

# Spring MVC

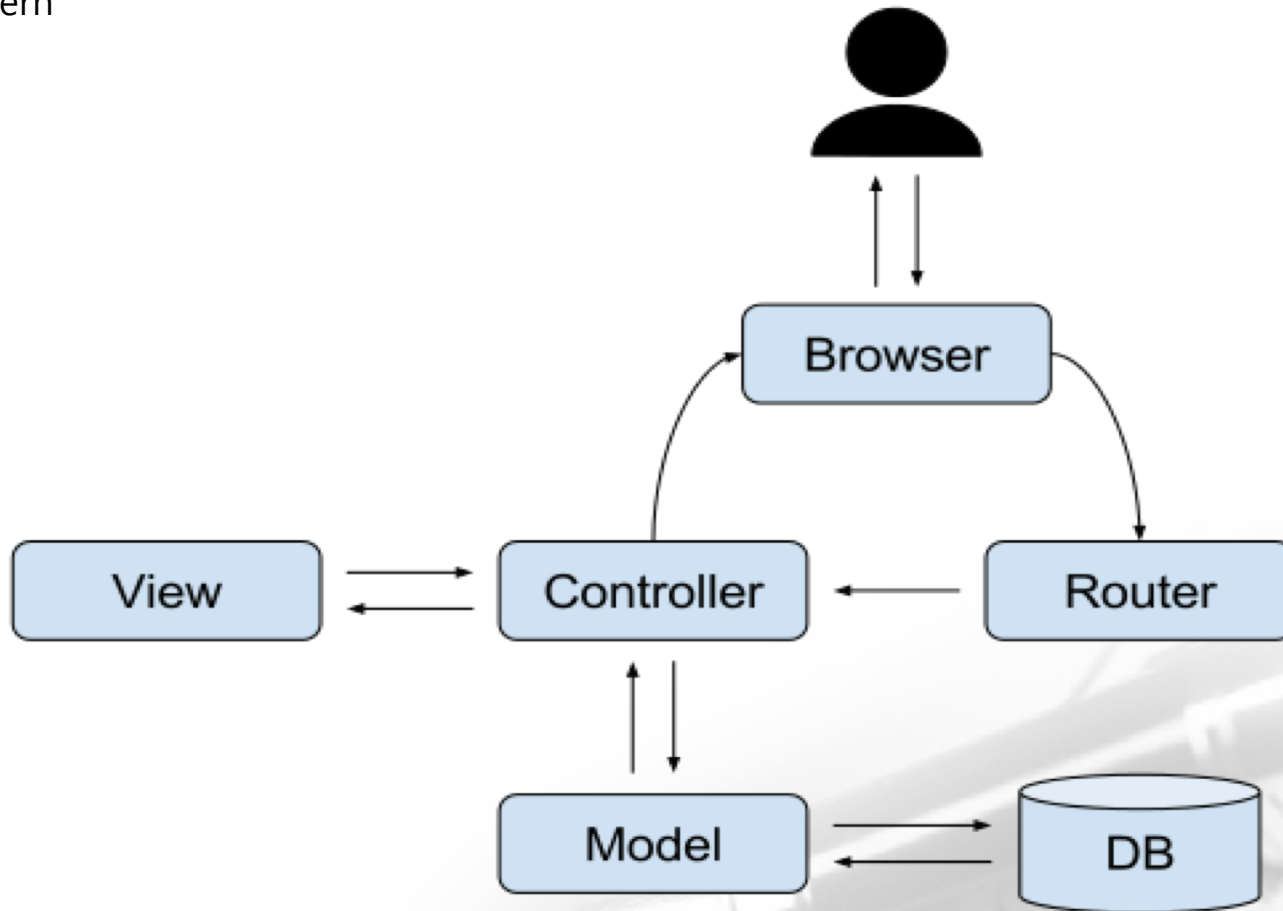
# MVC Pattern

## ❖ MVC Pattern

- ✓ Presentation Logic(View) 과 Business Logic(Processing) 을 분리해서 재사용이 가능하도록 하는 구조
- ✓ 구조
  - Model: 데이터 자체 또는 데이터를 처리하는 영역
  - View: 결과 화면을 만들어내는 영역
  - Controller: 웹의 요청을 처리하는 비즈니스 로직과 출력을 담당하는 뷰 사이의 중간 통신 역할
- ✓ 장점
  - 작업 영역을 분리할 수 있음
  - 뷰의 교체나 변경 같은 유지보수 편리
- ✓ 단점: Model 1 Pattern에 비해 복잡

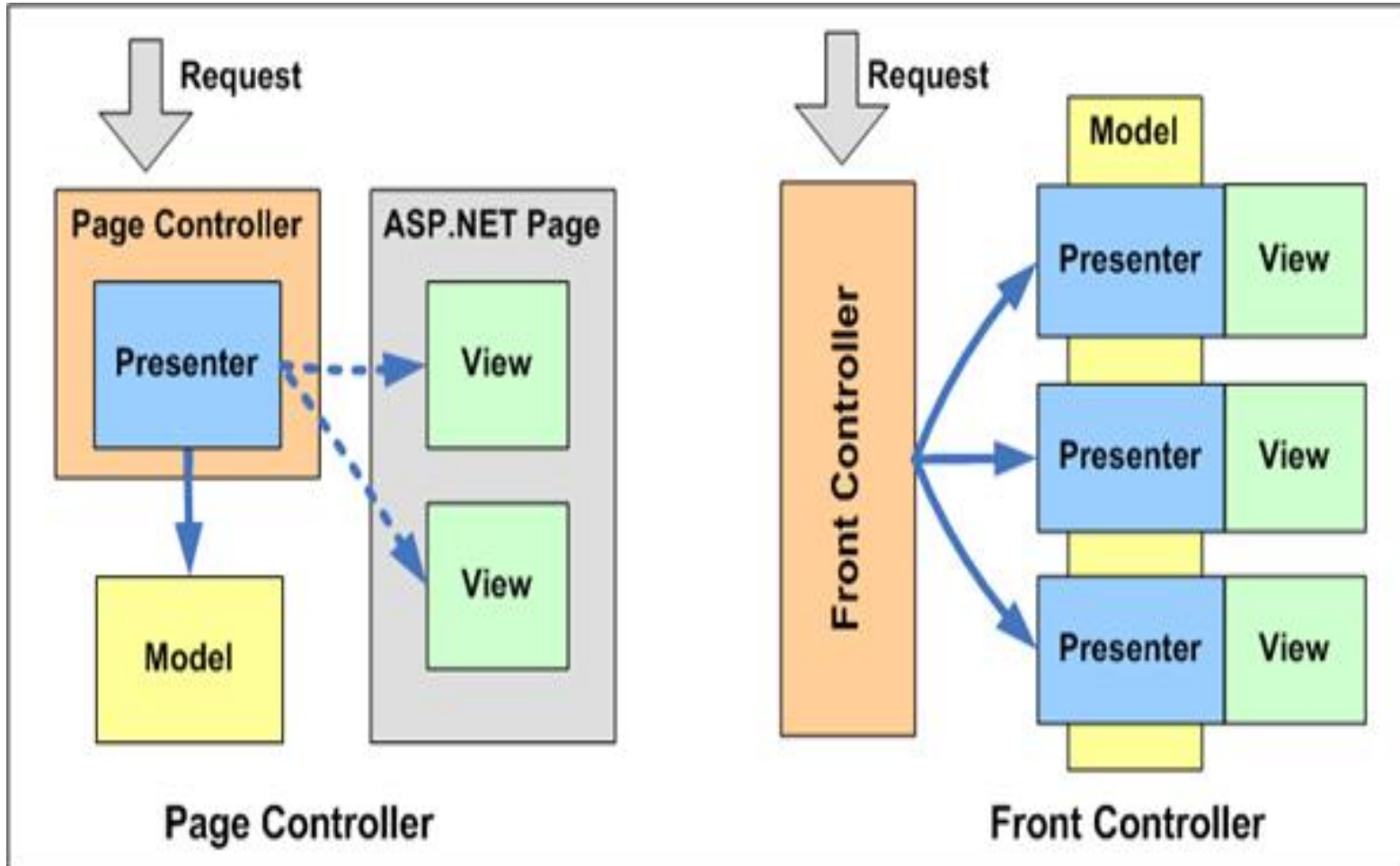
# MVC Pattern

## ❖ MVC Pattern



# MVC Pattern

## ❖ MVC Pattern





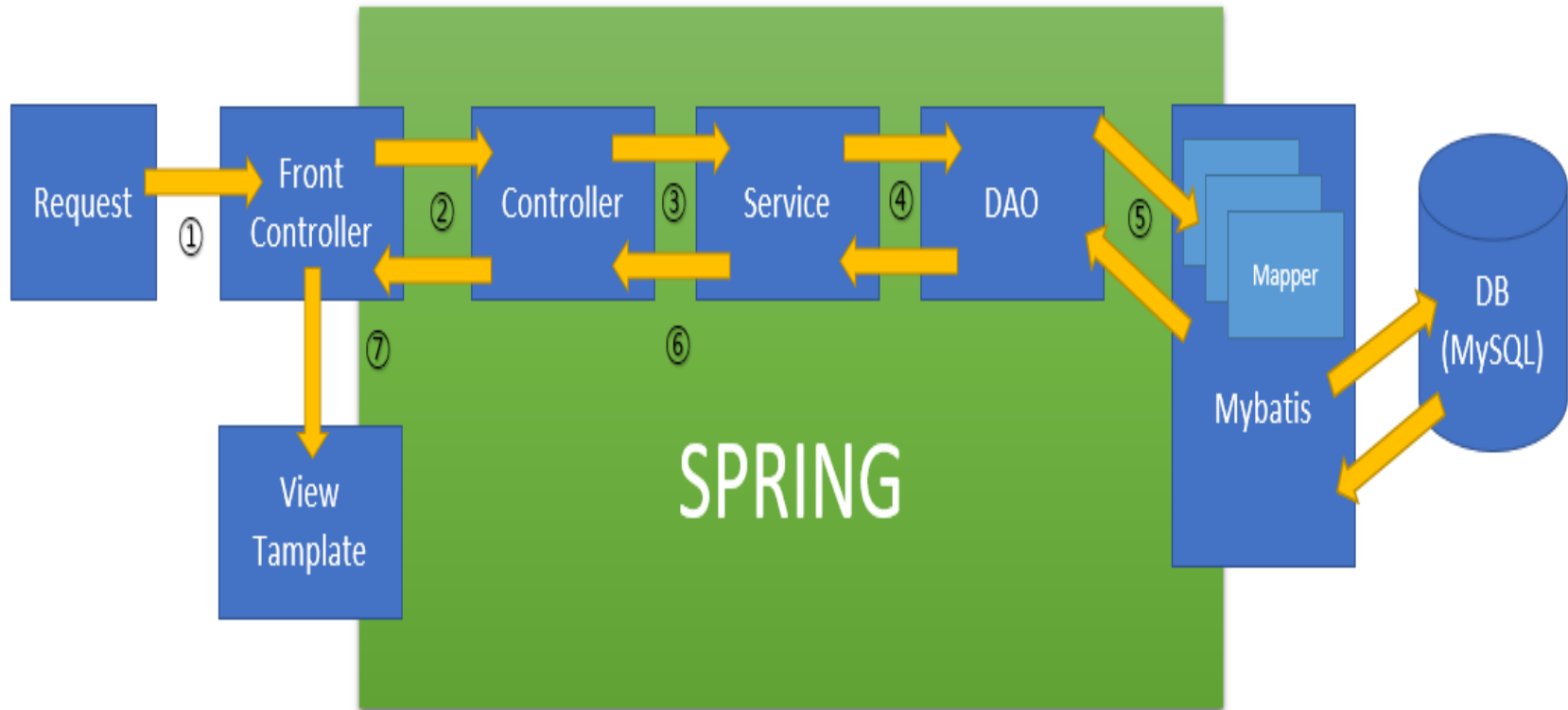
# MVC Pattern

## ❖ FrontController

- ✓ 모든 요청을 처리해주는 하나의 Controller(Controller)를 두는 패턴
- ✓ MVC(Model-View-Controller) Pattern과 함께 사용하는 패턴
- ✓ Front-Controller Pattern 장점
  - 추적(Tracking)이나 보안(Security)를 적용할 때 하나의 Controller(Controller)에서 가능하기 때문에 편리
  - 파일 구조가 바뀌어도 URL을 유지할 수 있음
- ✓ Spring에서는 DispatcherServlet이 Front-Controller의 역할을 수행하고 @Controller를 이용해서 만들어진 Controller들이 Page-Controller 의 역할을 수행

# MVC Pattern

❖ FrontController



# Spring MVC

## ❖ Spring MVC

### ✓ 처리 과정

- 사용자의 요청을 Front Controller에게 전달
- 전달된 요청은 적절한 Page Controller를 찾아 호출
- Page Controller는 클라이언트의 요청을 처리할 Service 객체를 찾아 호출
- Service객체는 DAO객체를 이용하여 원하는 Data 처리 작업을 요청
- DAO객체는 데이터베이스 연동 로직을 이용하여 작업을 수행(CRUD) – Java JDBC, Spring JDBC, MyBatis, Hibernate 등
- Service를 통해 처리한 데이터를 Controller에게 전달
- Controller는 다시 스프링 MVC쪽으로 데이터를 전달

# Spring MVC

- ❖ Spring MVC
  - ✓ 역할 분담

| Spring MVC가 처리하는 작업               | 개발자가 직접 해야 하는 작업                |
|-----------------------------------|---------------------------------|
| URI를 분석해서 적절한 Controller를 찾는 작업   | 특정URI에 동작하는 Controller를 설계하는 작업 |
| Controller에 필요한 메서드를 호출하는 작업      | Service / DAO 객체 생성             |
| Controller의 결과 데이터를 View로 전달하는 작업 | Controller 내에 메서드 설계            |
| 적절한 View를 찾는 작업                   | View에서 전달받은 데이터의 출력             |

# Spring MVC

## ❖ Spring MVC

### ✓ 자주 사용되는 Annotation

| annotation        | 기능                                   |
|-------------------|--------------------------------------|
| @Controller       | Controller 객체임을 명시                   |
| @Repository       | 저장소 관련 객체임을 명시                       |
| @Service          | Service 객체임을 명시                      |
| @RequestMapping   | 특정 URI에 매칭되는 클래스나, 메서드임을 명시          |
| @RequestParam     | 요청에서 특정한 Parameter 값을 찾아낼 때 사용       |
| @RequestHeader    | 요청에서 특정 HTTP 헤더 정보를 추출할 때 사용         |
| @PathVariable     | 현재의 URI에서 원하는 정보를 추출할 때 사용           |
| @CookieValue      | 쿠키가 존재하는 경우 쿠키의 이름을 이용해서 쿠키 값을 추출    |
| @ModelAttribute   | 자동으로 해당 객체를 뷰까지 전달하도록 만드는 Annotation |
| @SessionAttribute | 세션상에서 모델의 정보를 유지하고 싶은 경우 사용          |
| @InitBinder       | Parameter를 수집해서 객체로 만들 경우            |
| @ResponseBody     | 리턴 타입을 HTTP의 응답 메시지로 전송              |
| @RequestBody      | JSON 형식의 문자열 Parameter를 변환해서 전달      |

# Spring MVC

## ❖ 스프링 MVC

- ✓ org.springframework.web 패키지와 org.springframework.web.servlet 패키지에 포함된 클래스를 사용
- ✓ 스프링 MVC의 주요 구성
  - DispatcherServlet: 클라이언트의 요청을 전달받는 객체(FrontController)로 Controller에게 클라이언트의 요청을 전달하고 Controller가 리턴 한 결과값을 View에 전달하여 알맞은 응답을 생성해 주도록 하는 클래스
  - HandlerMapping: 웹의 요청 URL과 Controller 클래스를 매핑 시켜주는 클래스
  - Controller: 클라이언트의 요청을 받아서 비즈니스 로직을 호출하고 결과를 DispatcherServlet에 알려주는 클래스 – Page Controller
  - ModelAndView: Controller가 처리한 결과 정보를 저장하고 뷰 선택에 대한 정보 저장
  - ViewResolver: Controller 클래스가 처리한 결과를 출력할 뷰를 결정하는 클래스
  - View: 처리 결과 화면

# Spring MVC

## ❖ Spring MVC 처리 흐름

- ✓ 클라이언트의 요청이 FrontController 클래스의 역할을 수행하는 DispatcherServlet에 전달되서 HandlerMapping을 사용하여 클라이언트의 요청을 처리할 Controller 클래스 객체를 선택
- ✓ Controller 객체는 비즈니스 로직을 호출해서 처리하고 요청 처리 결과 정보를 ModelAndView 객체를 이용해서 리턴
- ✓ DispatcherServlet은 넘겨받은 ModelAndView 객체를 이용해서 ViewResolver로부터 응답 결과를 출력할 뷰 객체를 선택
- ✓ View 객체는 ModelAndView 객체로 전달된 데이터를 출력

# Spring MVC

## ❖ DispatcherServlet 설정 - web.xml 파일에 설정

### ✓ Spring Bean Configuration 파일을 이용하는 경우의 web.xml 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  version="2.5">
  <servlet>
    <servlet-name>Servlet 이름</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Servlet 이름</servlet-name>
    <url-pattern>경로패턴</url-pattern>
  </servlet-mapping>
</web-app>
```

- 위처럼 설정하게 되면 url 패턴에 해당하는 요청이 오면 WEB-INF에 있는 [Servlet이름]-servlet.xml 파일을 FrontController 설정 파일(Spring Bean Configuration 파일)로 사용해서 처리



# Spring MVC

## ❖ DispatcherServlet 설정 - web.xml 파일에 설정

### ✓ Spring Configuration 클래스를 이용하는 경우의 설정

```
<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.AnnotationConfigWebApplicationC
      ontext
    </param-value>
  </init-param>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>클래스의 경로</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

- 위처럼 설정하게 되면 URL 패턴에 해당하는 요청이 오면 config.MvcConfig 클래스를 FrontController 설정 클래스로 사용

# Spring MVC

## ❖ DispatcherServlet

- ✓ web.xml 파일에 URL 설정

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

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

- do로 끝나는 요청이 오면 앞에 설정한 dispatcher Servlet이 FrontController 요청을 수행

# Spring MVC

## ❖ DispatcherServlet

- ✓ DispatcherServlet은 내부적으로 스프링 컨테이너를 생성해서 내부에 설정된 bean 태그의 객체들을 생성
- ✓ DispatcherServlet 설정은 3개의 객체를 등록해주어야 하는데 경우에 따라서는 생략 가능
  - HandlerMapping 객체
  - HandlerAdapter 객체
  - ViewResolver 객체
- ✓ DispatcherServlet 설정 파일에 mvc 네임 스페이스를 추가하고 <mvc:annotation-driven /> 태그를 추가하면 HandlerMapping과 HandlerAdapter 객체를 빈으로 등록하고 JSON이나 XML 요청/응답 처리를 위한 ConversionService 객체를 빈으로 등록 하며 설정 클래스인 경우는 @EnableWebMVC 를 클래스 상단에 기재

# Spring MVC

## ❖ Controller

- ✓ 스프링에서 Controller를 구현하는 방법은 Controller 인터페이스를 implements 하는 클래스를 생성하거나 POJO 클래스에 @Controller Annotation을 적용
- ✓ Controller 인터페이스를 implements 한 경우에는 요청을 처리하는 메서드를 재정의
- ✓ Annotation으로 만든 경우에는 @RequestMapping Annotation을 이용해서 클라이언트의 요청을 처리할 메서드를 지정하고 ModelAndView 또는 문자열 및 ModelMap 이나 Map, String 그리고 void로 리턴 타입을 지정

# Spring MVC

## ❖ Controller

- ✓ a.do 요청을 action() 메서드가 처리하도록 한 것

@Controller

public class HomeController{

@RequestMapping("/a.do")

public ModelAndView action() {

    //모델 생성

    Map<String, Object> model = new HashMap<String, Object>();

    //반환값인 ModelAndView 인스턴스 생성

    ModelAndView modelAndView = new ModelAndView();

    modelAndView.setViewName("결과를 출력할 파일");

    //modelAndView.addAllObjects(model);

    modelAndView.addObject("키이름", 데이터);

    return modelAndView;

}

# Spring MVC

## ❖ Controller

- ✓ web.xml에서 dispatcherServlet의 [Servlet이름]-servlet.xml 파일 설정

```
<?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:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
```

```
<!-- 경로와 Controller 매핑 설정 -->
```

```
<bean id=아이디 name=경로 class=처리할 클래스>
```

```
</bean>
```

```
<!-- ViewResolver 설정 - 출력할 뷰 파일을 설정하는 객체-->
```

```
<bean id="viewResolver"
```

```
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
```

```
  <property name="prefix" value="공통위치/>
```

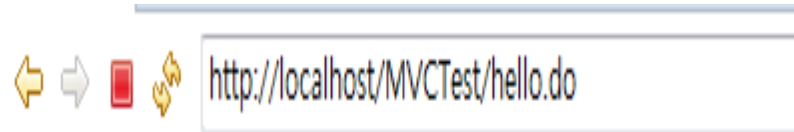
```
  <property name="suffix" value=" 확장자" />
```

```
</bean>
```

```
</beans>
```

# Spring MVC

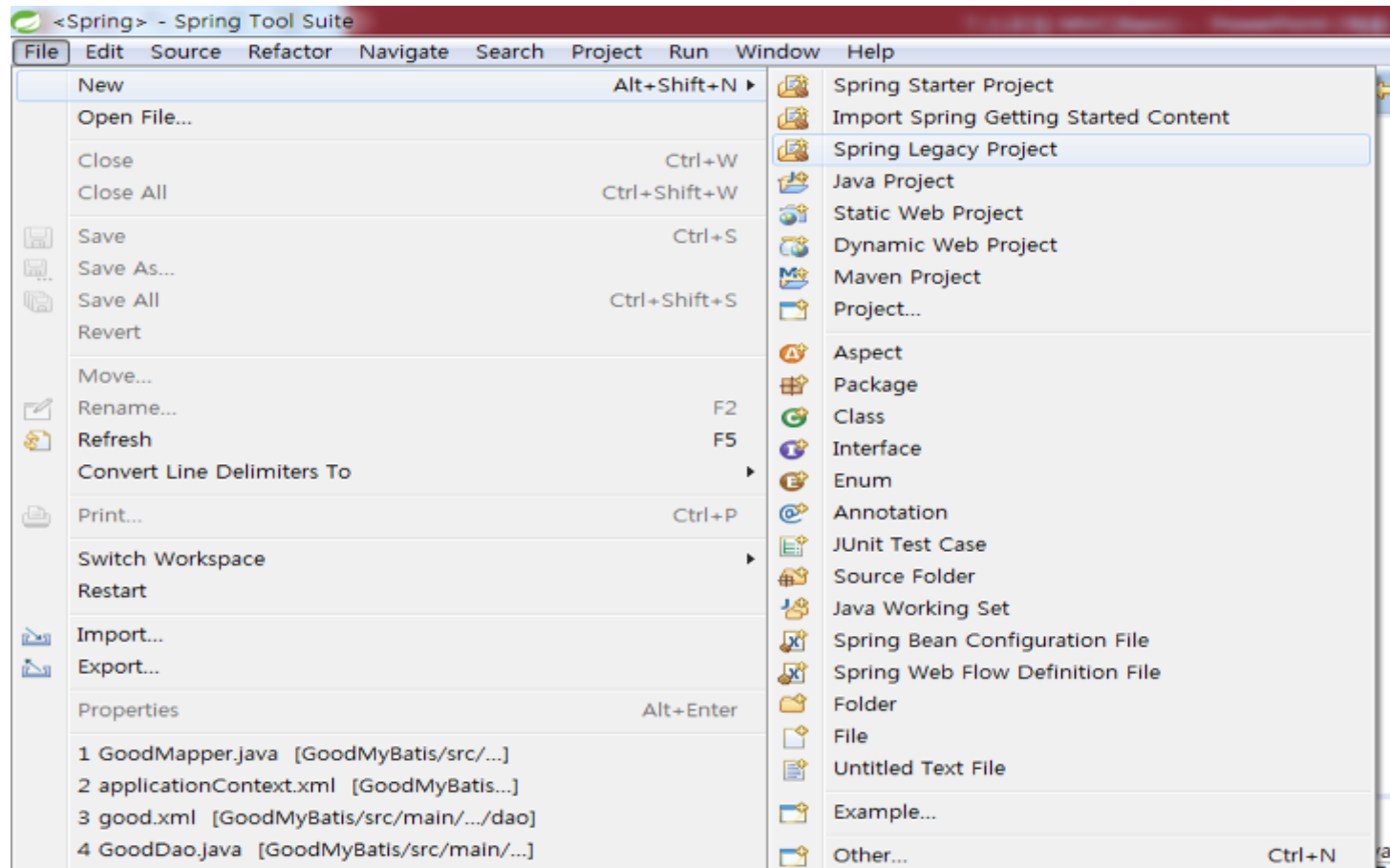
## ❖ Spring MVC Project



인사말: 안녕하세요

# Spring MVC

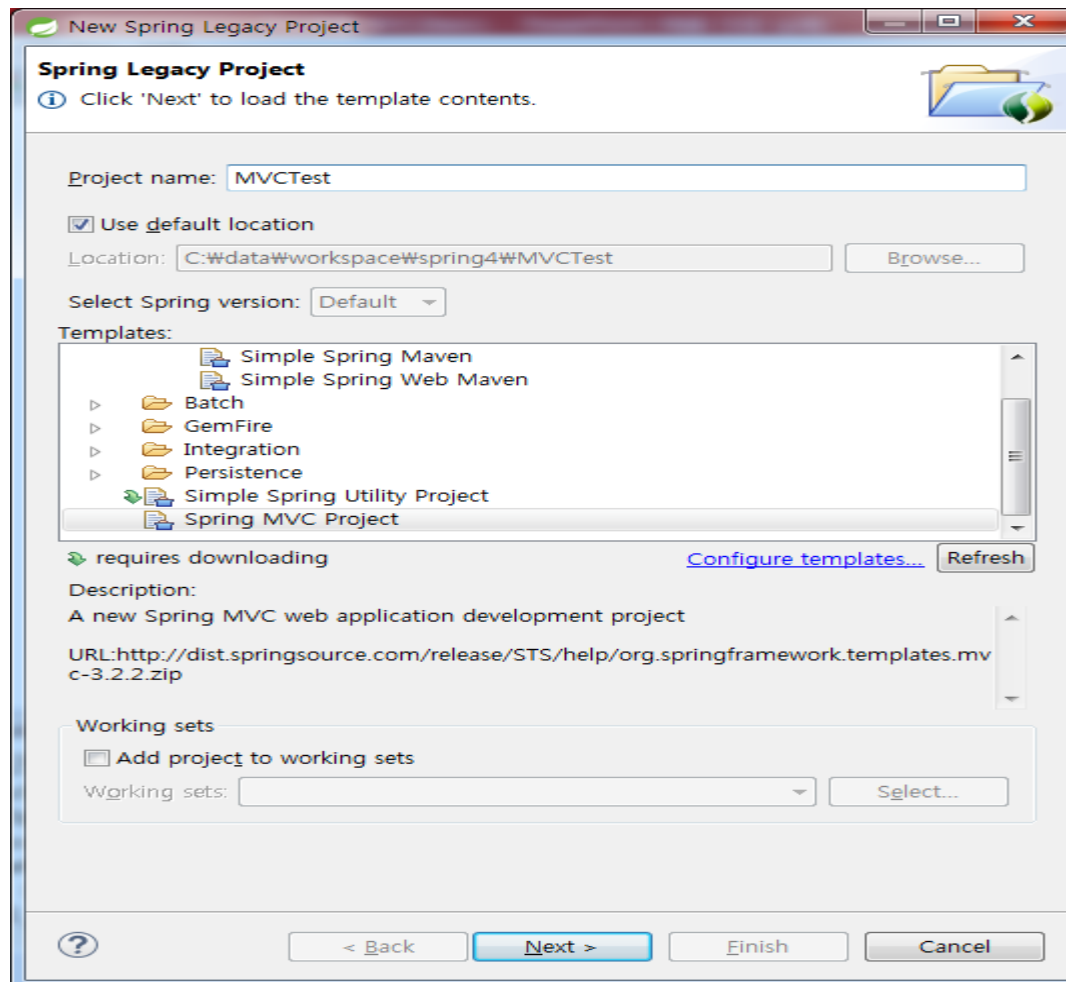
- ❖ Spring MVC Project
  - ✓ Spring MVC Project 생성





