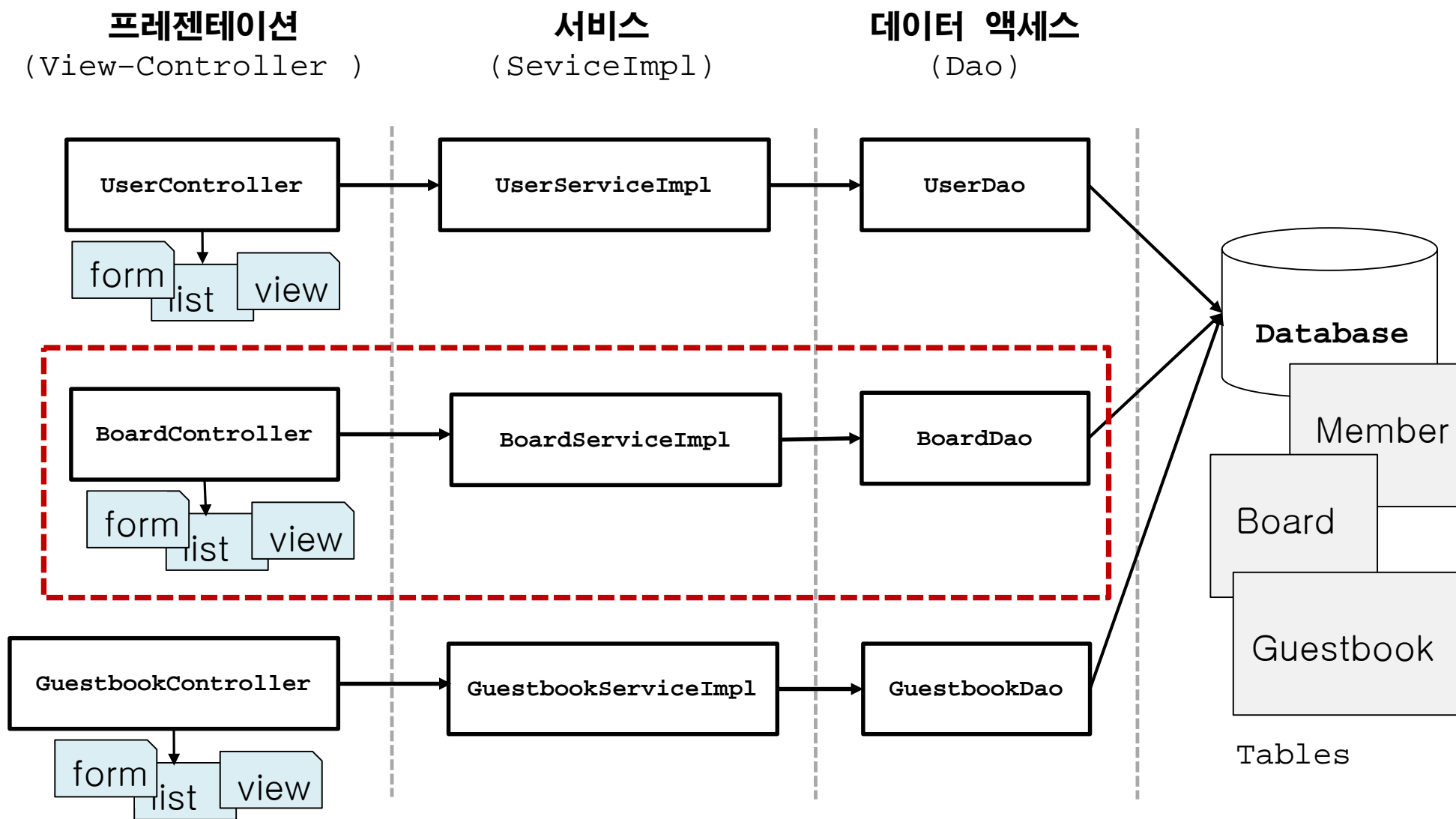
1. 서비스 인터페이스 정의 (애플리케이션 기능 목록)

```
Interface UserService {  
    void join();  
    void login();  
    void logout();  
    void modifyInfo();  
}
```

```
Interface GuestbookService {  
    void write();  
    void remove();  
}
```

```
Interface BoardService {  
    void write();  
    void remove();  
    void modify();  
    void list();  
    void view  
}
```