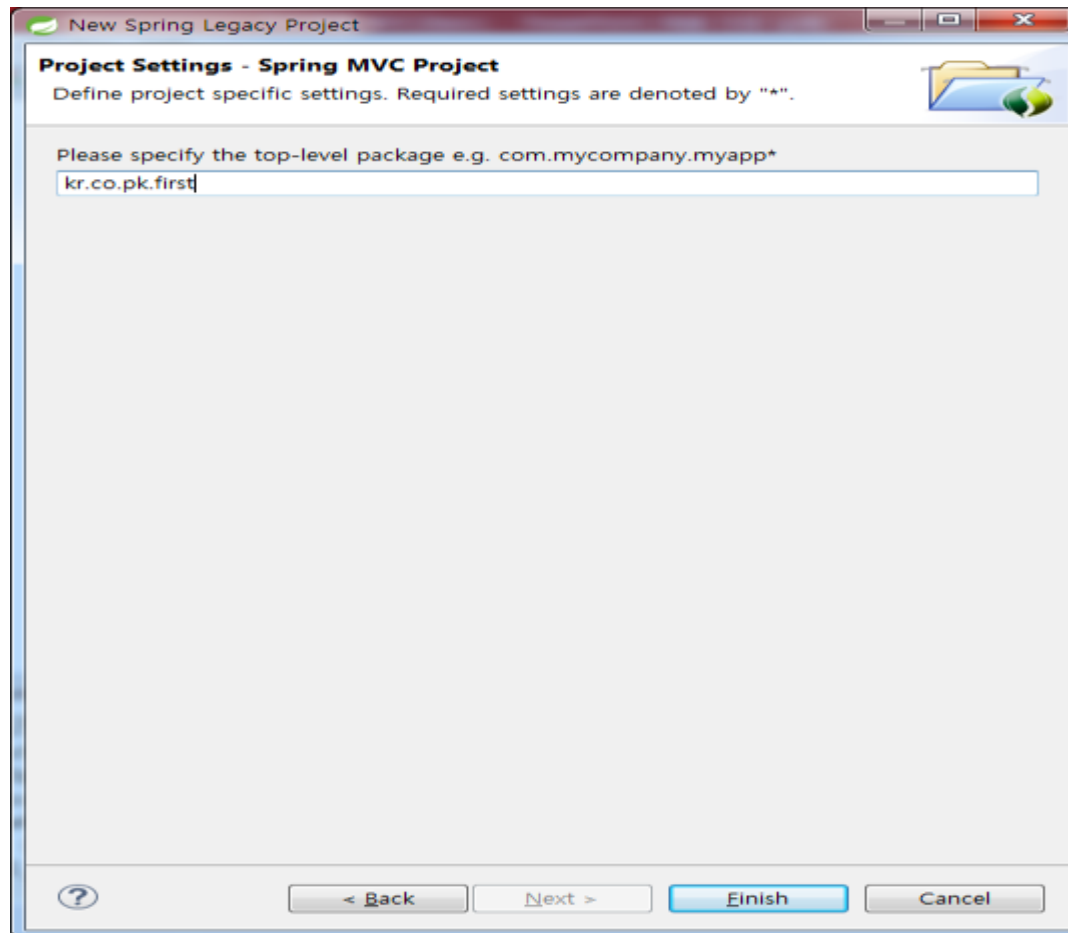
# Spring MVC

- ❖ Spring MVC Project
  - ✓ Spring MVC Project 생성



# Spring MVC

- ❖ Spring MVC Project
  - ✓ Spring MVC Project 생성



# Spring MVC

## ❖ Spring MVC Project

- ✓ java version(1.8) 과 spring version(5.0.7) 과 junit version(4.12) 변경 – pom.xml

```
<properties>
```

```
  <java-version>1.8</java-version>
```

```
  <org.springframework-version>5.0.7.RELEASE</org.springframework-version>
```

```
  <org.aspectj-version>1.6.10</org.aspectj-version>
```

```
  <org.slf4j-version>1.6.6</org.slf4j-version>
```

```
</properties>
```

# Spring MVC

## ❖ Spring MVC Project

- ✓ java version(1.8) 과 spring version(5.0.7) 과 junit version(4.12) 변경 – pom.xml

```
<!-- Test -->  
<dependency>  
  <groupId>junit</groupId>  
  <artifactId>junit</artifactId>  
  <version>4.12</version>  
  <scope>test</scope>  
</dependency>
```

# Spring MVC

## ❖ Spring MVC Project

- ✓ servlet 과 jsp 버전 변경 – pom.xml

```
<!-- Servlet -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
</dependency>

<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>javax.servlet.jsp-api</artifactId>
  <version>2.3.3</version>
</dependency>
```

# Spring MVC

## ❖ Spring MVC Project

- ✓ Maven의 자바 컴파일 버전 변경 – pom.xml

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>2.5.1</version>
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
    <compilerArgument>-Xlint:all</compilerArgument>
    <showWarnings>true</showWarnings>
    <showDeprecation>true</showDeprecation>
  </configuration>
</plugin>
```

# Spring MVC

- ❖ Spring MVC Project
  - ✓ 프로젝트를 실행



# Spring MVC

## ❖ Spring MVC Project

- ✓ 테스트를 위한 spring-test 와 DTO 생성을 편리하게 해주는 lombok 라이브러리 의존성 추가 – pom.xml

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

```
<dependency>  
  <groupId>org.projectlombok</groupId>  
  <artifactId>lombok</artifactId>  
  <version>1.18.0</version>  
  <scope>provided</scope>  
</dependency>
```



# Spring MVC

## ❖ Spring MVC Project

- ✓ WAS(WebContainer – Tomcat)을 이용해서 실행 - 프로젝트 이름을 제외하고 배포
  - 프로젝트를 선택하고 마우스 오른쪽을 눌러서 Properties 를 선택하고 Web Project Settings 의 Context root 속성을 /로 변경
  - Servers 디렉토리의 server.xml 파일의 Context 태그의 path를 /로 설정

# Spring MVC

## ❖ Spring MVC Project

- ✓ WEB-INF 디렉토리의 views 디렉토리의 home.jsp 파일을 수정

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>시작 페이지</title>
</head>
<body>
    <a href="hello">인사</a>
</body>
</html>
```

# Spring MVC

## ❖ Spring MVC Project

- ✓ src/main/java 디렉토리 안의 기본 패키지 안에 존재하는 HomeController 클래스의 기본 메서드를 삭제하고 추가

```
@Controller
public class HomeController {
    private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Locale locale, Model model) {
        logger.info("Welcome home! The client locale is {}.", locale);
        return "home";
    }

    @RequestMapping("/hello")
    public ModelAndView hello() {
        ModelAndView mav = new ModelAndView();
        mav.setViewName("hello");
        mav.addObject("greeting", "안녕하세요 Spring MVC Test 입니다.");
        return mav;
    }
}
```

# Spring MVC

## ❖ Spring MVC Project

- ✓ WEB-INF 디렉토리의 spring 디렉토리의 appServlet 디렉토리에 있는 servlet-context.xml 파일에서 viewResolver를 확인

```
<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>
```

# Spring MVC

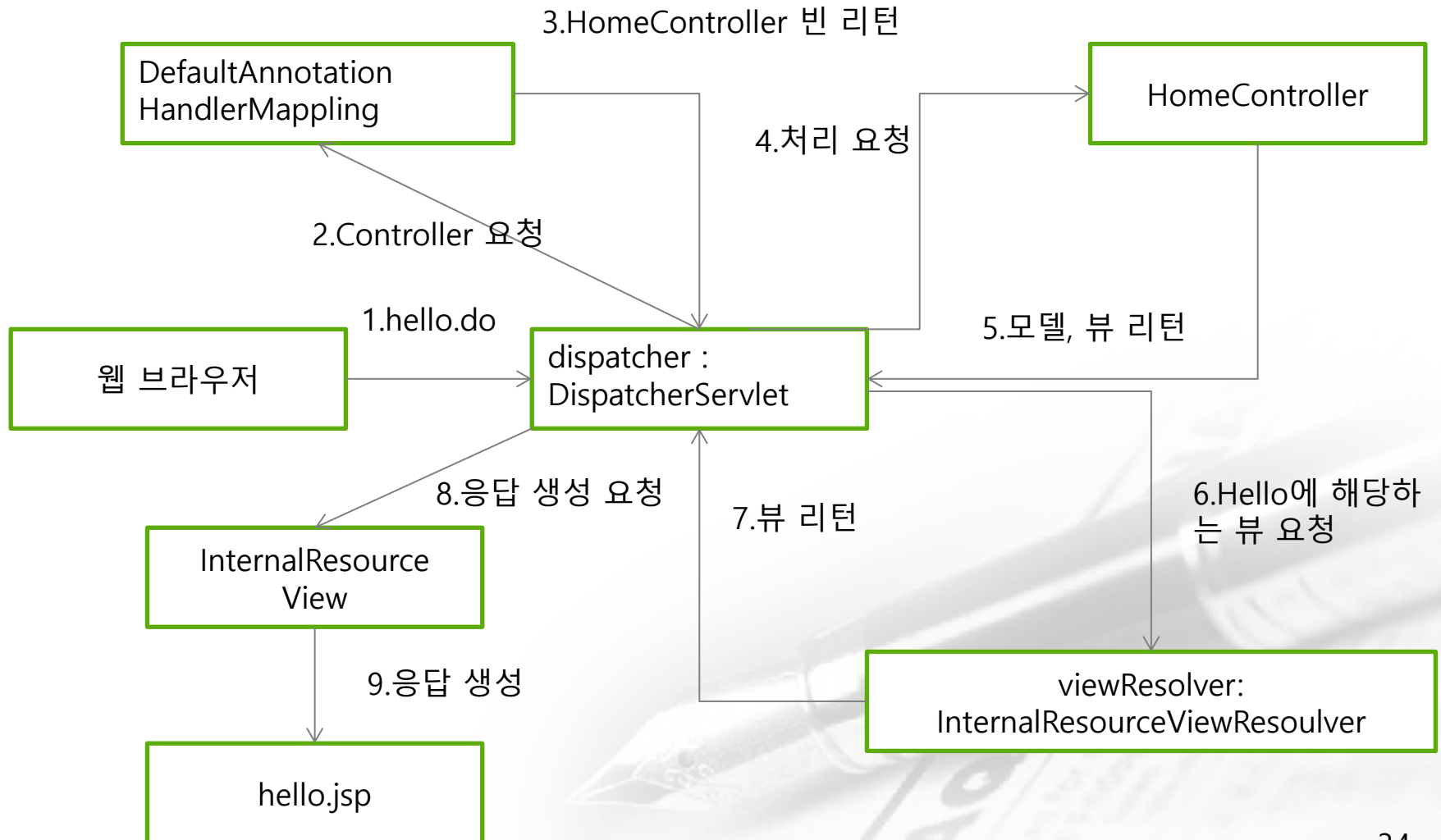
## ❖ Spring MVC Project

- ✓ webapp 디렉토리에 WEB-INF/views 디렉토리에 hello.jsp 파일을 작성

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

# Spring MVC

## ❖ Spring MVC Project



# Spring MVC

## ❖ DispatcherServlet

- ✓ DispatcherServlet은 클라이언트의 요청을 중앙에서 처리하는 스프링 MVC의 핵심 구성 요소
- ✓ web.xml 파일에 1개 이상의 DispatcherServlet을 설정할 수 있으며 각 DispatcherServlet은 한 개의 WebApplicationContext를 소유
- ✓ 각각의 WebApplicationContext를 소유하고 있어서 각각의 bean을 소유할 수 있으며 공유할 수 있는 bean을 소유 가능
- ✓ \*.do 요청이 오면 dispatcher-servlet.xml 파일에 작성된 설정 정보를 사용하도록 web.xml 파일에 설정

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

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

# Spring MVC

## ❖ DispatcherServlet

### ✓ URL 패턴

- /\* : 모든 요청 경로
- /: jsp 확장자를 제외한 모든 요청 경로
- /경로명/\*: /경로명이 포함된 모든 요청 경로
- \*.확장자: 확장자로 끝나는 모든 요청
- /경로명: 경로명에 해당하는 요청



# Spring MVC

## ❖ DispatcherServlet

### ✓ URL 패턴

- 초창기에는 확장자 패턴을 많이 사용했지만 최근에는 고정된 확장자 보다는 의미에 맞는 URL을 사용하는 곳이 증가하고 있는 추세

http://localhost:8080/springmvc/message/list

http://localhost:8080/springmvc/comic

위와 같은 경로가 존재한다면 아래처럼 설정해서 사용

```
<servlet-mapping>
```

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

```
    <url-pattern>/message/*</url-pattern>
```

```
    <url-pattern>/comic</url-pattern>
```

```
</servlet-mapping>
```

- /message/\*로 URL 패턴이 설정된 경우  
http://localhost:8080/springmvc/message/list.do 는 /list.do 로 처리
- /comic으로 설정된 경우 http://localhost:8080/springmvc/comic 은 /comic 로 처리
- \*.do 설정된 경우 http://localhost:8080/springmvc/message/list.do 는  
/message/list.do 로 처리

# Spring MVC

## ❖ DispatcherServlet

- ✓ 한 개 이상의 설정 파일을 이용하거나 설정 파일의 이름이나 위치를 변경하고자 하는 경우에는 contextConfigLocation Parameter에 value로 값을 설정

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

# Spring MVC

## ❖ DispatcherServlet

### ✓ 요청 처리

- 요청/응답 처리를 위해 필요한 변환 모듈이나 데이터 바인딩 처리를 위해서는 HandlerMapping 구현 객체 또는 HandlerAdapter 구현 객체가 필요한데 이 2개의 객체는 직접 생성하는 것 보다는 설정 파일에 <mvc:annotation-driven> 태그를 이용해서 자동으로 구현
- \*.jsp 요청을 제외한 모든 요청을 하나의 Servlet에서 처리하도록 하고자 할 때는 URL 패턴을 /로 설정
- URL 패턴을 / 나 /\* 로 설정을 할 때는 주의해야 하는데 위처럼 설정하게 되면 .css나 .js 와 같은 확장자를 가진 링크도 Controller가 처리하려고 해서 Controller에 처리하는 코드가 없으면 404 에러가 발생
- 설정 파일에 <mvc:default-servlet-handler>를 추가하게 되면 Default Servlet Handler가 설정되서 Controller에 처리하는 코드가 존재하지 않는 경우 404에러를 발생시키지 않고 WAS의 디폴트 Servlet에 요청을 전달

# Spring MVC

## ❖ Listener 설정

- ✓ 모든 Servlet이 공유해서 사용하는 빈이 있을 때는 web.xml 파일에 listener를 설정하면 되는데 사용하는 클래스는 org.springframework.web.context.ContextLoaderListener 클래스
- ✓ listener 이외의 다른 설정이 없으면 WEB-INF/applicationContext.xml 파일을 사용

--web.xml 파일에 기재 --

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

- ✓ context-param의 contextConfigLocation Parameter의 값을 설정하면 다른 파일을 공통 설정 파일로 사용이 가능

--web.xml 파일에 기재--

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>
    classpath:config/service.xml
    classpath:common.xml
    /WEB-INF/main.xml
  </param-value>
</context-param>
```

# Spring MVC

## ❖ Parameter 인코딩 설정

- ✓ POST 방식으로 전송된 Parameter를 읽을 때 인코딩을 설정하려면 `request.setCharacterEncoding("인코딩 방식")`을 기재
- ✓ Spring에서는 `CharacterEncodingFilter` 클래스를 제공
- ✓ web.xml 파일에 아래처럼 기재하면 모든 요청에서 Parameter 인코딩을 UTF-8로 변경

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

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

# Spring MVC

## ❖ Controller

### ✓ Controller 생성

- Spring 3.0 이전에는 Controller 인터페이스와 AbstractController 클래스, AbstractCommandController 클래스로부터 상속받아서 Controller를 구현했지만 3.0 이후부터는 @Controller Annotation을 이용해서 Controller를 구현하는 것을 권장
- @Controller Annotation은 POJO 클래스를 Controller 클래스로 변환해주는 역할을 수행하는데 설정 파일에 Controller 클래스를 빈으로 등록해야 하는데 대부분의 경우는 context:component-scan 을 이용해서 자동으로 생성

### ✓ 클래스에 작성하는 Annotation

- @Log4j Annotation을 추가하면 log.info 메서드를 이용해서 로그 출력 가능
- @RequestMapping("요청 경로")는 클래스에서 처리하는 모든 요청의 공통 URL을 설정하는 것으로 요청이 오면 메서드 위의 설정과 조합해서 요청을 처리

### ✓ Controller 클래스의 URL 매핑

- Controller 클래스 생성
- 클라이언트의 요청을 처리할 메서드에 @RequestMapping Annotation 적용

# Spring MVC

## ❖ Controller

### ✓ 기본적인 요청 처리 지정 방법

```
@Controller
public class FrontController{
    @RequestMapping("/event/list")
    public ModelAndView list(){
        내용;
    }
}
```

- 위처럼 설정되면 /event/list 라는 요청이 오면 list() 메서드를 호출하도록 설정한 것
- 실제 요청은 Servlet 매핑에 따른 Controller 경로 매핑과 디폴트 Servlet 설정에 따라 결정
- 프로젝트 이름(컨텍스트 경로)이 mvctest이고 URL 패턴이 /로 적용되어 있다면 http://서버경로:포트번호/mvctest/event/list 요청이 왔을 때 동작하게 되고 /main/\* 이 적용되어 있다면 http://서버경로:포트번호/mvctest/main/event/list 요청이 왔을 때 동작
- 디렉토리 형식으로 URL 패턴을 설정하면 디렉토리 부분을 제외한 부분을 처리하면 됨

# Spring MVC

## ❖ Controller

### ✓ 기본적인 요청 처리 지정 방법

- @RequestMapping에서는 요청 경로를 문자열로 직접 설정해도 되지만 value 속성에 문자열로 설정 가능
- Controller 클래스와 메서드에 함께 설정하는 경우도 있는데 이 경우에는 클래스의 요청 경로와 메서드에 설정된 요청 경로를 조합해서 매핑될 경로를 설정

```
@Controller
@RequestMapping("/event")
public class FrontController{
    @RequestMapping("/home")
    public ModelAndView home(){
    }
}
```

위 처럼 설정되면 /event/home 요청이 올 때 home() 메서드가 호출



# Spring MVC

## ❖ Controller

- ✓ 요청 경로에서 전송 방식을 지정하고자 하면 이 때는 method 속성을 이용

@RequestMapping(value="요청 경로", **method=RequestMethod.GET 또는 POST**)

- get 방식의 처리를 할 때 method를 생략하고 @RequestMapping 대신에 @GetMapping 으로 post 방식일 때는 @PostMapping을 사용할 수 있는데 이 경우는 Spring을 4.3 버전 이상을 사용해야 가능
- 여러 개의 경로를 매핑 하고자 하는 경우에는 value = "요청 경로" 대신에 value= {"경로", "경로"}를 입력
- @RequestMapping 메서드는 단순히 뷰 이름만 리턴해서 뷰 페이지를 지정할 수 있으며 클라이언트의 요청을 처리하고 처리 결과를 뷰에 전달하는 역할도 가능
- 기본 이동 방법은 포워딩

# Spring MVC

## ❖ Controller

### ✓ @PathVariable Annotation

- 요청 경로의 일부분을 변수로 만들어서 사용하는 Annotation
- 요청 경로의 가장 마지막 부분이 Parameter가 아닌 경우에 사용

요청 경로가 <http://host/member/10> 인 경우

@RequestMapping("/member/{memberId}") 로 설정

- ◆ public ModelAndView memberDetail(@PathVariable String memberId)로 메서드를 만들게 되면 memberId에 값이 대입되서 Parameter 처럼 사용 가능 – tistory 가 대표적
- ◆ 주의할 점은 RequestMapping에 적용된 변수 이름과 PathVariable에 사용된 변수 이름이 같아야 하는데 다르게 설정하고자 하는 경우에는 @PathVariable("memberId") String id 처럼 @PathVariable 뒤에 변수를 선언
- ◆ 데이터의 형 변환은 자동으로 수행

# Controller

## ❖ index.jsp 파일에 링크 추가

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>시작 페이지</title>
</head>
<body>
    <a href="hello.do">인사</a> <br />
    <a href="message/detail/10">디렉토리 패턴의 매개변수</a>
</body>
</html>
```

# Controller

- ❖ HomeController 클래스에 요청 처리 메서드 추가

```
@RequestMapping("/message/detail/{num}")
public ModelAndView detail(@PathVariable int num) {
    ModelAndView mav = new ModelAndView();
    mav.setViewName("detail");
    mav.addObject("num", num);
    return mav;
}
```

# Controller

- ❖ view 디렉토리에 detail.jsp 파일을 추가하고 작성

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

# Controller

## ❖ Ant 패턴을 이용한 경로 매핑

✓ Ant 패턴은 \*, \*\*, ? 세 가지 와일드카드 문자를 이용해서 경로를 표현하는 패턴

✓ 와일드카드 문자

❑ \* - 0개 이상의 문자

❑ \*\* - 0개 이상의 디렉토리 경로

❑ ? - 1개의 문자

✓ 사용 예

❑ @RequestMapping("/member/?\*.info") : /member/로 시작하고 확장자가 info로 끝나는 요청

❑ @RequestMapping("/member/f\*.info") : /member/로 시작하고 f로 시작하고 확장자가 info로 끝나는 요청

❑ @RequestMapping("/member/\*\*/files") : /member/로 시작하고 중간에 디렉토리가 존재하고 /files로 끝나는 요청

# Controller

- ❖ 처리 가능한 요청 콘텐츠 타입 또는 응답 가능한 콘텐츠 타입 한정
  - ✓ 처리 가능한 요청 콘텐츠 타입 한정
    - ❑ consumes 속성 이용
    - ❑ `@RequestMapping("value="/members", consumes=MediaType.APPLICATION_JSON_VALUE)`
  - ✓ 응답 가능한 콘텐츠 타입 한정
    - ❑ produces 속성 이용
    - ❑ `@RequestMapping("value="/members", produces=MediaType.APPLICATION_JSON_VALUE)`

# Controller

## ❖ HTTP 요청 Parameter 처리

### ✓ Parameter 처리 방법

- ❑ HttpServletRequest 객체의 String `getParameter("Parameter 이름")` 또는 `String[] getParameterValues("Parameter 이름")` 메서드를 이용하는데 이 메서드는 Parameter가 없으면 null을 리턴
- ❑ @RequestParam Annotation 이용
- ❑ Command 객체를 이용하는 방법



# Controller

- ❖ **index.jsp** 에 새글 작성을 클릭하면 **newArticle.do** 요청이 발생하도록 링크 추가

`<a href="newarticle">새글</a>`

- ❖ **HomeController**에 **newarticle.do** 요청을 처리하는 메서드를 생성

```
@RequestMapping("/newarticle")
public ModelAndView newArticle() {
    ModelAndView mav = new ModelAndView();
    mav.setViewName("newarticleform");
    return mav;
}
```

# Controller

## ❖ 이전 프로젝트의 webapp/view 디렉토리에 newarticleform.jsp 파일 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>게시글 쓰기</title>
<script>
```

```
</script>
```

```
</head>
```

```
<body>
```

게시글 쓰기 입력 폼:

```
<form method="post" action="form" id="articleform">
```

```
    제목: <input type="text" name="title" /> <br/>
```

```
    내용: <textarea name="content"> </textarea> <br/>
```

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

```
</form>
```

```
</body>
```

```
</html>
```

# Controller

## ❖ HomeController에 form.do 요청을 처리하는 메서드를 생성

```
@RequestMapping("/form")
public ModelAndView articleSave(HttpServletRequest request) {
    ModelAndView mav = new ModelAndView();
    System.out.println("제목:" + request.getParameter("title"));
    System.out.println("내용:" + request.getParameter("content"));
    return mav;
}
```

# Controller

## ❖ Parameter 인코딩 설정을 위해 web.xml 파일에 추가

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

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

# Controller

## ❖ @RequestParam Annotation

- ✓ Parameter를 읽어오기 위한 Annotation
- ✓ 기본 형식

```
@RequestMapping("/요청")  
public ModelAndView method(@RequestParam("query") String query,  
    @RequestParam("page") int page){  
    메서드 내용;  
}
```

- ✓ query Parameter는 query 변수에 대입되고 page Parameter는 page에 저장되는데 데이터의 타입 변환은 자동으로 수행
- ✓ 데이터의 형 변환은 자동으로 수행
- ✓ Parameter가 없거나 형 변환이 불가능한 데이터를 전달하는 경우 Controller는 메서드를 실행하지 않고 400 응답 코드를 웹 브라우저에 전송

# Controller

## ❖ @RequestParam Annotation

- ✓ 설정된 Parameter는 필수 요소로서 반드시 전달되어야 하며 만일 필수 요소를 해제하고자 하면 `required=false`를 붙여서 선언  
`@RequestParam(value="query" required=false)`
- ✓ `defaultValue` 속성을 이용해서 기본 값을 지정 가능  
`@RequestParam(value="query" defaultValue="")`
- ✓ 동일한 이름의 Parameter가 여러 개인 경우는 `ArrayList` 나 배열로 전달받을 수 있음
- ✓ 동일한 방법으로 `@RequestParam` 대신에 `@CookieValue("쿠키이름")`를 이용하면 쿠키 값을 전달 할 수 있으며 `@RequestHeader("헤더이름")`을 이용하면 헤더 값도 전달 가능

# Controller

## ❖ HomeController에 form.do 요청을 처리하는 메서드를 수정

```
@RequestMapping("/form")
public ModelAndView articleSave(
    @RequestParam("title")String title,
    @RequestParam("content") String content) {
    ModelAndView mav = new ModelAndView();
    System.out.println("제목:" + title);
    System.out.println("내용:" + content);
    return mav;
}
```

# Controller

## ❖ Parameter를 Command 객체를 이용해서 처리

- ✓ Parameter를 Command 객체로 전달받고 하면 Parameter 이름을 프로퍼티로 갖는 클래스를 만들어서 Controller의 요청 처리 메서드에서 매개변수로 처리 가능

```
@RequestMapping("/form.do")
```

```
public ModelAndView formProcessing(Article command) {  
    command.getTitle();  
}
```

- ✓ 요청 처리 메서드에서 Command 객체를 이용해서 Parameter를 받으면 Parameter를 별도로 넘기지 않아도 결과 페이지(View)로 자동 넘겨가게 되는데 이때는 매개변수 이름으로 넘겨가는 것이 아니고 클래스 이름을 가지고 넘겨가는데 첫 글자만 소문자로 변환
- ✓ request 객체의 attribute에 저장되서 전달
- ✓ 자동으로 넘겨가는 Parameter의 이름을 변경하고자 하는 경우에는 매개변수의 앞에 @ModelAttribute("다른이름")을 추가

```
public ModelAndView formProcessing(@ModelAttribute("command") Article command)
```



# Controller

## ❖ 기본 패키지에 domain 패키지를 만들고 Article 클래스 생성

```
public class Article {  
    private String title;  
    private String content;  
    public String getTitle() {  
        return title;  
    }  
    public void setTitle(String title) {  
        this.title = title;  
    }  
    public String getContent() {  
        return content;  
    }  
    public void setContent(String content) {  
        this.content = content;  
    }  
    @Override  
    public String toString() {  
        return "Article [title=" + title + ", content=" + content + "];"  
    }  
}
```

# Controller

## ❖ HomeController의 form.do 요청을 처리하는 메서드 수정

```
@RequestMapping("/form")
public ModelAndView formProcessing(Article command) {
    ModelAndView mav = new ModelAndView();
    mav.setViewName("form");
    return mav;
}
```

# Controller

## ❖ view 디렉토리에 form.jsp 파일 추가

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>내용 출력</title>
</head>
<body>
    제목:${article.title } <br />
    내용:${article.content}<br />
</body>
</html>
```

# Controller

## ❖ @InitBinder

- ✓ Parameter를 변환해서 처리해야 할 때 사용
- ✓ 문자열을 Date로 변환해서 사용해야 하는 경우 사용
- ✓ Controller 클래스에 아래처럼 메서드를 생성

```
@InitBinder
public void initBinder(WebDataBinder binder){
    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");
    binder.registerCustomEditor(java.util.Date.class, new
    CustomDateEditor(dateFormat, false));
}
```

## ❖ @DateTimeFormat

- ✓ Command 객체를 사용하는 경우 Date 타입에 패턴을 설정하고자 하는 경우 사용
- ```
@DateTimeFormat(pattern = "yyyy/MM/dd")
private Date dueDate;
```
- ✓ 문자열로 yyyy/MM/dd 형식의 문자열이 맞으면 날짜 타입으로 변환

# Controller

## ❖ @ModelAttribute

- ✓ Controller 클래스의 메서드 이름 위에 @ModelAttribute("이름")을 추가하면 메서드의 리턴되는 데이터가 Controller 안에서 설정된 모든 View에 request 객체의 attribute로 전달
- ✓ request.setAttribute("이름", 리턴되는 데이터) 를 수행한 것과 동일한 효과

# Controller

## ❖ Controller 메서드의 Parameter 타입

- ✓ **HttpServletRequest, HttpServletResponse, HttpSession**
  - ✓ java.util.Locale: 현재 요청에 대한 Locale
  - ✓ Reader: 요청 콘텐츠에 직접 접근하고자 하는 경우
  - ✓ Writer: 응답을 직접 생성하는 경우
  - ✓ **Map, Model, ModelMap: 뷰에 전달할 모델 데이터를 설정할 때 이용**
  - ✓ **@RequestParam: 요청 Parameter를 처리할 때 이용**
  - ✓ @CookieValue: 쿠키 값 요청
  - ✓ @RequestHeader: 헤더 값 요청
  - ✓ @PathVariable: 경로의 일부분을 데이터로 사용
  - ✓ **Errors, BindingResult: 요청 Parameter를 커맨드 객체에 저장한 결과, 커맨드 객체를 위한 Parameter 다음에 위치해서 에러 메시지를 전달**
  - ✓ SessionStatus: 세션 처리를 완료 했음을 처리하기 위해 사용하는 것으로 @SessionAttributes Annotation을 명시한 Session 속성을 제거하기 위한 것
  - ✓ **커맨드 객체: 요청 Parameter를 저장한 객체, @ModelAttribute 와 함께 사용해서 결과 페이지에 전송하는 이름을 변경할 수 있음**
  - ✓ RedirectAttributes: 리다이렉트 할 때 1번만 사용할 수 있는 데이터를 위한 자료형
- ❖ Errors와 BindingResult를 제외한 모든 Parameter는 순서에 상관없이 사용 가능

# Controller

- ❖ Controller의 요청 처리(@RequestMapping이 적용) 메서드의 리턴 타입
  - ✓ String: View 이름을 리턴
  - ✓ void: Parameter가 HttpServletResponse 인 경우 출력을 직접 할 수 있으므로 뷰를 생성할 필요가 없는 경우 또는 URL의 일부분을 가지고 뷰 이름을 결정하는 경우 사용
  - ✓ ModelAndView: 뷰 정보 및 모델 정보를 저장한 ModelAndView 객체
  - ✓ Model: 뷰에 전달할 객체 정보를 담고 있는 Model을 리턴
  - ✓ Map: 뷰에 전달할 객체 정보를 저장한 Map을 리턴
  - ✓ View 객체: 직접 뷰를 생성
  - ✓ VO, List 타입: 주로 JSON 타입의 데이터를 반환할 때 이용
  - ✓ ResponseEntity: response 할 때 HTTP 헤더 정보와 내용을 가공하고자 할 때 사용하는데 RestController에서 JSON을 반환하고자 할 때 사용
  - ✓ HttpHeaders: 응답에 내용없이 HTTP 헤더 메시지만 전달하고자 할 때

# View 설정

- ❖ View 이름 설정 – Controller 의 요청 처리 메서드나 설정이 결정
  - ✓ ModelAndView를 리턴할 때는 ModelAndView의 생성자나 setViewName(String view)을 이용해서 View 이름을 지정
  - ✓ String을 이용할 때는 리턴하는 문자열 값이 View 이름
  - ✓ 2가지 경우는 RequestToViewNameTranslator를 이용해서 URL로부터 뷰 이름을 결정
    - ❑ 리턴 타입이 Model 이나 Map 인 경우
    - ❑ 리턴 타입이 void 이면서 HttpServletResponse 타입의 Parameter가 없는 경우에는 요청 URI로부터 맨 앞의 슬래시와 확장자 패턴인 경우에는 확장자를 제외한 부분을 뷰 이름으로 사용
- ✓ View 이름에 **redirect:** 접두어를 붙이면 Redirect
  - ❑ ViewResolver가 적용되지 않음

redirect:/bbs/list

redirect:http://www.choongang.co.kr



# Model 설정

## ❖ 뷰에 출력할 데이터 전달

### ✓ 데이터 설정 방법

- ❑ HttpServletRequest, HttpSession, ServletContext(application 객체) 객체를 이용한 전달
- ❑ command 객체 - @ModelAttribute Annotation을 이용해서 이름 변경 가능
- ❑ @ModelAttribute Annotation이 적용된 메서드가 리턴한 객체
- ❑ 메서드의 Map, Model, ModelMap 타입의 Parameter를 통해 설정된 모델

### ✓ redirect의 경우의 데이터 전달

- ✓ HttpSession 객체 이용
- ✓ 요청 처리 메서드의 Parameter에 RedirectAttributes 타입의 매개변수를 설정한 후 addFlashAttribute 메서드를 호출해서 속성과 값을 설정해서 데이터를 넘길 수 있는데 이 때는 데이터가 계속 유지되지 않고 결과 페이지까지에서만 데이터가 유지

# Model 설정

## ❖ 뷰에 출력할 데이터 전달

### ✓ 데이터를 전달하기 위한 자료형의 데이터 저장

- ❑ Map: Map은 HashMap을 생성해서 데이터를 저장한 후 리턴
- ❑ Model 인터페이스는 addAttribute("키이름", 데이터)를 이용해서 데이터를 추가
- ❑ ModelMap 클래스의 경우는 Model 인터페이스의 구현 클래스를 이용해서 데이터를 넘겨줄 수 있음
- ❑ ModelAndView 클래스
  - addObject("키이름", 데이터)
  - addAllObjects(Map을 구현한 객체)

# Model 설정

## ❖ index.jsp 파일에 링크 추가

`<a href="redirect">리다이렉트</a>`

`<a href="header">헤더설정</a>`



# Model 설정

## ❖ HomeController에 요청 처리 메서드 추가

```
@RequestMapping("/redirect")
public String redirect(RedirectAttributes rtr) {
    rtr.addFlashAttribute("msg", "임시 데이터 전달");
    return "redirect:disp";
}
```

```
@RequestMapping("/disp")
public String disp(@ModelAttribute String msg) {
    return "redirect";
}
```

# Model 설정

## ❖ view 디렉토리에 redirect.jsp 파일을 추가

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

# Model 설정

## ❖ HomeController에 요청 처리 메서드 추가

```
@RequestMapping("/header")
public ResponseEntity<String> header() {
    String msg = "{₩"name₩" : ₩"박문석₩"}";
    HttpHeaders header = new HttpHeaders();
    header.add("Content-type", "application/json;charset=UTF-8");
    return new ResponseEntity<>(msg, header, HttpStatus.OK);
}
```

# HTTP Session 사용

- ❖ 여러 View 나 redirect 한 경우 데이터 공유 방법
  - ✓ view 마다 공유할 데이터를 계속해서 전달
  - ✓ 임시 데이터를 데이터베이스에 저장
  - ✓ 세션에 데이터를 저장
  - ✓ 쿠키를 이용한 저장
  - ✓ HTML5의 ApplicationStorage 나 SessionStorage를 이용
- ❖ HttpSession
  - ✓ 접속하는 브라우저마다 생성되는 서버의 메모리에 존재하는 객체
  - ✓ 트래픽이 작거나 단일 서버에서 동작하는 웹 애플리케이션의 경우 브라우저 내의 여러 View에서 데이터를 공유하고자 하는 경우 HttpSession을 이용해서 데이터를 공유
  - ✓ 임시 데이터를 보관하기 위해서 세션을 사용하는 것은 사용자마다 세션이 생성되는 것을 의미하는데 접속자 수가 늘어날수록 세션 객체 개수가 증가하고 이로 인해 세션 객체가 차지하는 메모리 비중이 높아지게 되는데 트래픽이 높을 것으로 예상하는 곳에서는 메모리 부족 현상이 일어날 수 있기 때문에 이러한 경우에는 데이터베이스나 외부 캐시 서버를 사용하는 것을 권장

# HTTP Session 사용

## ❖ HttpSession

### ✓ HttpSession 사용

- ❑ Controller의 요청 처리 메서드에서 HttpServletRequest를 넘겨받은 후 getSession()을 호출한 후 리턴받은 데이터를 사용
- ❑ 요청 처리 메서드에서 HttpSession Parameter를 넘겨받을 수 있도록 설정
- ❑ @SessionAttribute("세션에 저장할 이름")
  - 클래스에 이 Annotation을 적용하면 세션에 공유할 객체의 모델 이름을 설정하는 것이 되서 Model에 저장할 때 모델 이름을 사용하면 세션에 저장
  - 공유한 데이터의 사용이 끝나면 SessionStatus를 매개변수로 받아서 setComplete() 를 호출하면 세션에서 객체를 제거하지만 모델에서 제거하지는 않음
- ✓ 세션을 초기화 할 때는 invalidate 메서드를 호출하거나 HttpSession 객체의 메서드 나 web.xml 파일에 세션 만료 시간을 설정하면 됨



# 상품 목록 출력

## 상품 목록 화면

| 상품 ID | 상품명        | 가격    |
|-------|------------|-------|
| 1     | Lemon      | 500원  |
| 2     | Orange     | 1500원 |
| 3     | Kiwi       | 2000원 |
| 4     | Grape      | 1000원 |
| 5     | Strawberry | 2000원 |
| 6     | Mandarin   | 300원  |

# 상품 목록 출력

## ❖ 오라클 데이터베이스에 테이블과 샘플 데이터 입력

DROP TABLE ITEM;

```
CREATE TABLE ITEM(  
    itemid number(5),  
    itemname VARCHAR2(100),  
    price number(6),  
    description VARCHAR2(200),  
    pictureurl VARCHAR2(100),  
    PRIMARY KEY (itemid)  
);
```

# 상품 목록 출력

## ❖ 오라클 데이터베이스에 테이블과 샘플 데이터 입력

```
insert into item values(1, 'Lemon', 500, 'Vitamin-A', 'lemon.jpg');  
insert into item values(2, 'Orange', 1500, 'Vitamin-B', 'orange.jpg');  
insert into item values(3, 'Kiwi', 2000, 'Vitamin-C', 'kiwi.jpg');  
insert into item values(4, 'Grape', 1000, 'Vitamin-D', 'grape.jpg');  
insert into item values(5, 'Strawberry', 2000, 'Vitamin-E', 'strawberry.jpg');  
insert into item values(6, 'Mandarin', 300, 'Vitamin-F', 'mandarin.jpg');
```

```
commit;
```

```
select * from ITEM;
```

# 상품 목록 출력

## ❖ pom.xml 파일에 오라클을 사용하기 위한 repositories 추가

```
<repositories>
  <repository>
    <id>oracle</id>
    <name>ORACLE JDBC Repository</name>
    <url>http://maven.jahia.org/maven2</url>
  </repository>
</repositories>
```



# 상품 목록 출력

- ❖ pom.xml 파일에 오라클, spring-jdbc, mybatis, mybatis-spring, lombok, spring-test, log4jdbc를 사용하기 위한 repositories 추가

```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc7</artifactId>
  <version>12.1.0.2</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>${org.springframework-version}</version>
</dependency>
```

# 상품 목록 출력

- ❖ pom.xml 파일에 오라클, spring-jdbc, mybatis, mybatis-spring, lombok, spring-test, log4jdbc를 사용하기 위한 repositories 추가

```
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis</artifactId>
    <version>3.5.7</version>
</dependency>
<dependency>
    <groupId>org.mybatis</groupId>
    <artifactId>mybatis-spring</artifactId>
    <version>2.0.6</version>
</dependency>

<dependency>
    <groupId>org.bgee.log4jdbc-log4j2</groupId>
    <artifactId>log4jdbc-log4j2-jdbc4</artifactId>
    <version>1.16</version>
</dependency>
```

# 상품 목록 출력

## ❖ web.xml 파일의 DTD를 3.1.0 으로 변경

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee  
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"  
version="3.1">
```

# 상품 목록 출력

- ❖ **root-context.xml** 파일에 데이터베이스 접속 정보를 저장할 **DataSource** 클래스의 객체를 생성하는 코드를 추가

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">

    <property name="driverClassName">
        <value>oracle.jdbc.driver.OracleDriver</value>
    </property>
    <property name="url">
        <value>jdbc:oracle:thin:@localhost:1521:xe</value>
    </property>
    <property name="username">
        <value>system</value>
    </property>
    <property name="password">
        <value>wnddkd</value>
    </property>
</bean>
```



# 상품 목록 출력

## ❖ Test 클래스를 만들고 데이터베이스 접속을 확인

```
import org.apache.ibatis.session.SqlSession;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "file:src/main/webapp/WEB-INF/spring/root-
context.xml"})
public class SampleTesting {
    @Autowired
    private DataSource dataSource;
    @Test
    public void testFactory() {
        System.out.println(ds);
        try(Connection con = ds.getConnection()){
            System.out.println(con);
        }catch(Exception e){
            System.out.println(e.getLocalizedMessage());
        }
    }
}
```

# 상품 목록 출력

- ❖ 기본패키지.domain 패키지에 테이블의 데이터를 표현하는 Item 클래스 생성 - @Data를 했을 때 getter 와 setter 가 자동으로 만들어지지 않으면 getter&setter 그리고 toString을 추가해주면 됩니다.

```
import lombok.Data;
```

```
@Data
public class Item {
    private int itemid;
    private String itemname;
    private int price;
    private String description;
    private String pictureurl;
}
```

# 상품 목록 출력

- ❖ **src/main/resources** 디렉토리에 **mybatis** 디렉토리를 생성하고 MyBatis 환경 설정으로 사용할 **mybatis-config.xml** 파일을 생성하고 작성

```
<!DOCTYPE configuration
PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <typeAliases>
    <package name="kr.co.adamsoft.domain"/>
  </typeAliases>
</configuration>
```

# 상품 목록 출력

- ❖ **src/main/resources** 디렉토리의 **mybatis** 디렉토리에 **mappers** 디렉토리를 생성하고 **MyBatis sql** 파일로 사용할 **item.xml** 파일을 생성하고 **namespace**를 설정

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

# 상품 목록 출력

- ❖ **root-context.xml** 파일에 **MyBatis**를 사용할 수 있는 클래스의 객체를 생성하는 코드를 추가

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

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

# 상품 목록 출력

- ❖ 트랜잭션을 사용하기 위해서 root-context.xml 파일에 추가 – tx 네임스페이스 추가

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

```

```
</bean>
```

```
<tx:annotation-driven />
```

# 상품 목록 출력

## ❖ Test 클래스를 만들고 MyBatis 설정을 확인

```
import org.apache.ibatis.session.SqlSession;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "file:src/main/webapp/WEB-INF/spring/root-
context.xml"})
public class SampleTesting {
    @Autowired
    private SqlSession sqlSession;
    @Test
    public void testFactory() {
        System.out.println(sqlSession);
    }
}
```

# 상품 목록 출력

## ❖ Item 작업을 위한 구조 만들기

- ✓ 기본패키지.dao 패키지에 ItemDao 클래스를 만들고 작성

```
@Repository
public class ItemDao {

    @Autowired
    private SqlSession sqlSession;

}
```



# 상품 목록 출력

## ❖ Item 작업을 위한 구조 만들기

- ✓ 기본패키지.service 패키지에 ItemService 인터페이스를 생성

```
public interface ItemService {  
}
```

- ✓ 기본패키지.service 패키지에 ItemService 인터페이스의 내용을 구현할 ItemServiceImpl 클래스를 생성

```
@Service  
public class ItemServiceImpl implements ItemService {  
    @Autowired  
    private ItemDao itemDao;  
}
```

# 상품 목록 출력

## ❖ Item 작업을 위한 구조 만들기

- ✓ 기본패키지.HomeController 클래스에 ItemService 를 주입하는 코드를 작성

@Autowired

```
private ItemService itemService;
```



# 상품 목록 출력

## ❖ item.xml 파일에 테이블의 데이터를 전부 읽어오는 sql을 생성

```
<?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="itemmapper">
  <select id="allItem" resultType="Item">
    SELECT itemid, itemname, price, description, pictureurl FROM item
  </select>
</mapper>
```

# 상품 목록 출력

- ❖ ItemDao를 클래스에 테이블의 전체 데이터를 가져오는 메서드를 생성

```
public List<Item> getAll() {  
    return sqlSession.selectList("itemmapper.allItem");  
}
```

# 상품 목록 출력

- ❖ ItemService 인터페이스에 전체 데이터를 가져오는 메서드를 선언

```
public List<Item> getAll();
```



# 상품 목록 출력

- ❖ ItemServiceImpl 클래스에 전체 데이터를 가져오는 메서드를 구현

```
@Override  
public List<Item> getAll() {  
    return itemDao.getAll();  
}
```

# 상품 목록 출력

## ❖ HomeController 클래스에 시작 요청을 처리하는 메서드를 수정

```
@RequestMapping(value = "/", method = RequestMethod.GET)
public String home(Model model) {
    model.addAttribute("list", itemService.getAll());
    return "home";
}
```

# 상품 목록 출력

❖ index.jsp 파일을 삭제

❖ view 디렉토리에 home.jsp 파일을 생성하고 작성

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

```
<!-- JSTL Core 기능 사용 설정 -->
```

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

```
<!-- EL을 사용하기 위한 설정으로 없어도 되는 경우가 많지만 간혹 EL을 가지고 출력이 안되는
경우가 있어서 설정 -->
```

```
<%@ page isELIgnored="false"%>
```

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

```
<html>
```

```
<head>
```

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

```
<title>시작 페이지</title>
```

```
<!-- 웹에서 링크 설정은 경로를 기준으로 합니다. 파일의 위치가 아닙니다. -->
```

```
<link rel="stylesheet" type="text/css" href="/css/style.css">
```

```
</head>
```



## 상품 목록 출력

## ❖ view 디렉토리에 home.jsp 파일을 생성하고 작성

<body>

<a href="hello">인사</a>

[디렉토리 패턴의 매개변수](message/detail/10)

<a href="newarticle">새글</a>

<a href="redirect">리다이렉트</a>

<a href="header">JSON</a>

# 상품 목록 출력

❖ view 디렉토리에 home.jsp 파일을 생성하고 작성

```
<div align="center" class="body">
  <h2>상품 목록 화면</h2>
  <table border="1">
    <tr class="header">
      <th align="center" width="80">상품ID</th>
      <th align="center" width="320">상품명</th>
      <th align="center" width="100">가격</th>
    </tr>
    <c:forEach var="item" items="${list}">
      <tr class="record">
        <td align="center">${item.itemid}</td>
        <td align="left">${item.itemname}</td>
        <td align="right">${item.price}원</td>
      </tr>
    </c:forEach>
  </table>
</div>
</body>
</html>
```

# 상품 목록 출력

- ❖ webapp 디렉토리 안에 css 디렉토리를 만들고 style.css 파일 작성

```
div.body{  
    overflow-y: auto;  
    scrollbar-face-color: #C9BFED;  
    scrollbar-shadow-color: #EDEDED;  
    margin-top: 50px;  
    margin-bottom: 50px;  
}
```

```
tr.header{  
    background: #C9BFED;  
}
```

```
tr.record{  
    background: #EDEDED;  
}
```

# 상품 목록 출력

- ❖ **servlet-context.xml** 파일에 Controller가 처리하지 못하는 URL은 WAS가 처리하도록 해주는 태그 추가

`<default-servlet-handler/>`



# 상품 상세 보기

## 상품 목록 화면

| 상품 ID | 상품명                        | 가격    |
|-------|----------------------------|-------|
| 1     | <a href="#">Lemon</a>      | 500원  |
| 2     | <a href="#">Orange</a>     | 1500원 |
| 3     | <a href="#">Kiwi</a>       | 2000원 |
| 4     | <a href="#">Grape</a>      | 1000원 |
| 5     | <a href="#">Strawberry</a> | 2000원 |
| 6     | <a href="#">Mandarin</a>   | 300원  |

# 상품 상세 보기

## 상품 상세 화면



상품명     Mandarin

가격        300원

비고        Vitamin-F

■ [목록으로 돌아가기](#)

# 상품 상세 보기

- ❖ home.jsp 파일에 아이템 이름 출력하는 부분을 아이템 이름을 선택하면 detail.html 경로로 아이템의 아이디를 가지고 넘어가도록 수정

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ page session="false"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page isELIgnored="false"%>
<link rel="stylesheet" type="text/css" href="./css/style.css">
<html>
<head>
<title>상품 목록 화면</title>
</head>
<body>
  <div align="center" class="body">
    <h2>상품 목록 화면</h2>
    <table border="1">
      <tr class="header">
        <th align="center" width="80">상품ID</th>
        <th align="center" width="320">상품명</th>
        <th align="center" width="100">가격</th>
      </tr>
```

# 상품 상세 보기

- ❖ home.jsp 파일에 아이템 이름 출력하는 부분을 아이템 이름을 선택하면 detail.html 경로로 아이템의 아이디를 가지고 넘어가도록 수정

```
<c:forEach var="item" items="${ist}">
    <tr class="record">
        <td align="center">${item.item_Id}</td>
        <td align="left">
            <a href="detail.html?itemid=${item.itemid}">${item.itemname}</a>
            <td align="right">${item.price}원</td>
        </tr>
    </c:forEach>
</table>
</div>
</body>
</html>
```



# 상품 상세 보기

## ❖ item.xml 파일에 상세보기를 위한 SQL 생성

```
<select id="getItem" resultType="Item" parameterType="java.lang.Integer">  
    SELECT itemid, itemname, price, description, pictureurl FROM item where  
    itemid=#{itemid}  
</select>
```