4.2 애플리케이션 아키텍처 예제



Chapter 03. 자세히 알아보기2

01.DefaultServlet Handler 등록

02.ViewResolver 설정

03.예외처리

1.1 DefaultServlet 위임

1.1.1 정적 자원 접근 실패

- html, css, js 등의 파일 접근에 실패한다.
- DispatcherServlet 이 모든 URL처리에 서블릿 매핑을 하였기 때문에 톰캣은 정적 자원에 대한 URL처리도 DispatcherServlet에게 넘기기 때문이다. (즉, DefaultServlet에 위임을 하지 못한다.)
- 실습으로 알아 보자.

1.1.2 Spring MVC 에서 DefaultServlet 위임 처리하기

- HandlerMapping이 URL과 컨트롤러의 메소드(핸들러) 와의 매핑 정보를 가지고 있다.
- HandlerMapping에서 정적 자원에 대한 URL은 DefaultServlet으로 위임할 수 있도록 설정 해주어야 한다.
- spring-servlet.xml 파일에서...

```
<!-- validator, conversionService, messageConverter를 자동으로 등록 -->  
<mvc:annotation-driven />
```

```
<!-- 서블릿 컨테이너의 디폴트 서블릿 위임 핸들러 ->  
<mvc:default-servlet-handler/>
```

Chapter 03. 자세히 알아보기2

01.DefaultServlet Handler 등록

02.ViewResolver 설정

03.예외처리

2.1 ViewResolver란?

1. ViewResolver는 HandlerMapping이 컨트롤러를 찾아주는 것 처럼, View 이름을 가지고 View 오브젝트를 찾아준다.
2. ViewResolver 를 빈 등록하지 않으면 DispatcherServlet의 기본 ViewResolver 인 InternalResourceViewResolver가 사용된다.
3. 디폴트 사용에서는 View 로 이동하는 전체 경로를 다 적어 주어야 한다.
4. prefix와 suffix를 지정하여 앞 뒤의 내용을 생략하여 매우 편리하게 View를 지정할 수 있다.

2.2 ViewResolver 설정

<!-- ViewResolver 설정 ->

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView" />
    <property name="prefix" value="/WEB-INF/views/" />
    <property name="suffix" value=".jsp" />
    <property name="order" value="1" />
</bean>
```

1. JSTL 라이브러리가 클래스패스에 존재하면 JstlView를 사용하고 아니면 InternalResourceView를 사용하면 된다.
2. 여러 ViewResolver를 등록 될 수 있는 데, order 프로퍼티를 지정해 ViewResolver 의 우선순위를 지정할 수 있다.
3. URL에 일정한 이름을 주면 View와 자동으로 연결 시킬 수 있다.

Chapter 03. 자세히 알아보기2

01.DefaultServlet Handler 등록

02.ViewResolver 설정

03.예외처리

3.1 예외 처리

3.1.1 예외 블랙홀

```
try {  
  
    ...  
  
} catch( Exception ex ) {  
    // 여기를 비워두는 것은 가장 나쁜 예외 처리이다.  
}
```

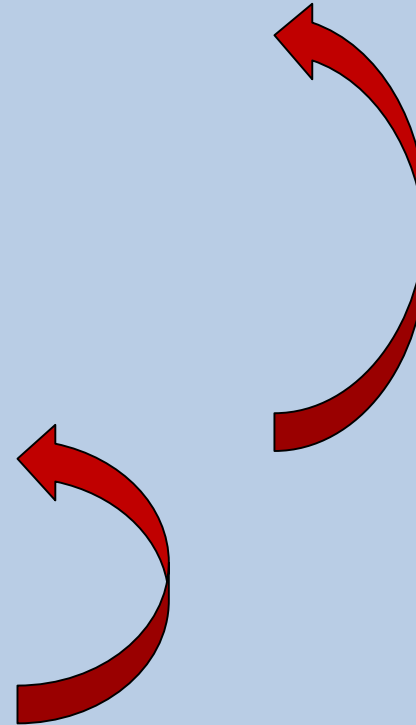
3.1 예외 처리

3.1.2 무의미하고 무책임한 throws (예외처리 회피)