# 상품 상세 보기

- ❖ ItemDao 클래스에 상세보기를 구현하기 위해서 상품 id를 받아서 데이터베이스에서 검색 한 후 검색 결과를 가지고 Item 객체를 생성해서 리턴하는 메서드를 구현

```
public Item getItem(Integer itemid) {  
    return sqlSession.selectOne("itemmapper.getItem", itemid);  
}
```



# 상품 상세 보기

- ❖ ItemService 인터페이스에 상세보기를 구현하기 위해서 상품 id를 받아서 데이터베이스에서 검색한 후 검색 결과를 가지고 Item 객체를 생성해서 리턴하는 메서드를 선언

```
public Item getItem(Integer itemid);
```



# 상품 상세 보기

- ❖ ItemServiceImpl 클래스에 상세보기를 구현하기 위해서 상품 id를 받아서 데이터베이스에서 검색 한 후 검색 결과를 가지고 Item 객체를 생성해서 리턴하는 메서드를 구현

```
@Override  
public Item getItem(Integer itemid) {  
    return itemDao.getItem(itemid);  
}
```

# 상품 상세 보기

## ❖ HomeController에 메서드 추가

```
@RequestMapping("/detail.html")
public String detail(@RequestParam("itemid") int itemid, Model model){
    Item item = itemService.getItem(itemid);
    //데이터를 저장
    model.addAttribute("item", item);
    return "item";
}
```

# 상품 상세 보기

## ❖ 상품 상세 보기를 출력할 item.jsp 파일을 view 디렉토리에 생성

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ page session="false"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head>
<title>상품 상세 화면</title>
<link rel="stylesheet" type="text/css" href="/css/style.css">
</head>
```

# 상품 상세 보기

## ❖ 상품 상세 보기를 출력할 item.jsp 파일을 view 디렉토리에 생성

```
<body>
  <div align="center" class="body">
    <h2>상품 상세 화면</h2>
    <table>
      <tr>
        <td></td>
        <td align="center">
          <table>
            <tr height="50">
              <td width="80">상품명</td>
              <td width="160">${item.itemname}</td>
            </tr>
            <tr height="50">
              <td width="80">가격</td>
              <td width="160">${item.price}원</td>
            </tr>
            <tr height="50">
              <td width="80">비고</td>
              <td width="160">${item.description}</td>
            </tr>
          </table>
        </td>
      </tr>
    </table>
  </div>
</body>
```

# 상품 상세 보기

- ❖ 상품 상세 보기를 출력할 item.jsp 파일을 view 디렉토리에 생성

```
align="center" width="240"><a href="./">■목록으로
    돌아가기</a></td>
</tr>
</table>
</div>
</body>
</html>
```

<tr>  
<td colspan="2">  
</td>  
</tr>  
</table>  
</td>  
</tr>  
</table>  
</div>  
</body>  
</html>



# 상품 상세 보기

- ❖ 프로젝트의 webapp 디렉토리에 img 디렉토리를 생성하고 이미지 파일 복사



# Mapper 인터페이스 이용

## ❖ Mapper 인터페이스를 생성 – 기본패키지.dao.ItemMapper

```
@Repository
public interface ItemMapper {
    @Select("select * from item")
    public List<Item> getAll();

    @Select("select * from item where itemid=#{itemid}")
    public Item getItem(Integer itemid);
}
```

# Mapper 인터페이스 이용

## ❖ ItemServiceImpl 클래스를 수정

```
@Service
public class ItemServiceImpl implements ItemService {
    @Autowired
    //private ItemDao itemDao;
    private ItemMapper itemDao;

    @Override
    @Transactional
    public List<Item> getAll() {
        return itemDao.getAll();
    }

    @Override
    @Transactional
    public Item getItem(Integer itemid) {
        return itemDao.getItem(itemid);
    }
}
```

# Mapper 인터페이스 이용

## ❖ root-context.xml 파일의 MyBatis bean 생성 수정

```
<!-- xml 파일을 사용하는 MyBatis를 이용할 때 사용하는 Bean -->
<!--
<bean id="sqlSessionFactory"
      class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation"
      value="classpath:mybatis/mybatis-config.xml" />
    <property name="mapperLocations"
      value="classpath:mybatis/mappers/**/*.xml" />
</bean>

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

# Mapper 인터페이스 이용

## ❖ root-context.xml 파일의 MyBatis bean 생성 수정

```
<!-- Mapper 인터페이스를 이용하는 MyBatis 설정 -->
<bean class="org.mybatis.spring.SqlSessionFactoryBean" id="sqlSessionFactory">
    <property name="dataSource" ref="dataSource" />
</bean>

<!-- Mapper 인터페이스 스캔 위치 설정 -->
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="kr.co.adamsoft.dao"/>
</bean>
```

# Mapper 인터페이스 이용

- ❖ ItemDao 클래스의 bean 이 자동으로 생성되지 않도록 수정

```
//@Repository  
public class ItemDao {
```



# Hibernate 활용

## ❖ pom.xml 파일에 Hibernate 사용을 위한 의존성 라이브러리를 추가

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-entitymanager</artifactId>
  <version>5.4.2.Final</version>
</dependency>

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

# Hibernate 활용

## ❖ 하이버네이트 설정 파일을 생성(기본패키지.dao 패키지에 item.hbm.xml)

```
<?xml version="1.0"?>
```

```
<!DOCTYPE hibernate-mapping PUBLIC
```

```
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
```

```
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
```

```
<hibernate-mapping package="kr.co.adamsoft.domain">
```

```
  <class name="Item" table="ITEM">
```

```
    <id name="itemid" column="ITEMID">
```

```
      </id>
```

```
    <property name="itemname" column="ITEMNAME" />
```

```
    <property name="pictureurl" column="PICTUREURL" />
```

```
    <property name="price" column="PRICE" />
```

```
    <property name="description" column="DESCRIPTION" />
```

```
  </class>
```

```
</hibernate-mapping>
```



# Hibernate 활용

## ❖ root-context.xml 파일에 하이버네이트 사용을 위한 객체 생성 코드 추가

```
<bean id="sessionFactory"
      class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="mappingResources">
        <list>
            <value>kr/co/adamsoft/dao/item.hbm.xml</value>
        </list>
    </property>
    <property name="hibernateProperties">
        <value>
            hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
        </value>
    </property>
</bean>
```

# Hibernate 활용

- ❖ **servlet-context.xml** 파일에 tx 네임스페이스를 추가하고 트랜잭션 사용을 위한 내용 추가

```
<tx:annotation-driven/>
```

```
<beans:bean id="transactionManager"  
    class="org.springframework.orm.hibernate5.HibernateTransactionManager">  
    <beans:property name="sessionFactory" ref="sessionFactory" />  
</beans:bean>
```

# Hibernate 활용

## ❖ ItemDao 클래스의 내용 수정

```
@Repository
public class ItemDao {
    @Autowired
    private SessionFactory sessionFactory;

    public List<Item> getAll() {
        CriteriaQuery<Item> criteriaQuery =
sessionFactory.getCurrentSession().getCriteriaBuilder().createQuery(Item.class);
        criteriaQuery.from(Item.class);
        List<Item> list =
sessionFactory.getCurrentSession().createQuery(criteriaQuery).getResultList();
        return list;
    }

    public Item getItem(Integer itemid) {
        Item item = (Item)sessionFactory.getCurrentSession().get(Item.class, itemid);
        return item;
    }
}
```



# Hibernate 활용

## ❖ ItemServiceImpl 클래스의 내용 수정

```
@Service
public class ItemServiceImpl implements ItemService {
    @Autowired
    private ItemDao itemDao;

    @Override
    @Transactional
    public List<Item> getAll() {
        return itemDao.getAll();
    }

    @Override
    @Transactional
    public Item getItem(Integer itemid) {
        return itemDao.getItem(itemid);
    }
}
```

# MySQL

## ❖ MySQL에 접속해서 샘플 데이터 생성

```
use cyberadam;
```

```
drop table item;
```

```
CREATE TABLE ITEM(  
    itemid int,  
    itemname VARCHAR(100),  
    price int,  
    description VARCHAR(200),  
    pictureurl VARCHAR(100),  
    PRIMARY KEY (itemid)  
)engine=innodb default charset=utf8;
```

```
insert into item values(1, 'Lemon', 500, 'Vitamin-A', 'lemon.jpg');  
insert into item values(2, 'Orange', 1500, 'Vitamin-B', 'orange.jpg');  
insert into item values(3, 'Kiwi', 2000, 'Vitamin-C', 'kiwi.jpg');  
insert into item values(4, 'Grape', 1000, 'Vitamin-D', 'grape.jpg');  
insert into item values(5, 'Strawberry', 2000, 'Vitamin-E', 'strawberry.jpg');  
insert into item values(6, 'Mandarin', 300, 'Vitamin-F', 'mandarin.jpg');
```

```
commit;
```

```
select * from ITEM;
```

# MySQL

## ❖ pom.xml 파일에 MySQL 의존성 설정

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->  
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <version>8.0.16</version>  
</dependency>
```

# MySQL

## ❖ root-context.xml 파일의 데이터베이스 접속 정보 수정

```
<bean id="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <property name="driverClassName">
        <value>com.mysql.cj.jdbc.Driver</value>
    </property>

    <property name="url">
<value>jdbc:mysql://localhost:3306/cyberadam?useUnicode=yes&characterEncoding=UTF-8</value>
    </property>

    <property name="username">
        <value>root</value>
    </property>

    <property name="password">
        <value>wnddkd</value>
    </property>
</bean>
```

# MySQL

## ❖ root-context.xml 파일의 하이버네이트 매핑 정보 수정

```
<bean id="sessionFactory"
      class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="mappingResources">
    <list>
      <value>kr/co/adamsoft/dao/item.hbm.xml</value>
    </list>
  </property>
  <property name="hibernateProperties">
    <value>
      hibernate.dialect=org.hibernate.dialect.MySQLDialect
    </value>
  </property>
</bean>
```



# Spring MVC(View)

# ViewResolver 설정

## ❖ 스프링의 뷰

- ✓ 스프링 컨트롤러는 뷰에 의존적이지 않음
- ✓ 컨트롤러에서는 뷰의 이름만 지정하며 출력되는 View 객체는 ViewResolver에 의해서 결정
- ✓ ViewResolver 인터페이스
  - ❑ View resolveViewName(String viewName, Locale locale): 매핑되는 View 객체를 리턴
- ✓ View 인터페이스
  - ❑ String getContentType(): 응답 결과의 콘텐츠 타입 리턴(ex - text/html)
  - ❑ void render(Map<String, ?>model, HttpServletRequest request, HttpServletResponse response) throws Exception: 화면에 출력될 결과 생성

# ViewResolver 설정

## ❖ ViewResolver의 구현 클래스

- ✓ InternalResourceViewResolver: jsp나 HTML과 같이 웹 애플리케이션의 내부 자원을 이용해서 응답 결과를 생성하는데 JSTL이 존재하는 경우 JstlView(스프링의 지역화 관련 설정이 JSTL 커스텀 태그에 적용) 객체를 리턴
- ✓ BeanNameViewResolver: View 이름과 동일한 이름을 갖는 빈 객체를 View 객체로 사용하는데 커스텀 View 클래스를 뷰로 사용할 때 이용
- ✓ ResourceBundleViewResolver: View 이름과 View 객체 간의 매핑 정보를 저장하기 위해 자원을 이용
- ✓ XmlViewResolver: View 이름과 View 객체 간의 매핑 정보를 저장하기 위해서 xml 파일을 이용

# ViewResolver 설정

## ❖ InternalResourceViewResolver 설정

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

=>prefix 나 postfix는 생략이 가능

## ❖ BeanNameViewResolver 설정 - 직접 생성한 view.DownloadView 클래스의 객체를 사용하는 경우

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.BeanNameViewResolver ">
</bean>
<bean id="download"
      class="view.DownloadView ">
</bean>
```

# ViewResolver 설정

- ❖ XmlViewResolver 설정 – 별도의 xml 파일로부터 뷰 객체를 선택

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="location" value="/WEB-INF/view.xml" />
</bean>
```

- ❖ ResourceBundleViewResolver 설정

별도의 프로퍼티 파일로부터 뷰 객체를 선택

```
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.BeanResourceBundleViewResolver ">
    <property name="basename" value="/WEB-INF/view" />
</bean>
```

- ❖ 하나의 dispatcher-servlet 파일에 2개 이상의 ViewResolver를 설정할 수 있는데 이 때는 각각의 ViewResolver 빈에 order 프로퍼티를 이용해서 우선 순위를 설정하면 되는데 order의 값이 작을 수록 우선 순위가 높고 여러 개의 ViewResolver를 사용하는 경우 InternalResourceViewResolver는 가장 마지막 우선순위를 갖도록 해야 함

# 파일 다운로드 뷰

## 상품 목록 화면

| 상품ID | 상품명                        | 가격    |
|------|----------------------------|-------|
| 1    | <a href="#">Lemon</a>      | 500원  |
| 2    | <a href="#">Orange</a>     | 1500원 |
| 3    | <a href="#">Kiwi</a>       | 2000원 |
| 4    | <a href="#">Grape</a>      | 1000원 |
| 5    | <a href="#">Strawberry</a> | 2000원 |
| 6    | <a href="#">Mandarin</a>   | 300원  |

파일 목록보기

# 파일 다운로드 뷰

❖ **home.jsp** 파일을 수정 – **table** 태그 아래에 추가

```
<ul>
```

```
  <li> <a href="fileview" class="menu">파일 목록보기 </a> </li>
```

```
</ul>
```



# 파일 다운로드 뷰

## ❖ style.css 파일에 추가

```
.menu{  
    display: block;  
    width: 10em;  
    margin: 3px 5px;  
    padding: .5em 1em;  
    text-decoration: none;  
    text-align: center;  
    color: #fff;  
    background-color: rgb(190, 190, 190);  
    border: 2px solid rgb(175, 175, 175);  
    border-radius: 6px;  
    text-shadow: #666 .1em .1em .1em;  
    box-shadow: 0 5px 3px rgba(0, 0, 0, .5);  
    position: relative;
```



# 파일 다운로드 뷰

## ❖ style.css 파일에 추가

```
-webkit-transition: background-color 0.2s, border-color 0.2s, top .2s,  
                    box-shadow 0.2s;  
-moz-transition: background-color 0.2s, border-color 0.2s, top .2s,  
                 box-shadow 0.2s;  
-o-transition: background-color 0.2s, border-color 0.2s, top .2s,  
                box-shadow 0.2s;  
-ms-transition: background-color 0.2s, border-color 0.2s, top .2s,  
                 box-shadow 0.2s;  
transition: background-color 0.2s, border-color 0.2s, top .2s,  
            box-shadow 0.2s;
```

```
}
```

# 파일 다운로드 뷰

## ❖ style.css 파일에 추가

```
ul {  
    list-style-type: none;  
    margin: 0;  
    padding: 0;  
}
```

```
li {  
    margin-bottom: 3em;  
}
```

/\* a 태그에 포커스가 오거나 마우스가 올라오면 적용  
: 다음에 나오는 것은 가상 선택자, 가상 클래스 \*/

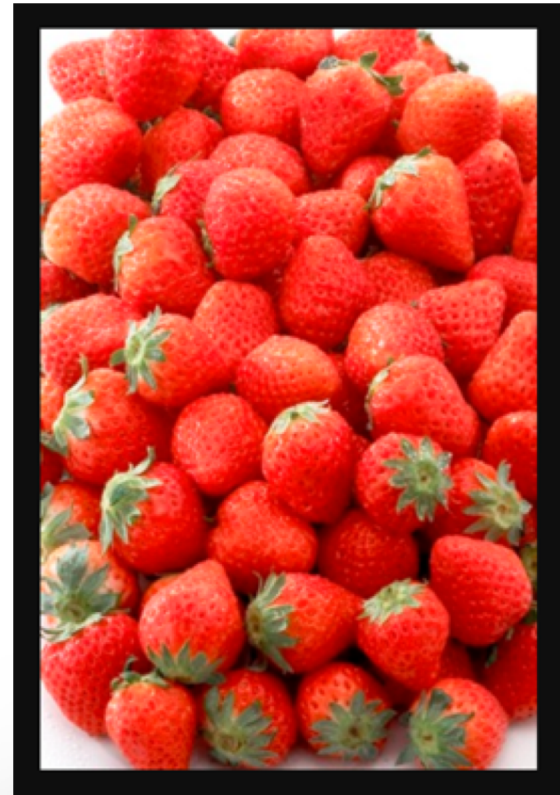
```
a:hover, a:focus {  
    background-color: #fdca00;  
    border-color: #fda700;  
}
```

```
a:active {  
    top: 3px;  
    box-shadow: 0 1px 2px rgba(0, 0, 0, .5);  
}
```

# 파일 다운로드 뷰

## 파일 목록 화면

| 파일명                            |
|--------------------------------|
| <a href="#">Thumbs.db</a>      |
| <a href="#">kiui.jpg</a>       |
| <a href="#">grape.jpg</a>      |
| <a href="#">mandarin.jpg</a>   |
| <a href="#">orange.jpg</a>     |
| <a href="#">lemon.jpg</a>      |
| <a href="#">strawberry.jpg</a> |



# 파일 다운로드 뷰

- ❖ 파일 목록을 문자열의 List로 만들어서 리턴해주는 메서드를 소유한 ViewService 인터페이스를 기본 패키지.service 패키지에 생성

```
import java.util.List;
```

```
import javax.servlet.http.HttpServletRequest;
```

```
public interface ViewService {  
    public List<String> fileview(HttpServletRequest request);  
}
```

# 파일 다운로드 뷰

- ❖ 프로젝트의 img 디렉토리의 파일 목록을 문자열의 List로 만들어서 리턴해주는 메서드를 구현한 ViewServiceImpl 클래스를 기본 패키지.service 패키지에 생성

@Service

```
public class ViewServiceImpl implements ViewService {
```

```
    public List<String> fileview(HttpServletRequest request){
        String path = request.getServletContext().getRealPath("/img");
        File f = new File(path);
        String [] s = f.list();
        List <String> list = new ArrayList<String>();
        for(String sub : s){
            if(sub.indexOf('.') != -1)
                list.add(sub);
        }
        return list;
    }
```

```
}
```

# 파일 다운로드 뷰

- ❖ 사용자의 요청을 처리하는 HomeController 클래스에 서비스 객체를 주입받는 코드를 추가하고 fileview 요청을 처리하는 메서드를 생성

```
@Autowired
private ViewService viewService;

@RequestMapping("/fileview")
public String fileview(Model model, HttpServletRequest request){
    List<String>list = viewService.fileview(request);
    model.addAttribute("list", list);
    return "fileview";
}
```

# 파일 다운로드 뷰

## ❖ view 디렉토리에 fileview.jsp 파일을 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ page isELIgnored="false"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>파일 목록</title>
</head>
```

# 파일 다운로드 뷰

## ❖ view 디렉토리에 fileview.jsp 파일을 생성

```
<body>
  <div align="center" class="body">
    <h2>파일 목록 화면</h2>
    <table border="1">
      <tr>
        <th width="200">파일명</th>
      </tr>

      <c:forEach var="item" items="${list}">
        <tr>
          <td><a
            href="${pageContext.request.contextPath}/img/${item}">${item}</a></td>
        </tr>
      </c:forEach>
    </table>
  </div>
</body>
</html>
```



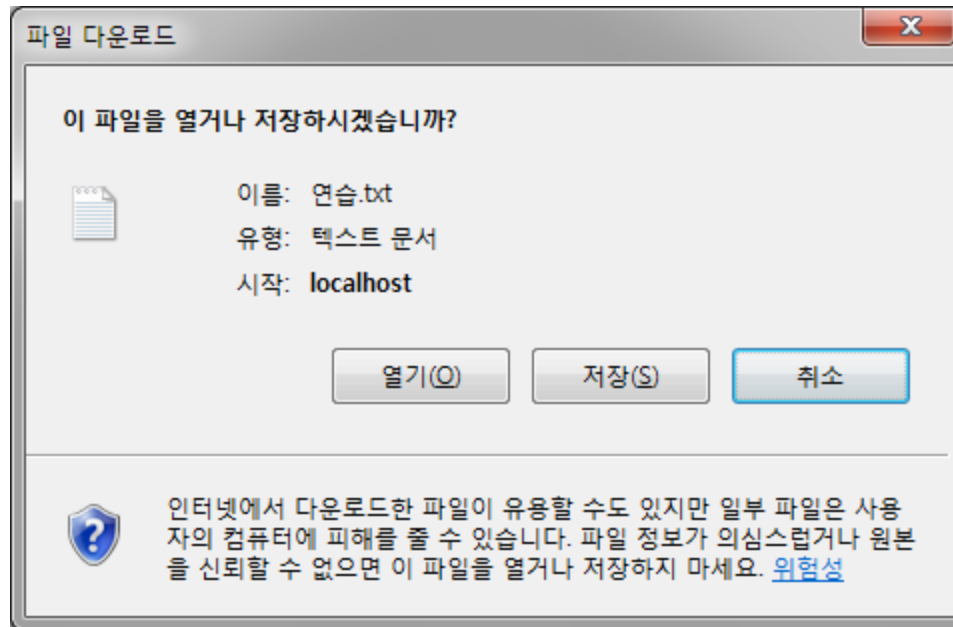
# 파일 다운로드 뷰

## ❖ 파일 다운로드

- ✓ 파일 다운로드를 구현할 때는 파일의 위치를 a 태그의 링크로 설정하면 되는데 이 때 브라우저가 출력할 수 있는 파일은 브라우저가 출력
- ✓ 브라우저가 파일을 출력하지 않고 다운로드 화면을 보여주게 하려면 출력되는 뷰의 ContentType을 application/octet-stream 이나 application/download로 설정해야 함
- ✓ 파일의 크기도 ContentLength 속성에 지정
- ✓ 파일 이름은 Header의 Content-Disposition에 적절하게 설정해야 하며 encoding 방식은 binary로 설정
- ✓ 위의 설정을 전부 한 후 response 객체의 OutputStream에 파일의 내용을 기록



# 파일 다운로드 뷰



# 파일 다운로드 뷰

## ❖ fileview.jsp 파일에서 파일의 링크를 수정

```
<td> <a href="download?filename=${item}">${item}</a> </td>
```



# 파일 다운로드 뷰

## ❖ HomeController 클래스에 download을 처리할 메서드를 생성

```
@RequestMapping("/download")
public String download(@RequestParam("filename") String filename, Model model,
    HttpServletRequest request){
    File downloadFile = new File(request.getServletContext().getRealPath("/img") + "/" +
    filename);
    model.addAttribute("downloadFile", downloadFile);
    return "download";
}
```

# 파일 다운로드 뷰

- ❖ 기본 패키지.view 패키지에 파일 다운로드를 처리해 줄 **AbstractView** 클래스를 상속받은 **DownloadView** 클래스를 생성

```
public class DownloadView extends AbstractView {
    public DownloadView() {
        setContentType("application/download; charset=utf-8");
    }

    @Override
    protected void renderMergedOutputModel(Map<String, Object> model,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {
        File file = (File) model.get("downloadFile");
        response.setContentType(getContentType());
        response.setContentLength((int) file.length());

        String userAgent = request.getHeader("User-Agent");
        boolean ie = userAgent.indexOf("rv") > -1;
        String fileName = null;
        if (ie) {
            fileName = URLEncoder.encode(file.getName(), "utf-8");
        } else {
            fileName = new String(file.getName().getBytes("utf-8"),
                "iso-8859-1");
        }
    }
}
```

# 파일 다운로드 뷰

- ❖ 기본 패키지.view 패키지에 파일 다운로드를 처리해 줄 **AbstractView** 클래스를 상속받은 **DownloadView** 클래스를 생성

```
response.setHeader("Content-Disposition", "attachment; filename=W"
                    + fileName + "W");
response.setHeader("Content-Transfer-Encoding", "binary");
```

```
OutputStream out = response.getOutputStream();
FileInputStream fis = null;
```

```
try {
    fis = new FileInputStream(file);
    FileCopyUtils.copy(fis, out);
} finally {
    if (fis != null)
        try {
            fis.close();
        } catch (IOException ex) {
        }
}
```

```
out.flush();
```

```
}
```

```
};
```

# 파일 다운로드 뷰

## ❖ servlet-context.xml 파일의 내용 수정

```
<beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
    <beans:property name="prefix" value="/view/" />  
    <beans:property name="suffix" value=".jsp" />  
    <beans:property name="order" value="1" />  
</beans:bean>
```

```
<beans:bean class="org.springframework.web.servlet.view.BeanNameViewResolver">  
    <beans:property name="order" value="0" />  
</beans:bean>
```

```
<beans:bean id="download" class="기본 패키지.view.DownloadView" />
```