```
public void method() throws Exception {  
    method2();  
}
```

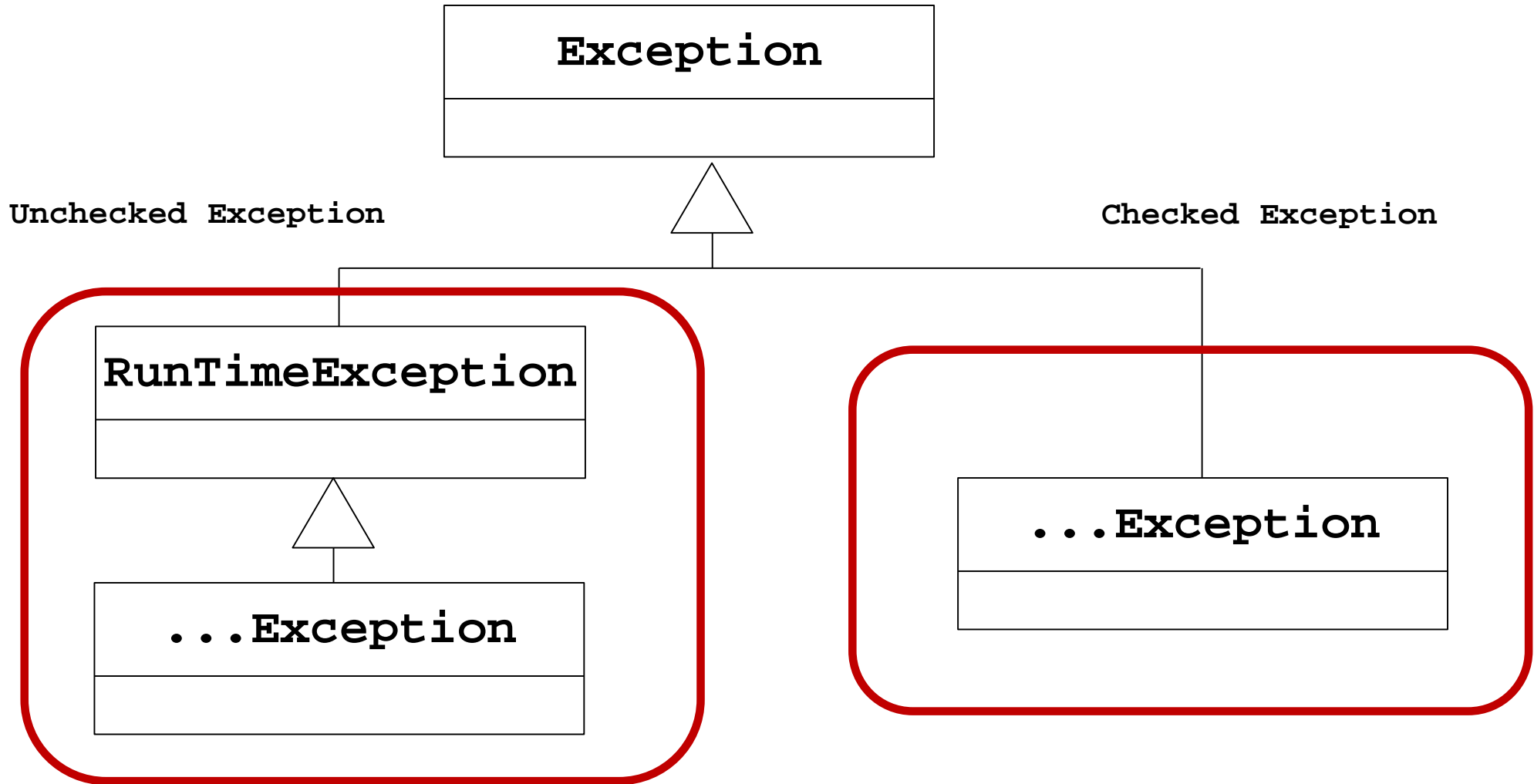
```
public void method2() throws Exception {  
    method3();  
}
```

```
public void method3() throws Exception {  
    method4();  
}
```



3.1 예외 처리

3.1.3 체크 예외 처리 [상투적인 try~catch]



3.2 예외 처리 방법

3.2.1 예외 복구

- 예외상황을 파악하고 문제를 해결해서 정상 상태로 돌려 놓는 것.
- 예외를 어떤 식으로 복구 가능성이 있을 때 예외 처리를 강제하는 체크 예외를 사용할 수 있다.
- 예외는 복구가 가능한가?

3.2.2 예외 처리 회피

- throws 문을 선언하여 예외가 발생하면 외부로 던지게 한다.
- 또는, catch로 예외를 잡아 로그를 남기고 다시 예외를 던지는 방법
- DAO가 SQLException를 외부로 던지면 서비스, 컨트롤은 처리 가능한가?

3.3 Spring 에서의 예외 처리 방법

3.3.1 예외 전환

- 대부분의 예외는 복구해서 정상적인 상태로 만들 수 없기 때문에 예외를 메소드 밖으로 던진다.
- 예외 처리 회피처럼 그대로 넘기지 않고 적절한 예외로 전환한다.
- 로우 레벨의 예외 상황에 대한 적합한 의미를 가진 예외로 변경하는 중요하다.

```
catch( SQLException ex ) {  
  
    throw new UserDaoException( ex );  
  
}
```

3.3 Spring 에서의 예외 처리 방법

3.3.2 런타임 예외의 보편화

- 예외는 미리 파악되어야 하고 예외가 발생하지 않는 것이 가장 좋다.
- 빨리 예외 발생 작업을 중지하고 서버 관리자나 개발자에게 통보해주고 예외 내용을 로그로 남겨야 한다.
- 자바 엔터프라이즈 서버 환경에서의 체크 예외의 활용성은 Spring에서 다시 고려
- 복구할 수 없는 예외는 언체크 예외로 만든다.
- 언체크 예외도 필요 시 catch가 가능하다.
- RuntimeException 를 사용해서 전환, 포장 해서 사용하도록 한다.

3.3 Spring 에서의 예외 처리 방법

3.3.3 Controller 에서의 예외 처리

```
@ExceptionHandler( UserDaoException.class )  
public String handleUserDaoException() {  
    return "/WEB-INF/views/error/exception.jsp";  
}
```

- @ExceptionHandler 를 사용해서 Exception 과 핸들러를 매핑 한다.
- Controller의 개별 핸들러 메소드에서 예외를 매핑하는 것보다 컨트롤러 어드바이스를 사용해서 애플리케이션의 같은 종류의 예외를 처리하는 것이 효과적이다.

3.3 Spring 에서의 예외 처리 방법

3.3.4 어드바이징 컨트롤러에서의 예외 처리 (spring 3.2 부터 지원)