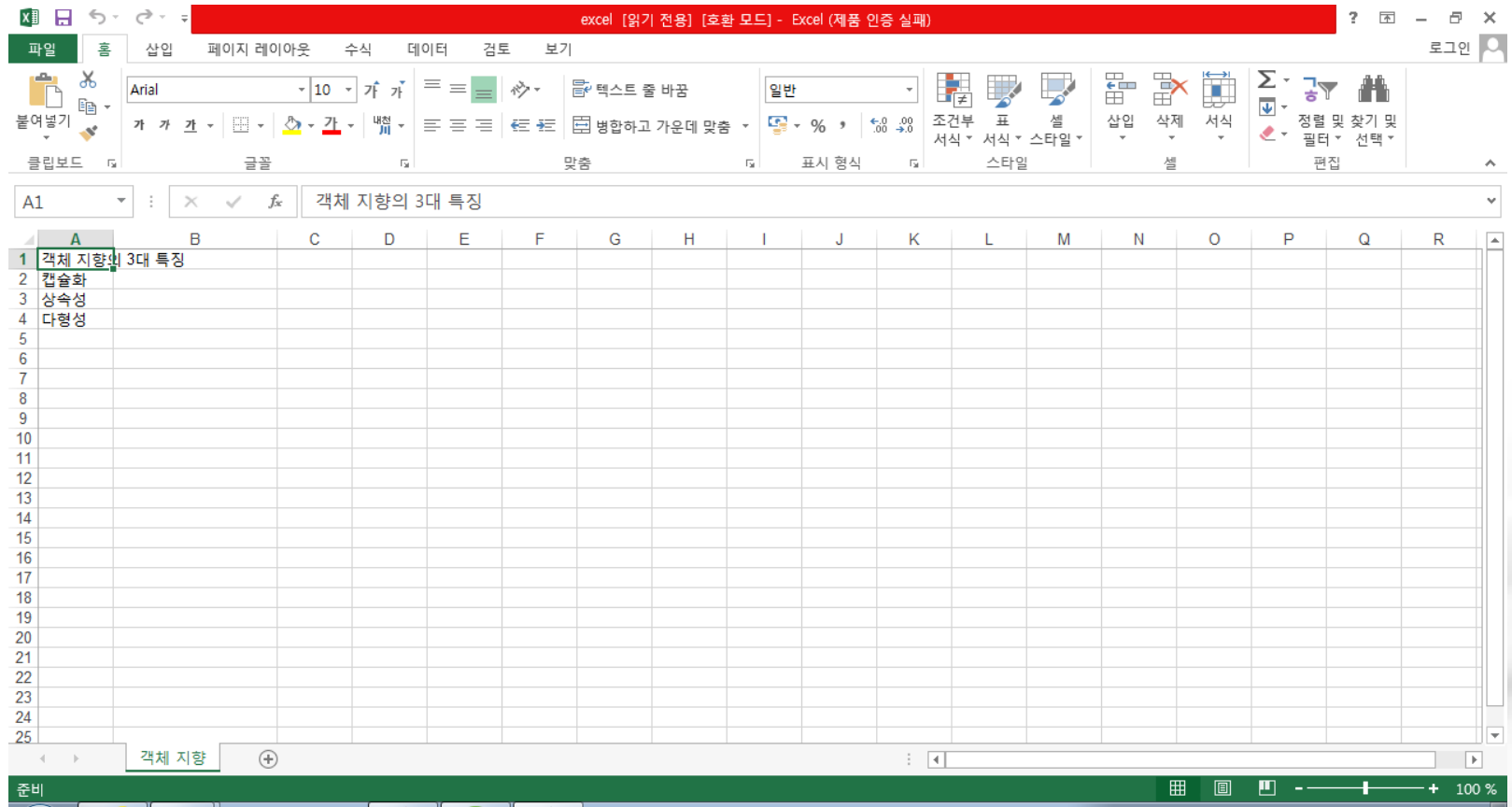
# Excel View

## ❖ 엑셀 파일 출력

- ✓ 엑셀 파일 뷰를 만들기 위해서는 apache poi 라이브러리를 이용
- ✓ 엑셀 파일로 출력 결과를 만들고자 하는 경우에는 AbstractXlsView 클래스로부터 상속받은 View 클래스를 이용
- ✓ AbstractExcelView 클래스는 poi.jar 파일에 포함되어 있음
- ✓ 이 파일을 이용하면 엑셀 파일을 읽을 수 있고 엑셀 파일로 내보낼 수 있음
- ✓ AbstractExcelView 클래스로부터 상속받는 클래스를 만들고 메서드를 재정의 해야 함
  - protected void buildExcelDocument(Map<String, Object> model, Workbook workbook, HttpServletRequest request, HttpServletResponse response) throws Exception
  - ❑ **Model**은 넘어온 데이터
  - ❑ **Workbook**을 이용해서 엑셀파일을 생성
  - ❑ **Workbook** 안에는 **Sheet** 가 있고 시트 안에는 **Row**가 있고 row 안에 다시 **Cell**이 있음



# Excel View



# Excel View

- ❖ **poi.jar** 파일을 사용하기 위한 **dependency**를 **pom.xml** 파일에 추가

```
<dependency>  
  <groupId>org.apache.poi</groupId>  
  <artifactId>poi</artifactId>  
  <version>3.16</version>  
</dependency>
```



# Excel View

## ❖ home.jsp 파일에 엑셀 파일 링크 추가

```
<%@ page language= " java " contentType= " text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>여러 가지 뷰</title>
</head>
<body>
    <ul>
        <li><a href="fileview" class="menu">파일 목록보기</a></li>
        <li><a href="item.xls" class="menu">엑셀 다운로드</a></li>
    </ul>
</body>
</html>
```

# Excel View

## ❖ HomeController에 excel.xls 처리를 위한 코드 추가

```
@RequestMapping("/item.xls")
public String excel(Model model) {
    List<Item> list = itemService.getAll();
    model.addAttribute("list", list);
    return "excel";
}
```



# Excel View

- ❖ 기본 패키지.view 패키지에 **AbstractExcelView**로 부터 상속받는 **ExcelView** 클래스 생성

```
public class ExcelView extends AbstractXlsView {  
  
    @Override  
    protected void buildExcelDocument(Map<String, Object> model, Workbook workbook,  
    HttpServletRequest request,  
                                   HttpServletResponse response) throws Exception {  
        Sheet sheet = createFirstSheet(workbook);  
        createColumnLabel(sheet);  
  
        List<Item> list = (List<Item>)model.get("list");  
        int rowNum = 1;  
        for (Item item : list) {  
            createRow(sheet, item, rowNum++, workbook);  
        }  
    }  
}
```

# Excel View

- ❖ 기본 패키지.view 패키지에 **AbstractExcelView**로 부터 상속받는 **ExcelView** 클래스 생성

```
private void createColumnLabel(Sheet sheet) {  
    Row firstRow = sheet.createRow(0);  
    Cell cell = firstRow.createCell(0);  
    cell.setCellValue("상품명");  
  
    cell = firstRow.createCell(1);  
    cell.setCellValue("설명");  
  
    cell = firstRow.createCell(2);  
    cell.setCellValue("가격");  
}  
  
private Sheet createFirstSheet(Workbook workbook) {  
    Sheet sheet = workbook.createSheet();  
    workbook.setSheetName(0, "ITEM");  
    sheet.setColumnWidth(1, 256 * 20);  
    return sheet;  
}
```

# Excel View

- ❖ 기본 패키지.view 패키지에 **AbstractExcelView**로 부터 상속받는 **ExcelView** 클래스 생성

```
private void createRow(Sheet sheet, Item item, int rowNum, Workbook workbook) {  
    Row row = sheet.createRow(rowNum);
```

```
    Cell cell = row.createCell(0);  
    cell.setCellValue(item.getItemname());
```

```
    cell = row.createCell(1);  
    cell.setCellValue(item.getDescription());
```

```
    cell = row.createCell(2);  
    cell.setCellValue(item.getPrice() + "원");
```

```
}
```

```
}
```

# Excel View

## ❖ **servlet-context.xml** 파일에 추가

```
<beans:bean id="excel" class="기본 패키지.view.ExcelView" />
```





# Excel Read

## ❖ 엑셀 파일을 읽기 위한 라이브러리 의존성

```
<dependency>  
  <groupId>org.apache.poi</groupId>  
  <artifactId>poi-ooxml</artifactId>  
  <version>3.13</version>  
</dependency>
```

## ❖ 엑셀 파일을 읽는 방법

### ✓ 엑셀 파일을 Workbook으로 만들기

```
new XSSFWorkbook(new FileInputStream(File 객체));
```

### ✓ 시트 읽어오기

Workbook에서 getSheetAt(시트번호)로 읽어냄

Sheet는 Row의 Collection

### ✓ Row는 Cell의 집합으로 getCell(셀번호)를 이용해서 Cell을 읽어옴

# Excel Read

test - Excel (제품 인증 실패)

파일 홈 삽입 페이지 레이아웃 수식 데이터 검토 보기 추가 기능 부하 테스트 팀 로그인

붙여넣기 글꼴 글꼴 크기 글꼴 색 글꼴 밑줄 글꼴 배경색

클립보드 글꼴

맞춤 표시 형식 스타일

일반 조건부 서식 삽입 삭제 서식

표 서식 셀 스타일

정렬 및 찾기 및 필터 선택 편집

C5

|    | A   | B  | C  | D | E | F | G | H | I | J | K | L | M |
|----|-----|----|----|---|---|---|---|---|---|---|---|---|---|
| 1  | 이름  | 나이 | 성별 |   |   |   |   |   |   |   |   |   |   |
| 2  | 유관순 | 35 | 여자 |   |   |   |   |   |   |   |   |   |   |
| 3  | 윤봉길 | 54 | 남자 |   |   |   |   |   |   |   |   |   |   |
| 4  |     |    |    |   |   |   |   |   |   |   |   |   |   |
| 5  |     |    |    |   |   |   |   |   |   |   |   |   |   |
| 6  |     |    |    |   |   |   |   |   |   |   |   |   |   |
| 7  |     |    |    |   |   |   |   |   |   |   |   |   |   |
| 8  |     |    |    |   |   |   |   |   |   |   |   |   |   |
| 9  |     |    |    |   |   |   |   |   |   |   |   |   |   |
| 10 |     |    |    |   |   |   |   |   |   |   |   |   |   |
| 11 |     |    |    |   |   |   |   |   |   |   |   |   |   |
| 12 |     |    |    |   |   |   |   |   |   |   |   |   |   |
| 13 |     |    |    |   |   |   |   |   |   |   |   |   |   |

Sheet1

준비

100 %

# Excel Read

## ❖ pom.xml 파일에 추가

```
<dependency>  
  <groupId>org.apache.poi</groupId>  
  <artifactId>poi-ooxml</artifactId>  
  <version>3.13</version>  
</dependency>
```

# Excel Read

- ❖ Excel 파일을 프로젝트의 webapp(WebContent) 디렉토리에 복사 – items.xls
- ❖ home.jsp 파일에 엑셀파일 읽기 링크 설정  
<li> <a href="excelread" class="menu">엑셀 파일읽기 </a> </li>

# Excel Read

- ❖ **ViewService** 인터페이스에 엑셀파일을 읽어서 **Map**의 **List**로 리턴하는 메서드를 선언  
`public List<Map<String, Object>> excelRead(HttpServletRequest request);`



# Excel Read

- ❖ **ViewServiceImpl** 클래스에 엑셀파일을 읽어서 **Map**의 **List**로 리턴하는 메서드를 구현

```
public List<Map<String, Object>> excelRead(HttpServletRequest request) {  
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();  
    System.out.println(list);  
    try {  
        // 엑셀파일 경로 설정  
        File file = new File(request.getServletContext().getRealPath("/") + "/item.xls");  
        // 엑셀 파일 오픈  
        HSSFWorkbook wb = new HSSFWorkbook(new FileInputStream(file));
```

# Excel Read

## ❖ ViewServiceImpl 클래스에 엑셀파일을 읽어서 Map의 List로 리턴하는 메서드를 구현

```
for (Row row : wb.getSheetAt(0)) {  
    if (row.getRowNum() > 0) {  
        Map<String, Object> map = new HashMap<String,  
Object>();  
  
        map.put("itemname", row.getCell(0));  
        map.put("description", row.getCell(1));  
        map.put("price", row.getCell(2));  
        System.out.println(map);  
        list.add(map);  
    }  
}  
wb.close();  
} catch (Exception e) {  
    System.out.println(e.getMessage());  
    e.printStackTrace();  
}  
return list;  
}
```

# Excel Read

- ❖ **HomeController** 클래스에서 엑셀파일 읽기 요청이 왔을 때 호출되는 메서드를 작성

```
@RequestMapping("/excelread")  
public String excelread(Model model, HttpServletRequest request) {  
    List<Map<String, Object>> list = viewService.excelRead(request);  
    model.addAttribute("list", list);  
    return "excelread";  
}
```



# Excel Read

## ❖ 엑셀파일 읽기 요청의 결과를 출력할 excelread.jsp 파일을 view 디렉토리에 생성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>엑셀 파일 읽기</title>
</head>
```

# Excel Read

- ❖ 엑셀파일 읽기 요청의 결과를 출력할 excelread.jsp 파일을 view 디렉토리에 생성

```
<body>
  <table align="center" border="1">
    <tr>
      <th>이름</th>
      <th>설명</th>
      <th>가격</th>
    </tr>
    <c:forEach var="item" items="${list}">
      <tr>
        <th>${item.itemname}</th>
        <th>${item.description}</th>
        <th>${item.price}</th>
      </tr>
    </c:forEach>
  </table>
</body>
</html>
```

# PDF

- ❖ iText API를 이용하면 PDF 파일을 생성할 수 있는 AbstractPdfView 클래스를 제공
- ❖ <http://itextpdf.com/product>  
protected void buildPdfDocument(Map<String, Object> model, Document document, PdfWriter writer, HttpServletRequest request, HttpServletResponse response) throws Exception 메서드 재정의
  - ✓ model은 넘어온 데이터
  - ✓ document가 실제 생성할 문서
  - ✓ writer는 pdf 문서를 만들 때 사용할 수 있는 객체로 body에 직접 pdf 객체를 삽입할 때 사용



# PDF

| 이름         | 설명        | 가격    |
|------------|-----------|-------|
| Lemon      | Vitamin-A | 500원  |
| Orange     | Vitamin-B | 1500원 |
| Kiwi       | Vitamin-C | 2000원 |
| Grape      | Vitamin-D | 1000원 |
| Strawberry | Vitamin-E | 2000원 |
| Mandarin   | Vitamin-F | 300원  |

# PDF

- ❖ iText.jar 파일을 사용할 수 있도록 pom.xml 파일에 추가

```
<dependency>  
  <groupId>com.lowagie</groupId>  
  <artifactId>itext</artifactId>  
  <version>2.1.7</version>  
</dependency>
```

# PDF

- ❖ **malgun.ttf** 파일을 프로젝트의 **webapp** 디렉토리에 복사
- ❖ **home.jsp** 파일을 수정  
`<li> <a href="pdf" class="menu">PDF 다운로드 </a> </li>`

# PDF

## ❖ pdf 요청을 처리할 코드를 HomeController에 추가

```
@RequestMapping("/pdf")  
public String pdfReport(Model model) {  
    List<Item> list = itemService.getAll();  
    model.addAttribute("list", list);  
    return "pdf";  
}
```



# PDF

## ❖ 기본 패키지.view 패키지에 AbstractPdfView 클래스로부터 상속받는 PdfView 클래스 생성

```
public class PdfView extends AbstractPdfView {  
    @SuppressWarnings("unchecked")  
    @Override  
    protected void buildPdfDocument(Map<String, Object> model, Document document,  
    PdfWriter writer,HttpServletRequest request, HttpServletResponse response) throws Exception  
    {  
        List<Item> list = (List<Item>) model.get("list");  
        Table table = new Table(3, list.size() + 1);  
        table.setPadding(5);  
  
        BaseFont bfKorean =  
        BaseFont.createFont(request.getServletContext().getRealPath("/") + "/malgun.ttf",  
        BaseFont.IDENTITY_H,BaseFont.EMBEDDED);  
        Font font = new Font(bfKorean);
```



# PDF

## ❖ 기본 패키지.view 패키지에 **AbstractPdfView** 클래스로부터 상속받는 PdfView 클래스 생성

```
Cell cell = new Cell(new Paragraph("이름", font));  
cell.setHeader(true);  
table.addCell(cell);
```

```
cell = new Cell(new Paragraph("설명", font));  
cell.setHeader(true);  
table.addCell(cell);
```

```
cell = new Cell(new Paragraph("가격", font));  
cell.setHeader(true);  
table.addCell(cell);
```

```
table.endHeaders();
```

# PDF

## ❖ 기본 패키지.view 패키지에 AbstractPdfView 클래스로부터 상속받는 PdfView 클래스 생성

```
for (Item item : list) {  
    Cell imsi = new Cell(new Paragraph(item.getItemname(), font));  
    table.addCell(imsi);  
  
    imsi = new Cell(new Paragraph(item.getDescription(), font));  
    table.addCell(imsi);  
  
    imsi = new Cell(new Paragraph(item.getPrice() + "원", font));  
    table.addCell(imsi);  
  
}  
document.add(table);  
}
```

# PDF

## ❖ servlet-context.xml 파일에 추가

```
<beans:bean id="pdf" class="kr.co.adamsoft.view.PdfView" />
```

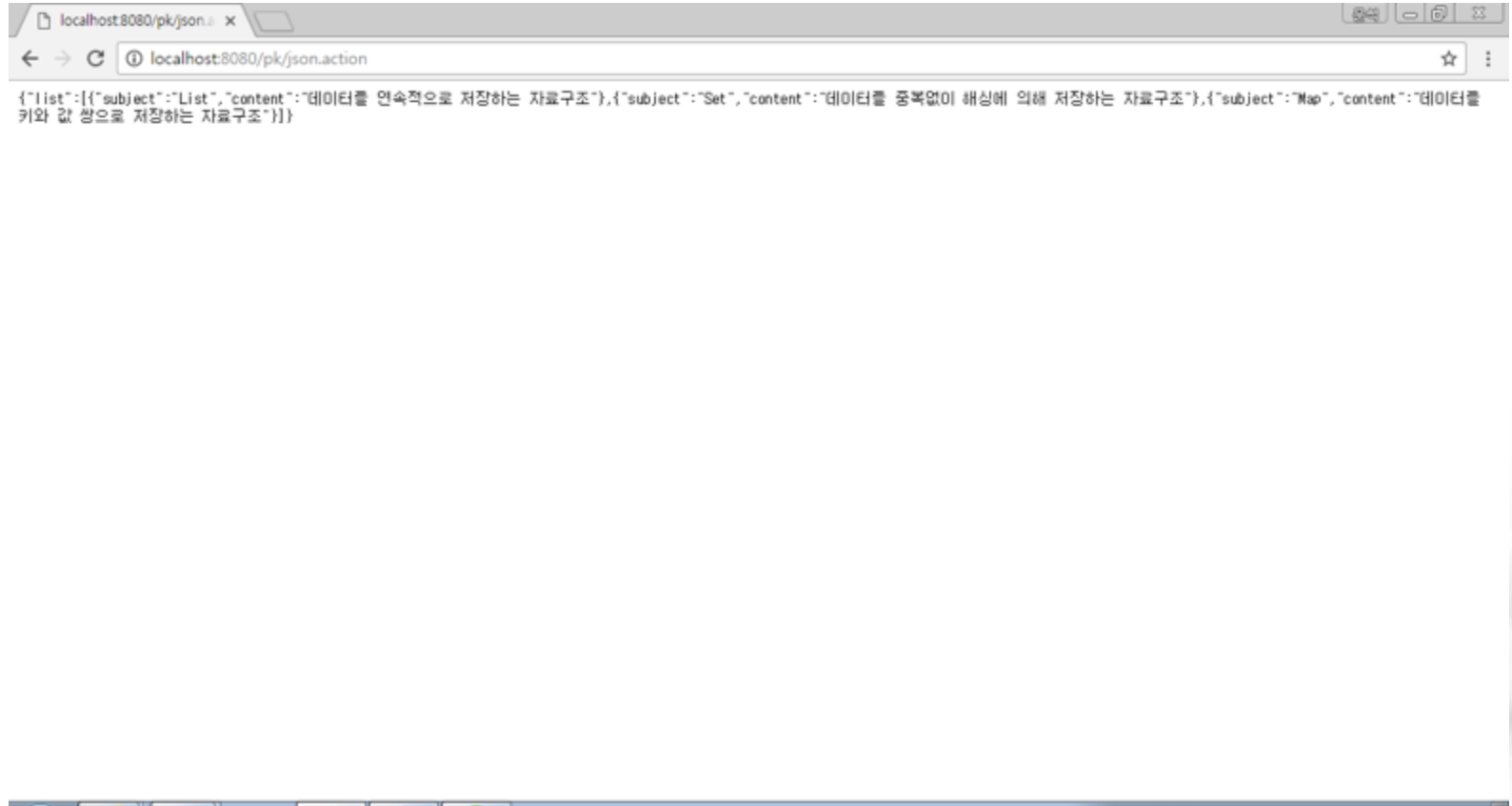


# JSON

## ❖ JSON 출력

- ✓ Controller 클래스의 요청 처리 메서드의 리턴 타입을 ResponseEntity로 설정하고 Content-type을 json으로 설정해서 리턴
- ✓ jackson-databind 라이브러리 이용하는 방법
  - ❑ 요청 처리 메서드의 리턴 타입을 객체로 설정하고 그 앞에 @ResponseBody Annotation을 추가
  - ❑ JSONObject 나 JSONArray 타입을 넘겨준 후 `org.springframework.web.servlet.view.json.MappingJacksonJsonView` 에서 출력
  - ❑ 4.x 버전에서는 @RestController를 이용해서 가능

# json



# json

❖ json 라이브러리 사용을 위한 코드를 pom.xml 파일에 추가

```
<dependency>  
  <groupId>com.fasterxml.jackson.core</groupId>  
  <artifactId>jackson-databind</artifactId>  
  <version>2.9.7</version>  
</dependency>
```



# json

## ❖ home.jsp 파일에 json 링크 추가

```
<li> <a href="itemlist.json" class="menu">아이템 목록</a> </li>
```



# json

## ❖ HomeController 클래스에 추가

```
@RequestMapping("/itemlist.json")  
public String jsonReport(Model model) {  
    List<Item> list = itemService.getAll();  
    model.addAttribute("list", list);  
    return "itemlist";  
}
```



# json

## ❖ servlet-context.xml 파일에 추가

- ✓ 스프링 3.0에서의 설정

```
<beans:bean id="itemlist"  
  class="org.springframework.web.servlet.view.json.MappingJacksonJsonView" />
```

- ✓ 스프링 4.0 이상에서의 설정

```
<beans:bean id="itemlist"  
  class="org.springframework.web.servlet.view.json.MappingJackson2JsonView" />
```



# RestController

- ❖ REST(Representational State Transfer)
  - ✓ 하나의 URI는 하나의 서비스를 대표하도록 설계
  - ✓ 서버에 접근하는 기기의 종류에 상관없이 처리할 수 있도록 해주는 개념
  - ✓ URI에 요청을 했을 때 출력하는 뷰를 만드는 것이 아니고 데이터를 넘기는 개념
  - ✓ OPEN API에 많이 사용하는데 이러한 서비스를 Restful 또는 RESTafrians 이라고 함
- ❖ RestController
  - ✓ Spring 4.0 버전 부터는 RestController 를 이용해서 Restful 서비스 제공이 가능
  - ✓ 결과 데이터만을 서버에서 제공하는 방식
  - ✓ Android 나 iPhone과 같은 모바일 환경에서의 서버의 데이터를 이용하는 경우나 Web Page에서 Ajax를 이용하는 경우에 사용
  - ✓ 이 클래스의 요청 처리 메서드에서 문자열 이나 Wrapper Class 의 객체를 리턴하면 텍스트 형태로 출력되고 Map 이나 DTO 또는 배열이나 List 객체를 리턴하면 그 객체를 json 형식으로 변환해서 출력
  - ✓ json 라이브러리를 추가해야 함

# RestController

## ❖ home.jsp 파일에 추가

<ul>

```
<li> <a href="fileview" class="menu">파일 목록보기 </a> </li>
<li> <a href="item.xls" class="menu">엑셀 다운로드 </a> </li>
<li> <a href="excelread" class="menu">엑셀 파일읽기 </a> </li>
<li> <a href="pdf" class="menu">PDF 다운로드 </a> </li>
<li> <a href="itemlist.json" class="menu">아이템 목록 </a> </li>
```

```
<li> <a href="rest/text" class="menu">텍스트 출력 </a> </li>
<li> <a href="rest/json" class="menu">JSON 출력 </a> </li>
<li> <a href="#" class="menu" onclick="ajax();" >ajax 출력 </a> </li>
```

</ul>

<div id="disp"> </div>

# RestController

## ❖ 프로젝트에 JSONController 클래스를 추가

```
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;
```

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

```
@RestController  
public class JSONController {  
    @RequestMapping("rest/text")  
    public String text() {  
        return "DataFormat : csv, xml, json";  
    }  
}
```

# RestController

## ❖ 프로젝트에 JSONController 클래스를 추가

```
@RequestMapping("rest/json")
public List<Map<String, Object>> json(){
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
    Map<String, Object>map1 = new HashMap<String, Object>();
    map1.put("csv", "구분자로 구분한 문자열");
    Map<String, Object>map2 = new HashMap<String, Object>();
    map2.put("xml", "태그 형식으로 데이터를 표현하는 방법");
    Map<String, Object>map3 = new HashMap<String, Object>();
    map3.put("json", "자바스크립트 객체를 표현하는 방법");
    list.add(map1);
    list.add(map2);
    list.add(map3);
    return list;
}
```

# RestController

## ❖ 프로젝트에 JSONController 클래스를 추가

```
@RequestMapping("rest/ajax")
public List<Map<String, Object>> ajax(){
    List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
    Map<String, Object>map1 = new HashMap<String, Object>();
    map1.put("title", "반응형 웹 디자인");
    map1.put("content", "하나의 뷰를 여러 디바이스에 맞게 보여주는 것 - CSS
이용");
    Map<String, Object>map2 = new HashMap<String, Object>();
    map2.put("title", "크로스 브라우징");
    map2.put("content", "모든 브라우저에서 동일한 콘텐츠를 사용할 수 있도록
하는 것 - jQuery 이용");
    Map<String, Object>map3 = new HashMap<String, Object>();
    map3.put("title", "웹 접근성");
    map3.put("content", "누구나 동일한 콘텐츠를 사용할 수 있도록 하는 것");
    list.add(map1);
    list.add(map2);
    list.add(map3);
    return list;
}
```

# RestController

## ❖ style.css 파일에 추가

```
span
{
    margin:10px;
    border:3px Solid Green;
    border-radius:10px;
    float:left;
    width:300px;
    height:150px;
    text-align:center;
}
h3:hover
{
    color:Red;
}
```



# RestController

## ❖ home.jsp 파일에 스크립트 추가

```
<script>
    function ajax(){
        var request = new XMLHttpRequest();
        request.open('get', 'rest/ajax');
        request.send('');
        request.addEventListener('load', function(data){

            var json = data.target.responseText;
            var list = JSON.parse(json);

            var output = "";
            for(i in list){
                var item = list[i];
                output += '<span>';
                output += '<h3>' + item.title + '</h3>';
                output += '<p>' + item.content + '</p>';
                output += '</span>';
            }
            document.getElementById('disp').innerHTML = output;
        });
    };
</script>
```



# XML

- ❖ `org.springframework.web.servlet.view.xml.MarshallingView`를 이용하면 자바 객체를 xml 형식으로 출력이 가능
  - ✓ 위의 클래스는 View이고 실제로 xml로 변환해주는 클래스는 별도로 지정
- ❖ `org.springframework.xml.jaxb.Jaxb2Marshaller` 클래스를 이용하면 자바 객체를 xml로 변환이 가능
  - ✓ 이 클래스의 객체를 `MarshallingView`의 생성자나 `marshaller` 프로퍼티에 전달
  - ✓ `Jaxb2Marshaller` 클래스의 `classesToBeBound` 프로퍼티에 List 타입으로 실제 변환할 클래스 이름을 전달
- ❖ Xml 변환을 위해서는 여러가지 annotation을 사용
  - ✓ `@XmlRootElement`: 최상위 Element 태그
  - ✓ `@XmlElement`: 각 객체에 해당하는 태그
  - ✓ `@XmlType`: 변환시 사용할 namespace와 element 출력 순서를 설정

# XML

This XML file does not appear to have any style information associated with it. The document tree is shown below.

---