```
@ControllerAdvice
public class ApplicationExceptionHandler {

    @ExceptionHandler( UserDaoException.class )
    public String handleDaoException( Exception e ) {
        return "/WEB-INF/views/error/exception.jsp";
    }
}
```

- @ControllerAdvice 는 @Component 를 상속한 어노테이션 이기 때문에 컴포넌트 스캐닝 을 통해 선택된다.
- 실용적인 방법은 @ExceptionHandler를 사용해서 하나의 예외에 하나의 예외 핸들러를 묶는 방식이다.
- 핸들러에 @ResponseStatus 를 사용하여 클라이언트의 응답 코드를 지정할 수 있다.

3.3 Spring 에서의 예외 처리 방법

3.3.4 어드바이징 컨트롤러에서의 예외 처리 (spring 3.2 부터 지원)

- [실습과제] 다음 코드를 참고해서 예외처리를 해보자.

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler( Exception.class )
    public ModelAndView handlerException(
        HttpServletRequest request,
        Exception e){

        // 1. 로깅
        e.printStackTrace();

        // 2. 시스템 오류 안내 화면
        ModelAndView mav = new ModelAndView();
        mav.setViewName( "error/exception" );

        return mav;
    }
}
```

3.3 Spring 에서의 예외 처리 방법

3.3.5 애플리케이션 밖에서 발생하는 오류 안내 페이지

- web.xml 페이지에 다음과 같은 설정을 한다.

```
<!-- 공통 에러 페이지 -->
<error-page>
    <error-code>404</error-code>
    <location>/WEB-INF/views/error/404.jsp</location>
</error-page>

<error-page>
    <error-code>500</error-code>
    <location>/WEB-INF/views/error/500.jsp</location>
</error-page>
```


Spring @MVC

Data Access

1. datasource (Connection Pool)
2. DAO Pattern
3. 예외처리
4. 템플릿과 API
5. Spring에서 MyBatis 사용하기

1. datasource

- JDBC를 통해 DB를 사용하려면, Connection 타입의 DB 연결 객체가 필요하다.
- 엔터프라이즈 환경에서는 각 요청마다 Connection을 새롭게 만들고 종료시킨다.
- 애플리케이션과 DB사이의 실제 커넥션을 매번 새롭게 만드는 것은 비효율적이고 성능저하
- 풀링(pooling) 기법 사용
정해진 개수의 DB Connection Pool에 준비하고 애플리케이션 요청때 마다 꺼내서 할당하고 돌려받아 pool에 저장.
- Spring에서는 DataSource를 하나의 독립된 빈으로 등록하도록 강력하게 권장.
- 엔터프라이즈 시스템에서는 반드시 DB 연결 풀 기능을 지원하는 DataSource를 사용해야 한다.

1. datasource

- 종류

스프링에서 제공 :

SimpleDriverDataSource, SingleConnectionDataSource

학습/테스트용. 실제 서비스에서 사용하지 말 것

아파치 Common DBCP :

가장 유명한 오픈소스 DB 커넥션 풀 라이브러리

<http://commons.apache.org/dbcp>

상용 DB 커넥션 풀 :

스프링 빈으로 등록 가능하고 프로퍼티를 통해 설정이 가능하다면 어떤 것이든지 사용하면 된다.

1. datasource

[실습]

1. 라이브러리 추가 (pom.xml)

```
<!-- spring jdbc -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${org.springframework-version}</version>
</dependency>
```

2. Oracle DataSource를 bean으로 등록한다.

```
<!-- oracle datasource -->
<bean id="oracleDatasource" class="oracle.jdbc.pool.OracleDataSource" destroy-method="close">
    <property name="URL" value="jdbc:oracle:thin:@localhost:1521:xe" />
    <property name="user" value="webdb" />
    <property name="password" value="webdb" />
    <property name="connectionCachingEnabled" value="true" />
    <qualifier value="main-db" />
</bean>
```

1. datasource

[실습]

3. Common DBCP의 DataSource를 bean으로 등록한다. (MySQL인 경우)

```
<!-- Common DBCP -->
<dependency>
    <groupId>commons-dbc</groupId>
    <artifactId>commons-dbc</artifactId>
    <version>1.4</version>
</dependency>
```

```
<!-- Connection Pool DataSource-->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/webdb" />
    <property name="username" value="webdb" />
    <property name="password" value="webdb" />
</bean>
```

1. datasource

[실습]

3. 현재 Dao 객체에 datasource 빈을 주입하고 getConnection() 메서드를 대체하고 테스트 합니다.

2. DAO 패턴

- 데이터 액세스 계층은 DAO 패턴이라 불리는 방식으로 분리하는 것이 원칙.
- 비즈니스가 단순하거나 없으면, DAO를 서비스 계층과 통합할 수 있다.
- 데이터 액세스 기술을 외부로 노출시키지 않는다.
- datasource를 주입(DI)받으려면 해야 하고 메서드는 `add()`, `update()` 등, 단순하고 CRUD에 따르는 일반적인 이름을 사용하도록 한다.

3. 예외처리

- 데이터 액세스 중에 발생하는 예외는 복구할 수 없다.
- DAO 외부로 던지는 예외는 런타임 예외(RuntimeException)여야 한다.
- SQLException은 서비스계층에서 직접 다뤄야 할 이유가 없으므로 RuntimeException으로 전환해야 한다.
- 그러나 받아야 하는 경우가 있다면, 의미를 갖는 예외로 전환한다.

4. 템플릿과 API

- 데이터 액세스 기술을 사용하는 코드는 try/catch/finally 와 반복되는 코드로 작성되는 경우가 많다.
- 데이터 액세스 기술은 외부의 리소스와 연동을 통해서 이루어 지기 때문에 다양한 예외 상황이 발생할 경우가 많고 예외상황을 종료하고 리소스를 반환하기 위한 코드가 길고 복잡해 지는 경향이 있다. (가독성이 좋지 않다.)
- 스프링에서는 DI의 응용 패턴인 **템플릿/콜백 패턴**을 이용해 반복되는 판에 박힌 코드를 피하고 예외 변환과 트랜잭션 동기화를 위한 **템플릿**을 제공한다.
- 해당 기술의 데이터 액세스 기술 API(MyBatis API, JDBC API)와 스프링 데이터 액세스 템플릿을 조합하여 사용한다.

5. MyBatis3.X (소개)

1. MyBatis2.x(iBatis)의 후속으로 등장한 ORM 프레임워크이다.
2. XML를 이용한 SQL과 **ORM** 을 지원한다.
3. **본격적인 ORM인 JPA나 Hibernate** 처럼 새로운 DB 프로그래밍 패러다임을 이해해야 하는 것은 아니다. (MyBatis3.x에서는 Mapper 인터페이스를 통해 지원)
3. 이미 익숙한 SQL를 그대로 사용하고 JDBC코드의 불편함을 제거
4. 가장 큰 특징은 SQL을 자바코드에서 분리해서 별도의 XML 파일 안에 작성하고 관리할 수 있는 것이다.
5. 스프링 3.0부터는 MyBatis3.x(iBatis2.x) 버전에 스프링 데이터 액세스 기술 대부분을 지원한다.(DataSource Bean 사용, 스프링 트랜잭션, 예외 자동변환, 템플릿)

5. MyBatis3.X (**스프링에서 사용하기**)

1. MyBatis의 DAO는 `SQLSession` 인터페이스를 구현한 클래스의 객체를 DI받아 사용한다.
2. MyBatis의 DAO는 `SQLSessionDaoSupport` 추상 클래스를 상속받아 구현하기도 한다.
3. 그리고 `Mapper` 인터페이스를 통한 OR 매핑 기능을 지원한다.
4. 이 중에 `SQLSession` 인터페이스를 구현한 클래스의 객체의 DI 방식을 주로 사용하게 된다.
`SQLSession` 인터페이스를 구현한 `SQLSessionTemplate` 클래스를 사용한다.