```
▼ <ITEMLIST>
  ▼ <ITEM>
    <itemid>1</itemid>
    <itemname>Lemon</itemname>
    <price>500</price>
    <description>Vitamin-A</description>
    <pictureurl>lemon.jpg</pictureurl>
  </ITEM>
  ▼ <ITEM>
    <itemid>2</itemid>
    <itemname>Orange</itemname>
    <price>1500</price>
    <description>Vitamin-B</description>
    <pictureurl>orange.jpg</pictureurl>
  </ITEM>
  ▼ <ITEM>
    <itemid>3</itemid>
    <itemname>Kiwi</itemname>
    <price>2000</price>
    <description>Vitamin-C</description>
    <pictureurl>kiwi.jpg</pictureurl>
  </ITEM>
  ▼ <ITEM>
    <itemid>4</itemid>
    <itemname>Grape</itemname>
    <price>1000</price>
    <description>Vitamin-D</description>
    <pictureurl>grape.jpg</pictureurl>
  </ITEM>
```

# XML

## ❖ xml 라이브러리 사용을 위한 코드를 pom.xml 파일에 추가

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-oxm</artifactId>
  <version>${org.springframework-version}</version>
</dependency>
<dependency>
  <groupId>javax.xml.bind</groupId>
  <artifactId>jaxb-api</artifactId>
  <version>2.1</version>
</dependency>
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-impl</artifactId>
  <version>2.2.11</version>
</dependency>
<dependency>
  <groupId>com.sun.xml.bind</groupId>
  <artifactId>jaxb-core</artifactId>
  <version>2.2.11</version>
</dependency>
```

# XML

## ❖ home.jsp 파일에 링크 추가

```
<li> <a href="item.xml" class="menu">XML 출력</a> </li>
```



# XML

## ❖ Item 클래스 수정

```
import javax.xml.bind.annotation.XmlAccessType;  
import javax.xml.bind.annotation.XmlAccessorType;  
import javax.xml.bind.annotation.XmlType;
```

```
@XmlAccessorType(XmlAccessType.FIELD)  
@XmlType(name = "", propOrder = { "itemid", "itemname", "price", "description",  
"pictureurl" })
```

```
public class Item {  
    private int itemid;  
    private String itemname;  
    private int price;  
    private String description;  
    private String pictureurl;
```

# XML

## ❖ 기본 패키지.domain 패키지에 데이터 모임을 나타낼 클래스 생성

```
import java.util.List;

import javax.xml.bind.annotation.*;
@XmlAccessorType(XmlAccessType.FIELD)
@XmlRootElement(name = "ITEMLIST")

public class ItemReport {
    @XmlElement(name = "ITEM")
    private List<Item> list;
    public ItemReport() {
    }
    public ItemReport(List<Item> list) {
        this.list = list;
    }
    public List<Item> getList() {
        return list;
    }
}
```

# XML

## ❖ HomeController 클래스에 추가

```
@RequestMapping("/item.xml")
public String xmlReport(Model model) {
    List<Item> list = itemService.getAll();
    model.addAttribute("list", new ItemReport(list));
    return "itemreport";
}
```

# XML

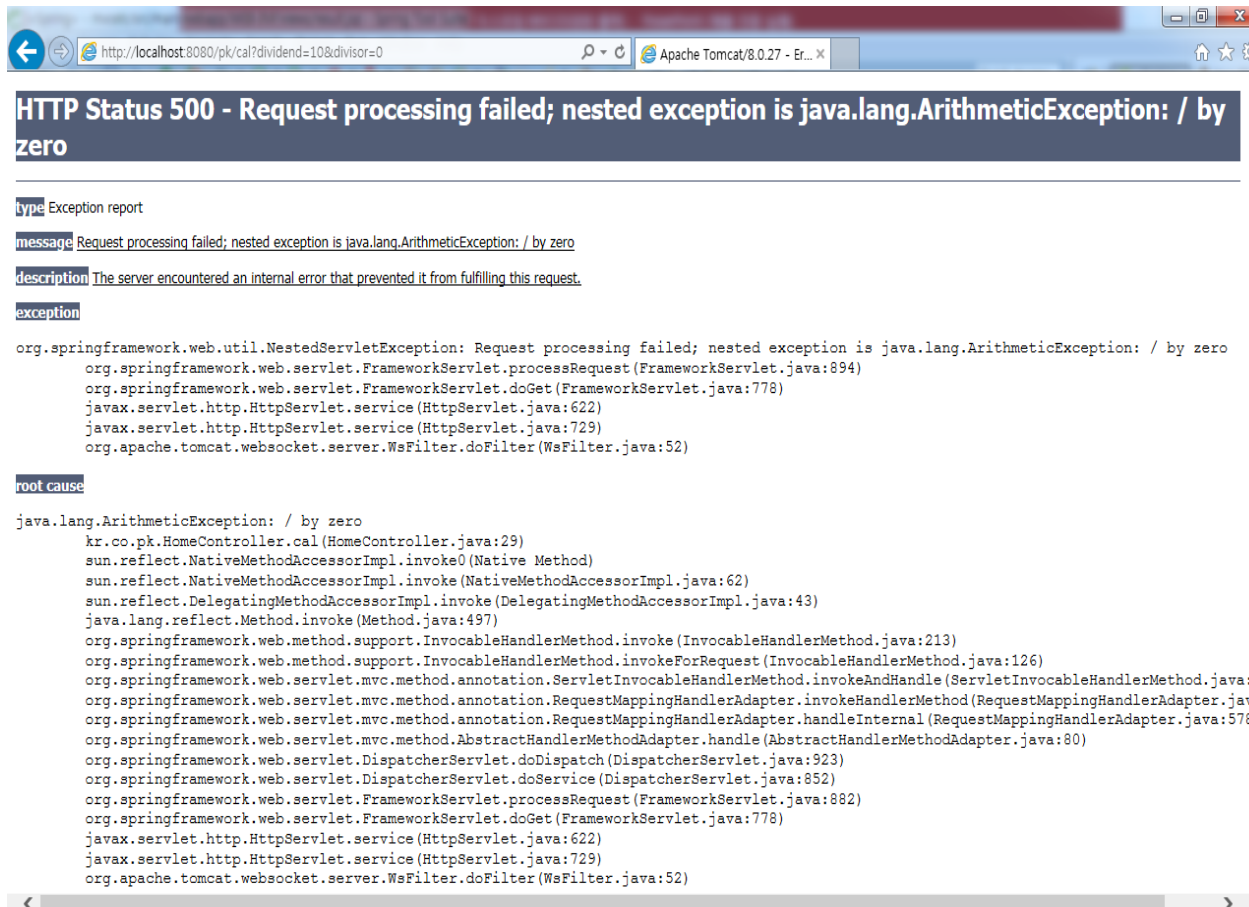
## ❖ servlet-context.xml 파일에 추가

```
<beans:bean id="marshaller"  
  class="org.springframework.xml.jaxb.Jaxb2Marshaller">  
  <beans:property name="classesToBeBound">  
    <beans:list>  
      <beans:value> 기본 패키지.domain.ItemReport </beans:value>  
    </beans:list>  
  </beans:property>  
</beans:bean>  
  
<beans:bean id="itemreport"  
  class="org.springframework.web.servlet.view.xml.MarshallingView">  
  <beans:property name="marshaller" ref="marshaller" />  
  <beans:property name="modelKey" value="list" />  
</beans:bean>
```



# 스프링 예외처리

- ❖ Controller의 메서드를 처리하는 도중에 예외가 발생했을 때 아래와 같은 에러 화면이 사용자에게 그대로 노출되면 보기에 좋지 않음



```
HTTP Status 500 - Request processing failed; nested exception is java.lang.ArithmeticException: / by zero

type Exception report
message Request processing failed; nested exception is java.lang.ArithmeticException: / by zero
description The server encountered an internal error that prevented it from fulfilling this request.
exception
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is java.lang.ArithmeticException: / by zero
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:894)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:778)
javax.servlet.http.HttpServlet.service(HttpServlet.java:622)
javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
root cause
java.lang.ArithmeticException: / by zero
kr.co.pk.HomeController.cal(HomeController.java:29)
sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
java.lang.reflect.Method.invoke(Method.java:497)
org.springframework.web.method.support.InvocableHandlerMethod.invoke(InvocableHandlerMethod.java:213)
org.springframework.web.method.support.InvocableHandlerMethod.invokeForRequest(InvocableHandlerMethod.java:126)
org.springframework.web.servlet.mvc.method.annotation.ServletInvocableHandlerMethod.invokeAndHandle(ServletInvocableHandlerMethod.java:
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.invokeHandlerMethod(RequestMappingHandlerAdapter.java:
org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter.handleInternal(RequestMappingHandlerAdapter.java:578)
org.springframework.web.servlet.mvc.method.AbstractHandlerMethodAdapter.handle(AbstractHandlerMethodAdapter.java:80)
org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:923)
org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:852)
org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:882)
org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:778)
javax.servlet.http.HttpServlet.service(HttpServlet.java:622)
javax.servlet.http.HttpServlet.service(HttpServlet.java:729)
org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

# 스프링 예외처리

## ❖ home.jsp 파일에 링크 추가

```
<li> <a href="exception" class="menu">예외처리 </a> </li>
```



# 스프링 예외처리

## ❖ view 폴더에 input.jsp 파일을 만들고 작성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>입력</title>
</head>
<body>
    <form action="cal">
        나누어지는 수:<input type="text" name="dividend" /><br />
        나누는 수:<input type="text" name="divisor" /><br />
        <input type="submit" value="계산" />
    </form>
</body>
</html>
```

# 스프링 예외처리

## ❖ HomeController에 exception 과 cal 요청 처리를 위한 메서드를 추가

```
@RequestMapping(value = "/exception", method = RequestMethod.GET)
public String input(Model model) {
    return "input";
}
```

```
@RequestMapping(value = "/cal", method = RequestMethod.GET)
public String cal(
    @RequestParam("dividend") int dividend , @RequestParam("divisor") int divisor, Model
    model) {
    model.addAttribute("result", dividend/divisor);
    return "result";
}
```

# 스프링 예외처리

## ❖ 결과를 출력할 result.jsp를 view 폴더에 추가하고 작성

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

# 스프링 예외처리

- ❖ 일반 웹 프로젝트에서는 web.xml 파일에 <error-page> 설정을 이용해서 예외가 발생한 경우에 보여지는 페이지를 설정
- ❖ 스프링에서는 예외 처리 방법을 자체적으로 제공
  - ✓ @ExceptionHandler Annotation을 이용한 예외 처리
  - ✓ @ControllerAdvice Annotation을 이용한 공통 예외 처리

# 스프링 예외처리

## ❖ @ExceptionHandler Annotation을 이용한 예외 처리

- ✓ 예외를 처리할 메서드 위에 @ExceptionHandler를 기재하고 괄호 안에 처리할 예외 클래스를 설정하면 예외가 발생하면 이 메서드가 호출

- ✓ 작성 방법

```
@ExceptionHandler(ArithmeticException.class)
public String handleException() {
    return "error/exception";
}
```

- ✓ 위처럼 작성하면 ArithmeticException 이 발생하면 exception 페이지가 있다면 그 페이지를 출력
- ✓ 메서드에서 예외 객체를 사용하고자 하면 예외를 매개변수로 설정
- ✓ jsp 페이지에서 <%@ page isErrorPage="true" %>를 설정하면 exception 객체를 사용할 수 있음

# 스프링 예외처리

## ❖ HomeController에 예외 처리를 위한 메서드를 추가

```
@ExceptionHandler(Exception.class)
public ModelAndView handleException(Exception e){
    ModelAndView mav = new ModelAndView();
    mav.setViewName("error/exception");
    System.out.println(e.getMessage());
    mav.addObject("result", e.getMessage());
    return mav;
}
```



# 스프링 예외처리

- ❖ **view 디렉토리에 error 디렉토리를 만들고 exception.jsp 파일을 생성하고 작성**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page isErrorPage="true" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<meta name="dummy1" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지
에러페이지 에러페이지"/>
<meta name="dummy2" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지
에러페이지 에러페이지"/>
<meta name="dummy3" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지
에러페이지 에러페이지"/>
<meta name="dummy4" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지
에러페이지 에러페이지"/>
<meta name="dummy1" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지
에러페이지 에러페이지"/>
```

# 스프링 예외처리

## ❖ view 디렉토리에 error 디렉토리를 만들고 exception.jsp 파일을 생성하고 작성

```
<meta name="dummy3" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지  
에러페이지 에러페이지"/>  
<meta name="dummy4" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지  
에러페이지 에러페이지"/>  
<meta name="dummy1" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지  
에러페이지 에러페이지"/>  
<meta name="dummy2" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지  
에러페이지 에러페이지"/>  
<meta name="dummy3" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지  
에러페이지 에러페이지"/>  
<meta name="dummy4" content="에러페이지 에러페이지 에러페이지 에러페이지 에러페이지  
에러페이지 에러페이지"/>  
  
<title>에러 페이지</title>  
</head>  
<body>  
    예외 내용:${result}  
</body>  
</html>
```

# 스프링 예외처리

## ❖ @ControllerAdvice("패키지이름") 을 이용한 예외 처리

- ✓ Controller 클래스에 @ExceptionHandler를 적용하면 하나의 Controller에서 발생한 예외만 처리
- ✓ 클래스를 만들고 @ControllerAdvice("패키지이름")을 기재한 클래스를 만들면 패키지 및 하위 패키지에서 발생하는 모든 예외를 처리하는 메서드를 만들 수 있음

```
@ControllerAdvice("kr.co.pk")
public class CommnoExceptionHandler{
    @ExceptionHandler(RuntimeException.class)
    public String handleRuntimeException(){

    }
}
```

- ✓ 위 클래스를 bean으로 등록하면 kr.co.pk 패키지 내에서 예외가 발생하면 handleRuntimeException 메서드를 호출

# 스프링 MVC 설정

## ❖ 설정을 이용한 뷰 출력 설정

- ✓ /index 경로로 요청이 오면 단순히 index 뷰를 출력하고자 하는 경우에는 아래와 같은 메서드가 필요

```
@RequestMapping("/index")  
public String index(){  
    return "index";  
}
```

- ✓ 특별한 구현없이 단순히 요청 경로에 알맞은 뷰 이름만 리턴해주는 컨트롤러가 필요한 경우에는 아래와 같은 설정으로 쉽게 처리

```
<mvc:view-controller path="/index" view-name="index" />
```

- ❑ 위에서 path 설정은 컨텍스트 경로를 제외한 부분

# 스프링 MVC 설정

## ❖ 뷰의 디폴트 설정

- ✓ web.xml에서 DispatcherServlet에 대한 경로 매핑을 /로 설정하게 되면 .jsp 요청을 제외한 모든 요청을 DispatcherServlet으로 전달되서 css, js, html 으로 끝나는 요청의 경우 controller에서 처리하지 않는 이상 404 응답 에러가 발생
- ✓ Controller가 처리하지 못하는 경우 WAS가 기본으로 제공하는 디폴트 Servlet이 처리하도록 설정

<mvc:default-servlet-handler/>

□ 위 설정이 있으면 아래 순서대로 요청을 처리

1. 요청 경로를 처리할 수 있는 Controller의 메서드를 검색해서 처리
2. Controller가 처리하지 않으면 Default Servlet Handler에 요청을 전달
3. Default Servlet Handler는 WAS의 Servlet Handler에 처리를 위임
4. Servlet Handler의 처리 결과를 응답으로 전송

# 스프링 MVC 설정

## ❖ 정적 자원 설정

- ✓ css, js, image 등의 자원은 거의 변하지 않기 때문에 웹 브라우저에 캐시를 하면 네트워크 사용량, 서버 사용량, 웹 브라우저의 반응 속도 등을 개선할 수 있음
- ✓ 정적 자원은 웹 서버에서 제공하기 때문에 웹 서버의 캐시 옵션 설정을 통해 웹 브라우저 캐시를 활성화 시킬 수 있음
- ✓ 웹 애플리케이션에 정적 자원 파일이 포함되어 있는 경우에는 <mvc:resources> 설정을 이용해서 웹 브라우저 캐시 설정을 하도록 지정할 수 있음

```
<mvc:resources mapping="/images/**" location="/img/, /WEB-INF/resources/"  
cache-period="60" />
```

- ❑ 자원의 요청 경로가 /images로 시작하면 /img/나 resources 디렉토리에서 검색
- ❑ 응답 헤더에 캐시 유효 시간을 60초로 지정해서 웹 브라우저에 전송

# Spring Custom Tag

- ❖ spring으로 시작하는 태그를 사용하기 위한 태그 라이브러리  
    <%@ taglib prefix="spring" uri="http://www.springframework.org/tags" %>



# Spring 메시지 출력

## ❖ messageSource(문자열을 가지고 올 파일) 설정 – 스프링 설정 파일에 작성

### ✓ 메시지 파일 작성

- ❑ 메시지 파일의 확장자는 properties
- ❑ 지역화를 할 때는 파일명\_언어.properties 파일로 파일 이름을 작성
- ❑ 메시지는 code = value 형태로 작성
- ❑ `<bean class="org.springframework.context.support.ResourceBundleMessageSource" id="messageSource">`
- ❑ 메시지 파일 경로 설정 – Spring Bean Configuration 파일에 작성

```
<beans:bean
class="org.springframework.context.support.ResourceBundleMessageSource"
id="messageSource">
    <property name="basenames">
        <list>
            <value>루트로부터의 경로</value>
            ....
        </list>
    </property>
</bean>
```

- 경로를 작성할 때 확장자는 생략



# Spring 메시지 출력

## ❖ <spring:message>

- ✓ messageSource로 부터 문자열을 가져와서 출력하는 태그
- ✓ code 속성에 properties 파일의 키를 설정
- ✓ message에 {0} 처럼 Parameter(place holder)가 있는 경우라면 arguments="값", "값" 의 형식으로 대입
- ✓ messageSource에 code 값이 없으면 예외가 발생하는데 text 속성에 값을 대입하면 code가 없으면 그 값을 출력
- ✓ JavaScript에서 이 태그의 값을 사용하고자 하는 경우에는 태그 안에 javaScriptEscape="true"를 설정
- ✓ 메시지를 attribute에 저장할 수 있는데 이 경우에는 태그 안에 var 속성에 변수 이름을 기재하고 scope 속성에 request, session, application 중에서 하나를 기재

# Spring 메시지 출력

이메일:

암호:

로그인

- 이메일/암호로 root/1234로 입력 테스트

# Spring 메시지 출력

- ❖ 기본적으로 제공되는 home.jsp 파일에 링크 추가

```
<li> <a href="message" class="menu">스프링 메시지 출력</a> </li>
```

# Spring 메시지 출력

- ❖ 프로젝트의 `src/main/resources` 디렉토리에 `message` 디렉토리를 생성
- ❖ `message` 디렉토리에 `label.properties` 파일을 생성하고 작성

`email=이메일`

`password=암호`

`login.form.title=로그인 폼`

`login.form.login=로그인`

`login.form.help=이메일/암호로 root/1234로 입력 테스트`

`login.form.type=로그인 유형`

`greeting={0} 회원님, {1}`

# Spring 메시지 출력

## ❖ servlet-context.xml 파일에 메시지를 읽을 수 있도록 설정

```
<beans:bean  
class="org.springframework.context.support.ResourceBundleMessageSource"  
    id="messageSource">  
    <beans:property name="basenames">  
        <beans:list>  
            <beans:value>message/label</beans:value>  
        </beans:list>  
    </beans:property>  
</beans:bean>
```

# Spring 메시지 출력

## ❖ HomeController에 message 요청을 처리하는 메서드를 작성

```
@RequestMapping(value = "/message", method = RequestMethod.GET)
public String form() {
    return "loginform";
}
```

# Spring 메시지 출력

- ❖ 기본적으로 제공되는 WEB-INF 디렉토리의 view 디렉토리에 loginform.jsp 파일을 생성하고 작성

```
<%@ page contentType="text/html; charset=utf-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<!DOCTYPE html>
<html>
<head>
<title> <spring:message code="login.form.title" /> </title>
</head>
```

# Spring 메시지 출력

- ❖ 기본적으로 제공되는 WEB-INF 디렉토리의 view 디렉토리에 loginform.jsp 파일을 생성하고 작성

```
<body>
  <form action="loginform" method="post">
    <p>
      <label for="email"> <spring:message code="email" /> </label>:
    <input
      type="text" name="email" id="email">
    </p>
    <p>
      <label for="password"> <spring:message code="password"
/> </label>: <input
      type="password" name="password" id="password">
    </p>
    <input type="submit"
      value="<spring:message code="login.form.login" />">
    </form>
    <ul>
      <li> <spring:message code="login.form.help" /> </li>
    </ul>
  </body>
</html>
```



# 폼 관련 커스텀 태그

- ❖ 스프링 폼 태그 사용을 위한 태그 라이브러리 설정

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

- ❖ `<form:form>`

- ✓ form 태그를 생성할 때 사용
- ✓ method 속성은 설정하지 않으면 post로 설정
- ✓ action은 설정하지 않으면 현재 요청 URL로 설정
- ✓ id 속성은 입력 폼의 값을 저장하는 커맨드 객체의 이름이 할당
- ✓ 커맨드 객체의 이름이 기본값인 "command"가 아니라면 `modelAttribute(4.x 버전에서는 - commandName)` 속성에 커맨드 객체의 이름을 명시해야 함
- ✓ 입력한 값이 잘못되어 다시 값을 출력해야 하는 경우에는 커맨드 객체의 값을 이용해서 값을 출력

# 폼 관련 커스텀 태그

- ❖ `<form:input>`: text 타입의 input 태그
- ❖ `<form:password>`: password 타입의 input 태그
- ❖ `<form:hidden>`: hidden 타입의 input 태그
- ❖ 위 3개의 태그에 `path` 속성을 설정하면 `id`와 `name`에 커맨드 객체의 프로퍼티 이름이 자동 할당되고 `value`에는 값이 할당

```
<form:form commandName="member">
```

```
<p>
```

```
    <form:input path="userId" />
```

```
</p>
```

위 태그는 아래처럼 변환

```
<input id="userId" name="userId" type="text" value="${member.userId}" />
```

# 폼 관련 커스텀 태그

## ❖ email 과 pw를 저장할 수 있는 Command Class 생성

```
import lombok.Data;

@Data
public class Member {
    private String email;
    private String pw;
}
```



# 폼 관련 커스텀 태그

## ❖ HomeController에 message 요청을 처리하는 메서드를 수정

```
@RequestMapping(value = "/message", method = RequestMethod.GET)
public String form(@ModelAttribute("member")Member member) {
    return "loginform";
}
```

# 폼 관련 커스텀 태그

## ❖ loginform.jsp 파일 수정

```
<%@ page contentType="text/html; charset=utf-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html>
<html>
<head>
<title> <spring:message code="login.form.title" /> </title>
</head>
<body>
    <!-- Spring 4.x 버전에서는 modelAttribute 대신에 commandName -->
    <form:form modelAttribute="member">
        <p>
            <label for="email"> <spring:message code="email" /> </label>:
            <form:input path="email" />
        </p>
        <p>
            <label for="password"> <spring:message code="password" />:
            <form:password path="pw" />
        </p>
    </form:form>
</body>
</html>
```

# 폼 관련 커스텀 태그

## ❖ loginform.jsp 파일 수정

```
<input type="submit"
      value="<spring:message code="login.form.login" />" />
</form:form>

<ul>
    <li> <spring:message code="login.form.help" /> </li>
</ul>
</body>
</html>
```

# 폼 관련 커스텀 태그

- ❖ `<form:select>`: 선택 태그 만들 때 사용
  - ✓ `path` 속성은 Command 객체와 연결 될 때 사용
  - ✓ `items` 속성에 List 타입의 속성을 대입하면 자동으로 옵션을 생성
- ❖ `<form:option>`
  - ✓ `<form:options>`: `items` 속성에 지정된 List 타입의 속성을 가지고 옵션을 생성하며 `value`와 `label`의 내용은 List의 문자열을 순서대로 적용
  - ✓ `<form:option>`: 하나의 옵션을 생성할 때 사용하는 것으로 만일 String 의 List가 아니고 DTO의 List 라면 `itemLabel`에 Label에 출력될 프로퍼티를 설정하고 `itemValue`에 Value에 설정될 프로퍼티를 설정

# 폼 관련 커스텀 태그

## ❖ HomeController에 아래 메서드 추가

```
@ModelAttribute("loginTypes")
public List<String> loginTypes(){
    List<String>list = new ArrayList<String>();
    list.add("일반회원");
    list.add("기업회원");
    list.add("비회원");
    return list;
}
```



# 폼 관련 커스텀 태그

- ❖ Member 클래스에 String loginType을 프로퍼티로 추가

```
private String loginType;
```

- ❖ loginform.jsp 파일의 form 안에 아래 코드 추가

```
<p>
```

```
    <form:select path="loginType" items="${loginTypes}" />
```

```
</p>
```



# 폼 관련 커스텀 태그

- ❖ `<form:checkboxes>`: 선택 태그 만들 때 사용
  - ✓ `path` 속성은 `Command` 객체와 연결 될 때 사용
  - ✓ `items` 속성에 `List` 타입의 속성을 대입하면 자동으로 `checkbox`를 생성 - `value`와 `label`의 내용은 `List`의 문자열을 순서대로 적용
  - ✓ `<form:checkbox>`: 하나의 옵션을 생성할 때 사용
  - ✓ `radiobuttons` 와 `radiobutton` 도 동일하게 적용
- ❖ `<form:textarea>`는 `path` 속성만 적용
- ❖ `<form:errors path="email">`: `email`에 관련된 에러 메시지를 출력

# Spring MVC(유효성 검사)

# 폼 데이터 검증

- ❖ 웹 프로젝트에서 요청 Parameter 값을 확인하지 않고 그대로 사용하게 되면 데이터베이스에 잘못된 데이터가 저장되는 일이 발생
- ❖ 유효성 검사 방법
  - ✓ 클라이언트에서 서버에게 전송하기 전에 검사 - 웹 프로그래밍에서는 JavaScript 나 HTML5 속성을 이용해서 검사(정규식 등을 이용)
  - ✓ 전송받은 데이터를 서버에서 검사한 후 데이터가 잘못된 경우 재입력을 위한 화면을 출력하는 방법
- ❖ 스프링은 서버 측에서 검사하는 방법을 제공
- ❖ 클라이언트 측에서 유효성 검사를 하는 방법은 사용자에게 빠르게 검사 결과를 보여줄 수 있다는 장점이 있지만 보안에 취약하기 때문에 보안에 관련된 부분은 서버에서 유효성 검사를 해주는 것이 바람직하고 전송 중간에 데이터의 변경이 있을 수 있으므로 서버에서도 수행하는 것을 권장

# 폼 데이터 검증

## ❖ Spring 의 유효성 검사

- ✓ 데이터 검증에 사용되는 인터페이스인 `org.springframework.validation.Validator` 와 유효성 검사 결과를 저장할 `Errors`와 `BindingResult` 인터페이스를 제공
- ✓ `Validator` 인터페이스
  - ❑ 유효성 검사를 하고 유효성 검사에 실패한 경우 `Errors` 인터페이스를 이용해서 에러 메시지를 저장한 후 뷰에 전달해서 출력하고 `BindingResult` 인터페이스에 에러 발생여부를 저장
  - ❑ `Validator` 인터페이스에는 2개의 메서드가 존재
    - `public boolean supports(Class<?> clazz)`: `Validator`가 해당 클래스에 대한 값 검증을 지원하는지의 여부를 설정하는 메서드  
Ex) `return Info.class.isAssignableFrom(clazz);`
    - `public void validate(Object target, Errors errors)`: `target`에 대한 검증을 수행하고 검증 결과에 문제가 발생하면 `errors` 객체에 문제에 대한 정보를 저장  
Ex) 

```
Info info = (Info) target;  
if (info.getId() == null || info.getId().trim().isEmpty()) {  
    errors.rejectValue("id", "required");  
}
```

      - `id`는 유효성 검사에 실패한 객체 이름이 되고 `required`는 뷰에서 에러메시지를 출력하도록 설정하면 출력되는 에러 메시지의 프로퍼티
      - 위의 경우는 `required.id` 프로퍼티에 설정된 문자열을 에러 메시지로 출력

# 폼 데이터 검증

## ❖ Spring 의 유효성 검사

- ✓ Errors 인터페이스: 유효성 검증 결과를 저장하기 위한 인터페이스
  - ❑ `reject(String errorCode)`: 전체 객체에 대한 에러 코드 설정
  - ❑ `reject(String errorCode, String defaultMessage)`: 에러 코드와 기본 메시지 설정
  - ❑ `rejectValue(String field, String errorCode)`: field 대한 에러 코드 설정
  - ❑ `rejectValue(String field, String errorCode, String defaultMessage)`
- ✓ `BindingResult` 인터페이스: Errors의 하위 인터페이스로 폼의 값들을 `command` 객체에 저장하고 에러 코드로부터 에러 메시지를 가져오는 인터페이스
  - ❑ `boolean hasErrors()`
  - ❑ `int getErrorCount()`
  - ❑ `boolean hasGlobalErrors()`
  - ❑ `int getGlobalErrorCount()`
  - ❑ `boolean hasFieldErrors()`
  - ❑ `int getFieldErrorCount()`
  - ❑ `boolean hasFieldErrors(String field)`
  - ❑ `int getFieldErrorCount(String field)`

# 폼 데이터 검증

## ❖ Member에 대한 유효성 검사 클래스

```
public class MemberInfoValidator implements Validator {  
    @Override  
    public boolean supports(Class<?> clazz) {  
        return MemberInfo.class.isAssignableFrom(clazz);  
    }  
    @Override  
    public void validate(Object target, Errors errors) {  
        MemberInfo memberInfo = (MemberInfo) target;  
        if (memberInfo.getId() == null || memberInfo.getId().trim().isEmpty()) {  
            errors.rejectValue("id", "required");  
        }  
        if (memberInfo.getName() == null  
            || memberInfo.getName().trim().isEmpty()) {  
            errors.rejectValue("name", "required");  
        }  
    }  
}
```

# 폼 데이터 검증

## ❖ Controller 클래스에 유효성 검사를 진행하도록 설정

```
@RequestMapping(method = RequestMethod.POST)
public String submit(@ModelAttribute("command") MemberInfo memberInfo,
    BindingResult result) {
    new MemberInfoValidator().validate(memberInfo, result);
    if (result.hasErrors()) {
        return "account/creationForm";
    }
    return "account/created";
}
```



# 폼 데이터 검증

- ❖ 에러 코드에 대한 에러 메시지 설정은 메시지를 사용하는 방법과 동일

```
<bean id="아이디"  
class="org.springframework.context.support.ResourceBundleMessageSource">  
    <property name="basenames">  
        <list>  
            <value>properties 파일 경로</value>  
        </list>  
    </property>  
</bean>
```

- ❖ 에러 코드에 해당하는 프로퍼티에 에러 메시지 작성은 전체 메시지를 작성할 수 있고 특정 객체에 대한 메시지를 작성할 수 있음

required=₩ud544₩uc218 ₩ud56d₩ubaa9₩uc785₩ub2c8₩ub2e4.

required.password=₩ud544₩uc218 ₩ud56d₩ubaa9₩uc785₩ub2c8₩ub2e4.

# 폼 데이터 검증

## ❖ jsp에서 에러 메시지 출력

<spring:hasBindErrors name="커맨드 객체 이름"> 커스텀 태그를 이용해서 에러 정보를 설정하고 <form:errors path="커맨드 객체" />를 이용하여 global 에러 정보를 설정하고 각각의 에러는 <form:errors path="커맨드 객체.프로퍼티" />를 이용

# 실습

아이디:  아이디는 필수 항목입니다.

이름:  필수 항목입니다.

쿼리 전송



# 실습

- ❖ 기본 패키지.validator 패키지를 생성하고 Validator 인터페이스를 구현한 MemberValidator 클래스 생성

```
public class MemberValidator implements Validator {

    @Override
    public boolean supports(Class<?> clazz) {
        return Member.class.isAssignableFrom(clazz);
    }

    @Override
    public void validate(Object target, Errors errors) {
        Member memberInfo = (Member) target;
        if (memberInfo.getEmail() == null || memberInfo.getEmail().trim().isEmpty()) {
            errors.rejectValue("email", "required");
        }
        if (memberInfo.getPw() == null || memberInfo.getPw().trim().isEmpty()) {
            errors.rejectValue("pw", "required");
        }
    }
}
```

# 실습

## ❖ HomeController 요청 처리 메서드 수정 및 생성

```
@RequestMapping(value = "/message", method = RequestMethod.GET)
public String form(@ModelAttribute("command") Member member) {
    return "loginform";
}
```

```
@RequestMapping(value = "/message", method = RequestMethod.POST)
public String created(@ModelAttribute("command") Member member,
    BindingResult result) {
    new MemberValidator().validate(member, result);
    if (result.hasErrors()) {
        return "loginform";
    }
    return "redirect:/";
}
```

# 실습

## ❖ WEB-INF 디렉토리의 view 디렉토리의 loginform.jsp 파일 수정

```
<%@ page contentType="text/html; charset=utf-8"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<!DOCTYPE html>
<html>
<head>
<title> <spring:message code="login.form.title" /> </title>
</head>
<body>
    <!-- Spring 4.x 버전에서는 modelAttribute 대신에 commandName -->
    <form:form modelAttribute="command">
        <p>
            <label for="email"> <spring:message code="email" /> </label>:
            <form:input path="email" />
            <form:errors path="email" />
        </p>
        <p>
            <label for="password"> <spring:message code="password"
/> </label>:
            <form:password path="pw" />
            <form:errors path="pw" />
        </p>
```

# 실습

## ❖ WEB-INF 디렉토리의 view 디렉토리의 loginorm.jsp 파일 수정

```
<p>
    <form:select path="loginType" items="${loginTypes}" />
</p>

<input type="submit"
    value="<spring:message code="login.form.login" />" />
</form:form>

<ul>
    <li><spring:message code="login.form.help" /> </li>
</ul>
</body>
</html>
```

# 실습

❖ **resources** 디렉토리의 **messages** 디렉토리에 **validation.properties** 파일을 생성하고 작성

```
required=WuD544WuC218 Wud56dWubaa9Wuc785Wub2c8Wub2e4.  
required.email=WuC544WuC774WuB514WuB294 WuD544WuC218  
WuD56DWuBAA9WuC785WuB2C8WuB2E4.
```

필수 항목 입니다.

이메일은 필수 항목입니다.



# 실습

## ❖ servlet-context.xml 파일의 messageSource 빈 수정

```
<beans:bean class="org.springframework.context.support.ResourceBundleMessageSource"
  id="messageSource">
  <beans:property name="basenames">
    <beans:list>
      <beans:value>message/label</beans:value>
      <beans:value>message/validation</beans:value>
    </beans:list>
  </beans:property>
</beans:bean>
```

# 실습 - 국제화

이메일:  이메일은 필수 항목입니다.

암호:  필수 항목 입니다.

일반회원 ▼

로그인

- 이메일/암호로 root/1234로 입력 테스트

이메일:  email is required.

암호:  required.

일반회원 ▼

로그인

- 이메일/암호로 root/1234로 입력 테스트

# 실습

- ❖ 기존의 validation.properties 파일을 validation\_ko.properties 파일로 이름 변경
- ❖ validation\_en.properties 파일 생성
  - required=required.
  - required.email=email is required.

# 실습

- ❖ null이나 공백 문자로만 이루어진 문자열을 매번 검사하는 것은 매우 귀찮은 작업 중의 하나
- ❖ 이 경우 Spring에서는 ValidationUtils 클래스의 static 메서드를 이용해서 이 문제를 간단하게 작성 가능
- ❖ 메서드

`rejectIfEmpty(Errors errors, String field, String errorCode)`

`rejectIfEmptyOrWhiteSpace(Errors errors, String field, String errorCode)`

# 실습

## ❖ MemberValidator 클래스의 validate 메서드 수정

```
@Override
public void validate(Object target, Errors errors) {
    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "email", "required");
    ValidationUtils.rejectIfEmptyOrWhitespace(errors, "pw", "required");
}
```

# @Valid Annotation

- ❖ @Valid Annotation은 Bean Validation API(JSR 303)에 의해 정의되는데 @NotNull, @Digits, @Size 등의 Annotation을 제공
- ❖ 이 Annotation들을 이용하면 Validator 클래스 없이 Annotation만으로 Command 객체의 값 검증을 처리 가능
- ❖ 검증 방법
  - ✓ 라이브러리 다운로드(Validator 와 Hibernate Validator)
  - ✓ Annotation을 이용해서 검증 규칙을 설정
  - ✓ LocalValidatorFactoryBean 클래스를 이용해서 JSR 303 프로바이더를 스프링의 Validator로 설정
  - ✓ Controller가 생성한 빈을 Validator로 사용하도록 설정

# @Valid Annotation

## ❖ pom.xml 파일에 의존성 설정

```
<dependency>  
  <groupId>javax.validation</groupId>  
  <artifactId>validation-api</artifactId>  
  <version>2.0.1.Final</version>  
</dependency>  
<dependency>  
  <groupId>org.hibernate</groupId>  
  <artifactId>hibernate-validator</artifactId>  
  <version>6.0.9.Final</version>  
</dependency>
```

# @Valid Annotation

- ❖ LocalValidatorFactoryBean 클래스를 이용해서 JSR 303 프로바이더를 스프링의 Validator로 설정하고자 할 때는 <mvc:annotation-driven/> 태그를 사용하면 기본적으로 등록
- ❖ Command 객체의 유효성 검사를 위해서는 @Controller에서 Command 객체 앞에 @Valid를 추가
- ❖ JSR의 Annotation
  - ✓ @Size: 속성으로는 min 과 max 속성이 있어서 숫자인 경우 min 과 max 속성 사이의 값인지 확인하고 문자열의 경우는 문자열의 길이를 검사하고 컬렉션인 경우는 길이를 검사하는데 값이 null이면 유효한 것으로 판단
  - ✓ @Min, @Max: 속성으로는 value를 설정할 수 있는데 숫자의 값의 최대 또는 최소를 검사하는데 BigDecimal, BigInteger, 정수 타입 및 정수 래퍼 타입에 적용되는데 double 과 float은 지원하지 않으며 null은 유효한 값
  - ✓ @Digits: integer 속성으로 정수 부분의 길이를 지정할 수 있고 fraction을 이용해서 소수부분의 길이를 지정할 수 있으며 null은 유효한 값
  - ✓ @Pattern: regexp 속성에 정규 표현식을 대입해서 문자열이 지정한 패턴에 일치하는지 검사



# @Valid Annotation

## ❖ Hibernate Validator의 Annotation

- ✓ @NotEmpty: String의 경우 빈 문자열이 아니어야 하며 컬렉션은 크기가 1이상
- ✓ @NotBlank: @NotEmpty와 동일하지만 뒤의 공백은 무시
- ✓ @Length: min과 max 속성을 대입해서 문자열의 길이를 검사
- ✓ @Range: min 과 max 속성을 이용해서 숫자의 범위 검사
- ✓ @Email: 이메일 경로 검사
- ✓ @URL: url 검사
- ✓ NotEmpty 나 NotBlank 대신에 NotNull을 사용 가능

- ❖ 에러 메시지는 기본적으로 작성되어 있어서 <form:errors path="속성명"/>의 형태로 하면 출력되는데 사용자가 직접 메시지를 지정하고자 하는 경우에는 **Annotation이름.커맨드객체모델명.프로퍼티명** 으로 메시지를 만들면 됨

# 실습

아이디:  may not be empty

아이디는 필수 항목입니다.

비번:  may not be empty

size must be between 5 and 2147483647

필수 항목입니다.

제출

## ❖ pom.xml 파일에 의존성 추가

```
<dependency>  
  <groupId>javax.validation</groupId>  
  <artifactId>validation-api</artifactId>  
  <version>2.0.1.Final</version>  
</dependency>  
<dependency>  
  <groupId>org.hibernate</groupId>  
  <artifactId>hibernate-validator</artifactId>  
  <version>6.0.9.Final</version>  
</dependency>
```

# 실습

## ❖ Member 클래스에 Annotation 설정

```
public class Member {  
    @NotNull  
    @Pattern(regexp = "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\w\\.?(:[a-zA-Z]{2,6})$",  
message="Invalid E-Mail")  
    private String email;  
    @NotNull  
    @Size(min=5)  
    private String pw;  
    private String loginType;
```

# 실습

## ❖ HomeController의 폼의 데이터 전송 처리 메서드를 수정

```
@RequestMapping(value = "/message",method = RequestMethod.POST)
public String created(@Valid @ModelAttribute("command") Member member,
    Errors errors) {
    if (errors.hasErrors()) {
        return "loginform";
    }
    return "redirect:/";
}
```

# 실습

## ❖ validation\_ko.properties 파일에 메시지 추가

NotBlank=WuD544WuC218 WuD56DWuBAA9 WuC785WuB2C8WuB2E4.

NotBlank.pw=WuBE44WuBC00WuBC88WuD638WuB294 WuD544WuC218  
WuC785WuB825WuC785WuB2C8WuB2E4.

# 날짜 나 숫자 포맷 설정

- ❖ Command 클래스의 프로퍼티에 `@DateTimeFormat(pattern="포맷")` 이나 `@NumberFormat(pattern="포맷")`을 설정하면 날짜나 숫자의 형식을 검사 가능
- ❖ 패턴은 `java.text.SimpleDateFormat` 과 `java.text.NumberFormat` 클래스의 형식을 이용
- ❖ 에러 메시지는 `typeMismatch` 추가



# 실습

아이디:  필수 항목 입니다.

비번:  비밀번호는 필수 입력입니다.

size must be between 5 and 2147483647

생일:  날짜 형식이 맞지 않습니다.



# 실습

## ❖ Member 클래스에 Date 타입으로 프로퍼티 추가

```
@DateTimeFormat(pattern="yyyyMMDD")  
private Date birthday;
```



# 실습

## ❖ loginform.jsp 파일에 생일을 입력받을 수 있는 input 추가

```
<p>  
    생일:  
    <form:input path="birthday" />  
    <form:errors path="birthday" />  
</p>
```

## ❖ validation\_ko.properties 파일에 메시지 추가

```
typeMismatch.birthday=\\uB0A0\\uC9DC \\uD615\\uC2DD\\uC774 \\uB9DE\\uC9C0  
\\uC54A\\uC2B5\\uB2C8\\uB2E4.
```

# 파일 업로드 처리

- ❖ html에서의 파일 업로드를 위한 설정
  - ✓ 폼의 데이터를 일반적으로 전송하는 경우
    - ❑ 파일 업로드가 필요한 경우 html 폼의 enctype을 multipart/form-data으로 설정
    - ❑ method는 post로 설정
  - ✓ ajax로 전송할 때는 FormData 로 묶어서 전송



# 파일 업로드 처리

## ❖ 스프링에서의 파일 업로드 처리

- ✓ 스프링은 Multipart 기능을 제공하기 때문에 추가적인 처리없이 Multipart 형식으로 전송된 Parameter와 파일 정보를 쉽게 구할 수 있음
- ✓ Multipart 지원 기능을 사용하려면 먼저 MultipartResolver를 스프링 설정 파일에 등록해야 함
- ✓ MultipartResolver는 Multipart 형식으로 데이터가 전송된 경우 해당 데이터를 스프링 MVC에서 사용할 수 있도록 변환
- ✓ 스프링이 기본적으로 제공하는 MultipartResolver는 CommonsMultipartResolver인데 이 클래스는 Commons FileUpload API를 이용해서 Multipart를 처리
- ✓ Maven dependency

```
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.1</version>
</dependency>
```
- ✓ CommonsMultipartResolver 클래스의 빈을 생성하는 코드를 스프링 설정 파일에 추가

```
<bean id="multipartResolver"
class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
</bean>
```

# 파일 업로드 처리

## ❖ CommonsMultipartResolver 클래스의 프로퍼티

- ✓ long maxUploadSize: 업로드 가능한 파일의 크기로 단위는 byte, 기본값은 -1로 크기 제한 없음
- ✓ int maxInMemorySize: 디스크에 임시 파일을 생성하기 전에 메모리에 보관할 수 있는 최대 바이트 크기로 기본값은 10240바이트
- ✓ String defaultEncoding: 요청을 파싱할 때 사용할 캐릭터 인코딩으로 지정하지 않을 경우 request.setCharacterEncoding으로 설정한 값이 사용되며 이 메서드를 호출하지 않는 경우 ISO-8859-1

# 파일 업로드 처리

## ❖ File Parameter 처리 방법

- ✓ 요청을 `MultipartHttpServletRequest`(`HttpServletRequest`로 부터 상속)로 받아서 처리하는 방법
  - ❑ `Iterator<String> getFileNames()`: 파일의 이름 목록을 리턴
  - ❑ `MultipartFile getFile(String name)`: Parameter의 이름이 name인 업로드 파일 정보를 리턴
  - ❑ `List<MultipartFile> getFiles(String name)`: Parameter의 이름이 name인 업로드 파일 정보를 List로 리턴
  - ❑ `Map<String, MultipartFile> getFileMap()`: 파일에 대한 정보를 맵으로 리턴
- ✓ 요청을 처리하는 메서드의 매개변수로 `@RequestParam("Parameter이름") MultipartFile 변수 이름`의 형태로 받아서 처리
- ✓ Command 객체를 이용해서 처리 가능

# 파일 업로드 처리

## ❖ MultipartFile

- ✓ 업로드 한 파일 정보 및 파일 데이터를 표현하기 위한 용도로 사용되는 인터페이스
- ✓ 주요 메서드
  - ❑ String getName(): Parameter 이름을 리턴
  - ❑ String getOriginalFileName(): 업로드 한 파일의 원본 이름
  - ❑ boolean isEmpty()
  - ❑ long getSize()
  - ❑ byte[] getBytes(): 업로드 한 파일의 데이터
  - ❑ void transferTo(File dest) throws IOException: 파일을 지정한 파일에 저장
- ✓ getBytes()로 파일 데이터를 구한 후 데이터베이스나 파일에 저장하거나 transferTo를 이용해서 파일의 내용을 다른 파일에 전송
- ✓ transferTo를 이용할 때는 파일 객체를 생성한 후 MultipartFile 인터페이스의 transfer 메서드로 대입해서 호출

# 실습

## ❖ pom.xml 파일에 추가

```
<dependency>  
  <groupId>commons-fileupload</groupId>  
  <artifactId>commons-fileupload</artifactId>  
  <version>1.3.1</version>  
</dependency>
```



# 실습

## ❖ servlet-context.xml 파일에 추가

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



# 실습

## ❖ home.jsp에 링크 추가

```
<li><a href="fileupload" class="menu">파일 업로드 처리</a></li>
```



# 실습

## ❖ HomeController에 파일 업로드 처리 요청을 처리하는 메서드 작성

```
@RequestMapping(value = "fileupload", method = RequestMethod.GET)
public String form() {
    return "submissionform";
}
```



# 실습

## ❖ 파일 업로드 폼을 소유한 파일 작성(view 디렉토리의 submissionform.jsp)

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>리포트 제출</title>
</head>
<body>
    <h3>@RequestParam 사용</h3>
    <form action="fileupload1.action" method="post"
        enctype="multipart/form-data">
        학번: <input type="text" name="studentNumber" /> <br /> 리포트파일:
    <input
        type="file" name="report" /> <br /> <input type="submit" />
    </form>

    <h3>MultipartHttpServletRequest 사용</h3>
    <form action="fileupload2.action" method="post"
        enctype="multipart/form-data">
        학번: <input type="text" name="studentNumber" /> <br /> 리포트파일:
    <input
        type="file" name="report" /> <br /> <input type="submit" />
    </form>
```

# 실습

## ❖ 파일 업로드 폼을 소유한 파일 작성(view 디렉토리의 submissionform.jsp)

```
<h3>커맨드 객체 사용</h3>
<form action="fileupload3.action" method="post"
      enctype="multipart/form-data">
  학번: <input type="text" name="studentNumber" /> <br /> 리포트파일:
  <input
                                type="file" name="report" /> <br /> <input type="submit" />
  </form>
</body>
</html>
```

# 실습

## ❖ 파일 업로드 폼의 domain 클래스 생성(domain 패키지의 ReportCommand.java)

```
import org.springframework.web.multipart.MultipartFile;
```

```
import lombok.Data;
```

```
@Data
```

```
public class ReportCommand {  
    private String studentNumber;  
    private MultipartFile report;  
}
```

# 실습

## ❖ HomeController에 파일 업로드 폼의 전송 요청을 처리하는 메서드 작성

```
private void printInfo(String studentNumber, MultipartFile report) {  
    System.out.println(studentNumber + "가 업로드 한 파일: " +  
report.getOriginalFilename());  
}
```

```
@RequestMapping(value = "fileupload1.action", method = RequestMethod.POST)  
public String submitReport1(@RequestParam("studentNumber") String studentNumber,  
    @RequestParam("report") MultipartFile report) {  
    printInfo(studentNumber, report);  
    return "submissioncomplete";  
}
```

```
@RequestMapping(value = "fileupload2.action", method = RequestMethod.POST)  
public String submitReport2(MultipartHttpServletRequest request) {  
    String studentNumber = request.getParameter("studentNumber");  
    MultipartFile report = request.getFile("report");  
    printInfo(studentNumber, report);  
    return "submissioncomplete";  
}
```

# 실습

## ❖ HomeController에 파일 업로드 폼의 전송 요청을 처리하는 메서드 작성

```
@RequestMapping(value = "fileupload3.action", method = RequestMethod.POST)
public String submitReport3(ReportCommand reportCommand) {
    println(reportCommand.getStudentNumber(), reportCommand.getReport());
    if (reportCommand.getReport().isEmpty()) {
        System.out.println("업로드한 파일이 없습니다.");
    } else {
        //파일 업로드를 위한 디렉토리 - 자신의 디렉토리로 수정
        String filePath = "/Users/adam/Documents/data/";
        filePath += UUID.randomUUID() +
reportCommand.getReport().getOriginalFilename();
        System.out.println("filePath:" + filePath);
        File file = new File(filePath);
        FileOutputStream fos = null;
```



# 실습

## ❖ HomeController에 파일 업로드 폼의 전송 요청을 처리하는 메서드 작성

```
try {  
    fos = new FileOutputStream(file);  
} catch (FileNotFoundException e1) {}  
  
try {  
    fos.write(reportCommand.getReport().getBytes());  
    System.out.println("fos:" + fos);  
} catch (IllegalStateException e) {  
    e.printStackTrace();  
} catch (IOException e) {  
    System.out.println("전송 실패");  
}  
}  
return "submissioncomplete";  
}
```

# 실습

## ❖ 파일 업로드 결과를 출력할 파일 작성(view 디렉토리의 submissioncomplete.jsp)

```
<%@ page contentType="text/html; charset=UTF-8"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>리포트 제출 완료</title>
</head>
<body>
리포트 제출 완료
</body>
</html>
```



# 회원 관리

## ❖ 회원 정보 테이블 생성 - MySQL

```
CREATE TABLE SPRINGUSER(  
    EMAIL VARCHAR(100)PRIMARY KEY,  
    PW VARCHAR(100)not null,  
    NICKNAME VARCHAR(100)not null,  
    IMAGE VARCHAR(200) DEFAULT 'default.jpg',  
    REGDATE DATETIME DEFAULT NOW(),  
    LOGINDATE DATETIME DEFAULT NOW(),  
    EMAILAUTH INT DEFAULT 0  
);
```

```
INSERT INTO SPRINGUSER(EMAIL, PW, NICKNAME) values('ggangpae1@gmail.com', '123456',  
'군계')
```

```
COMMIT;
```

```
SELECT * FROM SPRINGUSER;
```