5. MyBatis3.X (**설정하기** – 라이브러리 추가)

1. 라이브러리 추가(pom.xml)

```
<!-- MyBatis -->
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.2.2</version>
</dependency>

<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>1.2.0</version>
</dependency>
```

5. MyBatis3.X (설정하기 – Bean 설정)

2. SqlSessionFactoryBean 설정 (applicationContext.xml)

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

3. SqlSessionTemplate 설정 (applicationContext.xml)

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

4. DAO 에서는 SqlSessionTemplate를 DI 한다.

```
public class EmailListDao {
    @Autowired
    private SqlSession sqlSession;

    . . .
}
```

5. MyBatis3.X (설정하기 - 설정 파일과 매핑파일)

5. MyBatis 설정 파일(configuration.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>
    <typeAliases>
    </typeAliases>
    <mappers>
        <mapper resource="mybatis/mappers/emaillist.xml" />
    </mappers>
</configuration>
```

6. sql 매핑 파일(emaillist.xml)

```
<?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="emaillist">
</mapper>
```

5. MyBatis3.X (SQL 작성)

select 하기

1. 결과의 칼럼 이름과 resultMap의 Class의 필드 명이 다른 경우

```
<resultMap type="com.bit2015.emaillist3.vo.EmailListVo" id="resultMapList">
    <result column="no" property="no"/>
    <result column="first_name" property="firstName"/>
    <result column="last_name" property="lastName"/>
    <result column="email" property="email"/>
</resultMap>

<select id="list" resultMap="resultMapList">
    select no,
           first_name,
           last_name,
           email
    from email_list
    order by no desc
</select>
```

5. MyBatis3.X (SQL 작성)

select 하기

2. 결과의 칼럼 이름과 resultType의 Class의 필드 명이 다른 경우(resultMap를 사용안함)

```
<select id="list2" resultType="com.bit2015.emaillist3.vo.EmailListVo">
    select no,
           first_name as firstName,
           last_name as lastName,
           email
    from email_list
    order by no desc
</select>
```

3. resultType, parameterType의 이름은 configuration.xml alias를 사용해 짧게 줄인다.

```
<typeAliases>
    <typeAlias alias="EmailListVo" type="com.bit2015.emaillist3.vo.EmailListVo" />
</typeAliases>
```


5. MyBatis3.X (SQL 작성)

select 하기

resultType 의 클래스가 존재하지 않을 경우 map를 사용한다.

```
<select id="joinlist" resultType="map">
  select  a.no, a.title, a.reg_date, b.no, b.name, b.email
    from  board a,
         member b
   where  a.member_no = b.no
  order by a.reg_date desc
</select>
```

Map으로 리턴 되고 칼럼 이름이 대문자로 Map의 Key가 된다.

5. MyBatis3.X (SQL 작성)

파라미터 바인딩

1. 객체를 사용한 여러 파라미터 바인딩

```
<insert id="insert" parameterType="EmailListVo">
    insert
    into email_list
    values ( email_list_no_seq.nextval, #{firstName }, #{lastName }, #{email } )
</insert>
```

2. 파라미터가 하나인 바인딩

```
<select id="insert" parameterType="int">
    select no,
           name,
           message,
           to_char( reg_date, 'yyyy-MM-dd hh:mi:ss' ) as regDate
    from guestbook
    where no=#{no }
</select>
```

- **#{no }**는 임의 지정해도 상관없다.

- **int** 는 내장된 **alias** 이다. (**byte, short, long, int, integer, double, float, boolean, string**)

5. MyBatis3.X (SQL 작성)

파라미터 바인딩

파라미터 클래스가 존재하지 않고 여러 값을 파라미터로 넘겨야 하는 경우

```
<select id="getId" parameterType="map" resultType="string">
    select id from member where name=#name# and ssn=#ssn#
</select>
```

```
jMap map = new HashMap();
map.put("name", "홍길동");
map.put("ssn", "1234561234567");
```

```
String id2 = (String)sqlSession.selectOne( "getId", map );
```

5. MyBatis3.X (SQL 작성)

Insert 후, 새로들어 간 row의 Primary Key를 받아야 하는 경우

```
<insert id="insert" parameterType="GuestbookVo">
    <selectKey keyProperty="no" resultType="long" order="BEFORE">
        select guestbook_seq.nextval from dual
    </selectKey>
    <![CDATA[
insert
    into guestbook
values ( #{no }, #{name }, #{password }, #{message }, SYSDATE )
]]>
</insert>
```