# 회원 관리

## ❖ 회원 정보 테이블 생성 - Oracle

```
CREATE TABLE SPRINGUSER(  
    EMAIL VARCHAR2(100)PRIMARY KEY,  
    PW VARCHAR2(100)not null,  
    NICKNAME VARCHAR2(100)not null,  
    IMAGE VARCHAR2(200) DEFAULT 'default.jpg',  
    REGDATE DATE DEFAULT sysdate,  
    LOGINDATE DATE DEFAULT sysdate,  
    EMAILAUTH NUMBER(1) DEFAULT 0  
);
```

-- 중복 검사를 위해서 삽입

```
INSERT INTO SPRINGUSER(EMAIL, PW, NICKNAME) values('ggangpae1@gmail.com', '123456',  
'군계');
```

```
COMMIT;
```

```
SELECT * FROM SPRINGUSER;
```

# 회원 관리

- ❖ pom.xml 파일에 비밀번호 암호화(복호화가 불가능)를 위한 라이브러리의 의존성을 설정

```
<dependency>
  <groupId>org.mindrot</groupId>
  <artifactId>jbcrypt</artifactId>
  <version>0.4</version>
</dependency>
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.15</version>
</dependency>
```

# 회원 관리를 준비

- ❖ util 패키지를 생성하고 암호화(복호화 가능) 클래스를 추가 – CryptoUtil.java
- ❖ webapp디렉토리에 유저의 이미지를 저장할 profile 디렉토리를 생성하고 기본 이미지(default.jpg)를 추가



# SpringUser 서비스를 위한 클래스 구조

## ❖ SpringUser 처리를 위한 DTO 클래스를 생성

```
import java.util.Date;
```

```
import lombok.Data;
```

```
@Data
```

```
public class SpringUser {  
    private String email;  
    private String pw;  
    private String nickname;  
    private String image;  
    private Date regdate;  
    private Date logindate;  
    private int emailauth;  
}
```

# SpringUser 서비스를 위한 클래스 구조

❖ dao 패키지에 SpringUserMapper 인터페이스를 생성

```
@Repository  
public interface SpringUserMapper {  
}
```



# SpringUser 서비스를 위한 클래스 구조

- ❖ **service** 패키지에 **SpringUserService** 인터페이스를 생성

```
public interface SpringUserService {  
  
}
```



# SpringUser 서비스를 위한 클래스 구조

## ❖ SpringUserServiceImpl 클래스를 생성

```
@Service
public class SpringUserServiceImpl implements SpringUserService {
    @Autowired
    private SpringUserMapper SpringUserDao;
}
```

# SpringUser 서비스를 위한 클래스 구조

## ❖ 페이지 이동을 처리할 **SpringUserController** 클래스를 생성

```
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Controller;
```

```
import 기본 패키지.service.SpringUserService;
```

```
@Controller  
public class SpringUserController {  
    @Autowired  
    private SpringUserService springUserService;  
  
}
```

# SpringUser 서비스를 위한 클래스 구조

- ❖ ajax 요청을 처리하기 위해서 데이터를 전송하는 **SpringUserRestController** 클래스를 생성

```
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.RestController;
```

```
@RestController  
public class SpringUserRestController {  
    @Autowired  
    private SpringUserService springUserService;  
}
```

# 회원가입

## 회원가입

이메일

비밀번호

비밀번호확인

별명

이미지



회원가입

메인

로그인

# 회원가입

## ❖ home.jsp 파일에 회원 가입 링크 만들기

```
<li><a href="user/join" class="menu">회원가입</a></li>
```



# 회원가입

- ❖ **UserPageController** 클래스에 회원가입 페이지로 이동하는 처리를 위한 메서드를 생성

```
@RequestMapping(value="user/join", method=RequestMethod.GET)
public String join() {
    return "user/join";
}
```

# 회원가입

## ❖ view 디렉토리에 user 디렉토리를 생성하고 join.jsp 파일을 생성한 후 작성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport"
    content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no"
/>
<link rel="stylesheet" href="${pageContext.request.contextPath}/css/user/join.css">
<script src="${pageContext.request.contextPath}/js/user/join.js"> </script>
<title>회원가입</title>
</head>
```



# 회원가입

## ❖ view 디렉토리에 user 디렉토리를 생성하고 join.jsp 파일을 생성한 후 작성

```
<body>
  <form id="joinform" enctype="multipart/form-data">
    <h2>회원가입</h2>
    <div align="center" id="msg"> </div>
    <ul>
      <li>
        <label for="email">이메일</label>
        <input type="text" id="email" name="email"
class="textinput" placeholder="이메일 경로를 입력하세요" />
      </li>
      <div id="emailmsg"> </div>
      <li>
        <label for="pw">비밀번호</label>
        <input type="password" id="pw" name="pw"
class="textinput" placeholder="비밀번호는 8-15자로 만들어야 합니다." />
      </li>
      <div id="pwmsg"> </div>
      <li>
        <label for="pw1">비밀번호확인</label>
        <input type="password" id="pw1" class="textinput"
placeholder="영문 대소문자와 특수문자와 숫자 1개 이상으로 만들어야 합니다." />
      </li>
```

# 회원가입

❖ view 디렉토리에 user 디렉토리를 생성하고 join.jsp 파일을 생성한 후 작성

```
<li>
    <label for="nickname"> 별명 </label>
    <input type="text" id="nickname" name="nickname"
class="textinput" />
</li>
<div id="nicknamemsg"> </div>
<li>
    <label for="image"> 이미지 </label>
    <input type="file" id="image" class="fileinput"
accept="image/*" name="image"/>
    <img id="img" />
</li>
<div align="center">
<li class="buttons">
    <input type="button" value="회원가입"
id="joinbtn"/>
    <input type="button" value="메인" id="mainbtn" />
    <input type="button" value="로그인" id="loginbtn"
/>
</li>
</div>
</ul>
</form>
```

# 회원가입

❖ view 디렉토리에 user 디렉토리를 생성하고 join.jsp 파일을 생성한 후 작성

```
</body>  
</html>
```



# 회원가입

❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
ul {  
    list-style-type: none;  
}  
  
ul li {  
    clear: both;  
}  
  
form {  
    width: 60em;  
    border: 1px solid #666;  
    border-radius: 10px;  
    box-shadow: .2em .2em .5em #999;  
    background-color: #d0e9f6;  
    padding: 1em;  
    overflow: hidden;  
}
```

# 회원가입

- ❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
label {  
    display: block;  
    float: left;  
    width: 10em;  
    text-align: right;  
    margin-right: .5em;  
    color: #04699d;  
}  
  
input.textinput {  
    width: 30em;  
    height: 2em;  
    border: 1px solid #666;  
    margin-bottom: 1em;  
    display: block;  
    float: left;  
}
```

# 회원가입

❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
input.fileinput {  
    width: 15em;  
    height: 2em;  
    border: 1px solid #666;  
    margin-bottom: 1em;  
    margin-right: 1em;  
    display: block;  
    float: left;  
}
```

```
img{  
    width: 7em;  
    height: 7em;  
    border: 1px solid #666;  
    margin-bottom: 1em;  
}
```

# 회원가입

- ❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
textarea {  
    display: block;  
    width: 30em;  
    height: 5em;  
    border: 1px solid #666;  
    margin-bottom: 1em;  
    line-height: 1.25;  
    overflow: auto;  
    resize: none;  
}
```

```
input.textinput, textarea {  
    font-family: Georgia, "Times New Roman", Times, serif;  
    font-size: .875em;  
}
```

# 회원가입

❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
input[type="submit"], input[type="button"] {  
    display: block;  
    width: 10em;  
    height: 2em;  
    float: left;  
    background: white;  
    font-size: inherit;  
    border: 1px solid #04699d;  
    border-radius: 4px;  
    box-shadow: 2px 2px 3px rgba(0, 0, 0, .5);  
    margin-top: 1em;  
    margin-right: 1em;  
    color: #C00;  
}  
  
input[type="submit"] {  
    margin-left: 5em;  
}
```



# 회원가입

- ❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
div {  
    color: red;  
    clear: both;  
    margin-left: 11em;  
    margin-bottom: 1em;  
}
```



# 회원가입

❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
@media screen and (max-width: 960px) {  
  form {  
    width: 20em;  
    border: 1px solid #666;  
    border-radius: 10px;  
    box-shadow: .2em .2em .5em #999;  
    background-color: #d0e9f6;  
    padding: 1em;  
    overflow: hidden;  
  }  
  
  label {  
    display: block;  
    float: left;  
    width: 4em;  
    text-align: right;  
    margin-right: .5em;  
    color: #04699d;  
  }  
}
```

# 회원가입

❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
input.textinput {  
    width: 13em;  
    height: 2em;  
    border: 1px solid #666;  
    margin-bottom: 1em;  
}  
  
img{  
    width: 7em;  
    height: 7em;  
    border: 1px solid #666;  
    margin-bottom: 1em;  
    margin-left: 5em;  
}
```

# 회원가입

❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
textarea {  
    display: block;  
    width: 13em;  
    height: 5em;  
    border: 1px solid #666;  
    margin-bottom: 1em;  
    line-height: 1.25;  
    overflow: auto;  
    resize: none;  
}  
  
input.textinput, textarea {  
    font-family: Georgia, "Times New Roman", Times, serif;  
    font-size: .875em;  
}
```

# 회원가입

❖ webapp/css 디렉토리에 user 디렉토리를 생성하고 join.css 파일을 생성한 후 작성

```
input[type="submit"], input[type="button"] {  
    display: block;  
    width: 10em;  
    height: 2em;  
    float: left;  
    background: white;  
    font-size: inherit;  
    border: 1px solid #04699d;  
    border-radius: 4px;  
    box-shadow: 2px 2px 3px rgba(0, 0, 0, .5);  
    margin-top: 1em;  
    color: #C00;  
    margin-left: 3em;  
}
```

```
}
```

# EMAIL 중복 체크

❖ **SpringUserMapper** 인터페이스에 **email** 중복 체크를 위한 **sql**을 작성

```
@Select("select email from SpringUser where email = #{email}")  
public List<String> emailCheck(String email);
```

# EMAIL 중복 체크

## ❖ Test 클래스에서 테스트

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "file:src/main/webapp/WEB-INF/spring/root-
context.xml"})
public class SampleTesting {

    @Autowired
    private SpringUserMapper springUserMapper;

    @Test
    public void testFactory() {
        System.out.println(springUserMapper);
        System.out.println(springUserMapper.emailCheck("ggangpae1@gmail.com"));
        System.out.println(springUserMapper.emailCheck("ggangpae2@gmail.com"));
    }
}
```

# NICKNAME 중복 체크

## ❖ SpringUserMapper 인터페이스에 nickname 중복 체크를 위한 sql을 작성

//닉네임 중복 처리를 위한 SQL

```
@Select("select nickname from SpringUser where nickname = #{nickname}")  
public String nicknameCheck(String nickname);
```





# NICKNAME 중복 체크

## ❖ Test 클래스에서 테스트

```
@Test
public void mybatistest() throws Exception{
    //존재하므로 nickname이 리턴
    System.out.println(sqlSession.selectOne("springuser.nicknamecheck", "군계"));
    //존재하지 않으므로 null 리턴
    System.out.println(sqlSession.selectOne("springuser.nicknamecheck", "싸움닭"));
}
```

# 회원가입

## ❖ SpringUserMapper 인터페이스에 회원가입을 위한 sql을 작성

//회원 가입을 위한 SQL

```
@Insert("insert into SpringUser(email, pw, nickname, image) "  
        + "values(#{email}, #{pw}, #{nickname}, #{image})")  
public int join(SpringUser springUser);
```



# 회원가입

## ❖ Test 클래스에서 테스트

```
@Test
public void mybatistest() throws Exception{
    //성공하면 1리턴
    SpringUser springUser = new SpringUser();
    springUser.setEmail("ggangpae2@gmail.com");
    springUser.setPw("1234");
    springUser.setNickname("싸움닭");
    springUser.setImage("default.jpg");
    System.out.println(springUserMapper.join(springUser));
}
```

# 회원가입

## ❖ SpringUserService 인터페이스에 회원가입을 위한 메서드를 선언

```
public Map<String, Object> join(MultipartHttpServletRequest request, HttpServletResponse response);
```



# 회원가입

## ❖ SpringUserServiceImpl 클래스에 회원가입을 위한 메서드를 구현

```
@Override
public Map<String, Object> join(MultipartHttpServletRequest request, HttpServletResponse
response) {
    // 결과 초기화
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("result", false);
    map.put("emailcheck", false);
    map.put("nicknamecheck", false);

    // Parameter 읽기
    String email = request.getParameter("email");
    String pw = request.getParameter("pw");
    String nickname = request.getParameter("nickname");
    String key = "itgangpae";
```

# 회원가입

## ❖ SpringUserServiceImpl 클래스에 회원가입을 위한 메서드를 구현

```
String emailResult = SpringUserDao.emailCheck(email);
if (emailResult == null) {
    map.put("emailcheck", true);
} else {
    map.put("emailcheck", false);
}

String nicknameResult = SpringUserDao.nicknameCheck(nickname);
if (nicknameResult == null) {
    map.put("nicknamecheck", true);
} else {
    map.put("nicknamecheck", false);
}
```

# 회원가입

## ❖ SpringUserServiceImpl 클래스에 회원가입을 위한 메서드를 구현

```
String image = "default.jpg";
if (emailResult == null && nicknameResult == null) {
    MultipartFile imageFile = request.getFile("image");
    if (imageFile.isEmpty() == false) {
        String filePath =
request.getServletContext().getRealPath("/profile");
        image = UUID.randomUUID() + imageFile.getOriginalFilename();
        filePath = filePath + "/" + image;
        File file = new File(filePath);
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(file);
        } catch (FileNotFoundException e1) {
        }
        try {
            fos.write(imageFile.getBytes());
        } catch (Exception e) {
            System.out.println("전송 실패");
        }
    }
}
```

# 회원가입

## ❖ SpringUserServiceImpl 클래스에 회원가입을 위한 메서드를 구현

```
SpringUser user = new SpringUser();
try {
    user.setEmail(CryptoUtil.encryptAES256(email, key));
    user.setPw(BCrypt.hashpw(pw, BCrypt.gensalt()));
    user.setNickname(nickname);
    user.setImage(image);
} catch (Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
int r = SpringUserDao.join(user);
if (r == 1) {
    map.put("result", true);
}
return map;
}
```



# 회원가입

## ❖ **UserRestController** 클래스에 회원가입 요청을 처리하는 메서드를 구현

```
@RequestMapping(value="user/join", method=RequestMethod.POST)
public Map<String, Object> join(MultipartHttpServletRequest request, HttpServletResponse
response) {
    return springUserService.join(request, response);
}
```

# 회원가입

❖ **join.jsp** 파일의 버튼 처리를 위한 스크립트 코드를 **webapp/js/user/join.js** 파일에 작성

```
window.addEventListener("load", function(event){  
    //DOM 객체 찾아오기  
    var joinform = document.getElementById("joinform");  
  
    var email = document.getElementById("email");  
    var pw = document.getElementById("pw");  
    var pw1 = document.getElementById("pw1");  
    var nickname = document.getElementById("nickname");  
    var image = document.getElementById("image");  
  
    var msg = document.getElementById("msg");  
    var emailmsg = document.getElementById("emailmsg");  
    var pwmsg = document.getElementById("pwmsg");  
    var nicknamemsg = document.getElementById("nicknamemsg");  
  
    var joinbtn = document.getElementById("joinbtn");  
    var mainbtn = document.getElementById("mainbtn");  
    var loginbtn = document.getElementById("loginbtn");
```

# 회원가입

❖ join.jsp 파일의 버튼 처리를 위한 스크립트 코드를 webapp/js/user/join.js 파일에 작성

```
mainbtn.addEventListener("click", function(event){
    location.href="../";
});

loginbtn.addEventListener("click", function(event){
    location.href="login";
});

image.addEventListener("change", function(event) {
    if (this.files && this.files[0]) {
        var filename = this.files[0].name;
        var reader = new FileReader();
        reader.addEventListener("load", function(e) {
            img.src = e.target.result;
        });
        reader.readAsDataURL(this.files[0]);
    }
})
```

# 회원가입

❖ join.jsp 파일의 버튼 처리를 위한 스크립트 코드를 webapp/js/user/join.js 파일에 작성

```
joinbtn.addEventListener("click", function(event){
    var flag = false;
    emailmsg.innerHTML = "";
    pwmsg.innerHTML = ""
    nicknamemsg.innerHTML = "";

    if (email.value.trim().length < 1) {
        emailmsg.innerHTML = '이메일은 필수 입력입니다.<br/>';
        flag = true;
    } else {
        var emailRegExp = /^[0-9a-zA-Z]([-_.]?[0-9a-zA-Z])*@[0-9a-zA-Z]([-_.]?[0-9a-zA-Z])*[a-zA-Z]{2,3}$/i;
        if (!emailRegExp.test(email.value.trim())) {
            emailmsg.innerHTML = '잘못된 이메일
형식입니다.<br/>';
            flag = true;
        }
    }
}
```

# 회원가입

❖ join.jsp 파일의 버튼 처리를 위한 스크립트 코드를 webapp/js/user/join.js 파일에 작성

```
if (pw.value.trim().length < 1) {  
    pwmsg.innerHTML += '비밀번호는 필수 입력입니다.<br/>';  
    flag = true;  
} else {  
    var pwRegExp = /^.*(?:^.{8,15}$)(?=.*\Wd)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#%^&+=]).*$/;  
    if (!pwRegExp.test(pw.value.trim())) {  
        pwmsg.innerHTML = '비밀번호는 숫자와  
영문 대소문자 그리고 특수문자가 포함되어야 합니다.<br/>';  
        flag = true;  
    } else {  
        if (pw.value.trim() !== pw1.value.trim()) {  
            pwmsg.innerHTML += '2개의 비밀번호는  
같아야 합니다.<br/>';  
            flag = true;  
        }  
    }  
}
```

# 회원가입

❖ join.jsp 파일의 버튼 처리를 위한 스크립트 코드를 webapp/js/user/join.js 파일에 작성

```
if (nickname.value.trim().length < 1) {  
    nicknamemsg.innerHTML += '별명은 필수 입력입니다.<br/>';  
    flag = true;  
} else {  
    var nicknameRegExp = /^[a-zA-z가-힣0-9]{2,5}$/;  
    if (!nicknameRegExp.test(nickname.value.trim())) {  
        nicknamemsg.innerHTML = '닉네임은 영문 한글  
숫자로 2자 이상 5자 이하이어야 합니다.<br/>';  
        flag = true;  
    }  
}  
  
if (flag == true) {  
    return;  
    event.preventDefault();  
}
```

# 회원가입

❖ join.jsp 파일의 버튼 처리를 위한 스크립트 코드를 webapp/js/user/join.js 파일에 작성

```
var url="join";  
var request=new XMLHttpRequest();
```

```
request.open("post", url, true);  
var formdata = new FormData(joinform);  
request.send(formdata);  
request.addEventListener('load', function show(e){  
    var map = JSON.parse(e.target.responseText);  
    if(map.result == true){  
        location.href = "../";  
    }else{
```

이메일입니다.;

이메일입니다.;

```
        if(map.emailcheck == false){  
            emailmsg.innerHTML = "사용 불 가능한  
  
            emailmsg.style.color = "Red";  
        }else{  
            emailmsg.innerHTML = "사용 가능한  
  
            emailmsg.style.color = "Blue";  
        }  
    }
```

# 회원가입

❖ join.jsp 파일의 버튼 처리를 위한 스크립트 코드를 webapp/js/user/join.js 파일에 작성

```
가능한 닉네임입니다.";
닉네임입니다.";

        if(map.nicknamecheck == false){
            nicknamemsg.innerHTML = "사용 불
        }else{
            nicknamemsg.innerHTML = "사용 가능한
            nicknamemsg.style.color = "Blue";
        }
    }
});
});
```



# FocusOut - EMAIL, NICKNAME 중복 검사

## ❖ SpringUserService 인터페이스에 2개의 메서드 선언

```
public Map<String, Object> emailcheck(HttpServletRequest request, HttpServletResponse response);
```

```
public Map<String, Object> nicknamecheck(HttpServletRequest request, HttpServletResponse response);
```



# FocusOut - EMAIL, NICKNAME 중복 검사

## ❖ SpringUserServiceImpl 클래스에 2개의 메서드 구현

```
@Override
public Map<String, Object> emailCheck(HttpServletRequest request, HttpServletResponse
response) {
    Map<String, Object> map = new HashMap<>();

    String email = request.getParameter("email");
    String result = springUserDao.emailCheck(email);

    if(result == null) {
        map.put("emailcheck", true);
    } else {
        map.put("emailcheck", false);
    }
    return map;
}
```

# FocusOut - EMAIL, NICKNAME 중복 검사

## ❖ SpringUserServiceImpl 클래스에 2개의 메서드 구현

```
@Override
public Map<String, Object> nicknameCheck(HttpServletRequest request, HttpServletResponse
response) {
    String nickname = request.getParameter("nickname");
    Map<String, Object> map = new HashMap<>();

    String nicknameResult = springUserDao.nicknameCheck(nickname);
    if (nicknameResult == null) {
        map.put("nicknamecheck", true);
    } else {
        map.put("nicknamecheck", false);
    }
    return map;
}
```

# FocusOut - EMAIL, NICKNAME 중복 검사

## ❖ UserRestController 클래스에 2개의 메서드 구현

```
@RequestMapping(value="user/emailcheck", method=RequestMethod.GET)
public Map<String, Object> emailcheck(HttpServletRequest request, HttpServletResponse
response) {
    return springUserService.emailCheck(request, response);
}

@RequestMapping(value="user/nicknamecheck", method=RequestMethod.GET)
public Map<String, Object> nicknamecheck(HttpServletRequest request, HttpServletResponse
response) {
    return springUserService.nicknameCheck(request, response);
}
```

# EMAIL, NICKNAME 중복 검사

## ❖ join.js에 파일에 이벤트 처리 코드를 추가

```
email.addEventListener("focusout", function(event){
    var url="emailcheck?email=" + email.value;
    var request=new XMLHttpRequest();
    request.open("get", url, true);
    request.send("");
    request.addEventListener('load', function(e){
        var map = JSON.parse(e.target.responseText);
        if(map.emailcheck == false){
            emailmsg.innerHTML = "사용 불 가능한 이메일입니다.";
            emailmsg.style.color = "Red";
        }else{
            emailmsg.innerHTML = "사용 가능한 이메일입니다.";
            emailmsg.style.color = "Blue";
        }
    })
})
```

# EMAIL, NICKNAME 중복 검사

## ❖ join.js에 파일에 이벤트 처리 코드를 추가

```
nickname.addEventListener("focusout", function(event){
    var url="nicknamecheck?nickname=" + nickname.value;
    var request=new XMLHttpRequest();
    request.open("get", url, true);
    request.send("");
    request.addEventListener('load', function show(e){
        var map = JSON.parse(e.target.responseText);
        if(map.nicknamecheck == false){
            nicknamemsg.innerHTML = "사용 불 가능한 별명입니다.";
            nicknamemsg.style.color = "Red";
        }else{
            nicknamemsg.innerHTML = "사용 가능한 별명입니다.";
            nicknamemsg.style.color = "Blue";
        }
    })
})
```

# 암호화 적용

## ❖ SpringUserMapper 인터페이스에 email 비교하는 SQL을 수정

//이메일 중복 확인을 위한 SQL

//이메일은 암호화를 했으므로 데이터베이스에서 직접 비교가 불가능해서 이메일을 리턴

@Select("select email from SpringUser")

public List<String> emailCheck();

# 암호화 적용

## ❖ SpringUserServiceImpl 클래스의 이메일 중복 검사 메서드 수정

```
@Override
public Map<String, Object> emailCheck(HttpServletRequest request, HttpServletResponse
response) {
    Map<String, Object> map = new HashMap<>();

    String email = request.getParameter("email");
    List<String> emailList = springUserDao.emailCheck();

    boolean flag = false;
    try {
        for(String encryptEmail : emailList) {
            if(email.equals(CryptoUtil.decryptAES256(encryptEmail,
"itggangpae")))) {
                flag = true;
                break;
            }
        }
    } catch (Exception e) {
        System.out.println(e.getLocalizedMessage());
    }
}
```



# 암호화 적용

## ❖ SpringUserServiceImpl 클래스의 이메일 중복 검사 메서드 수정

```
    if(flag == false) {  
        map.put("emailcheck", true);  
    } else {  
        map.put("emailcheck", false);  
    }  
    return map;  
}
```

# 암호화 적용

## ❖ SpringUserServiceImpl 클래스의 회원 가입 처리 메서드 수정

```
@Override  
public Map<String, Object> join(MultipartHttpServletRequest request, HttpServletResponse  
response) {
```

```
    // 결과 초기화
```

```
    Map<String, Object> map = new HashMap<String, Object>();  
    map.put("result", false);  
    map.put("emailcheck", false);  
    map.put("nicknamecheck", false);
```

```
    // Parameter 읽기
```

```
    String email = request.getParameter("email");  
    String pw = request.getParameter("pw");  
    String nickname = request.getParameter("nickname");  
    String key = "itggangpae";
```

# 암호화 적용

## ❖ SpringUserServiceImpl 클래스의 회원 가입 처리 메서드 수정

```
/*
String emailResult = springUserDao.emailCheck(email);
if (emailResult == null) {
    map.put("emailcheck", true);
} else {
    map.put("emailcheck", false);
}
*/
```

```
/*
String emailResult = springUserDao.emailCheck(email);

if (emailResult == null) {
    map.put("emailcheck", true);
} else {
    map.put("emailcheck", false);
}
*/
```

# 암호화 적용

## ❖ SpringUserServiceImpl 클래스의 회원 가입 처리 메서드 수정

```
List<String> emailList = springUserDao.emailCheck();  
boolean flag = false;
```

```
try {  
    for(String encryptEmail : emailList) {  
        if(email.equals(CryptoUtil.decryptAES256(encryptEmail,  
"itggangpae")))) {  
            flag = true;  
            break;  
        }  
    }  
} catch (Exception e) {  
    System.out.println(e.getLocalizedMessage());  
}
```

```
String nicknameResult = springUserDao.nicknameCheck(nickname);  
if (nicknameResult == null) {  
    map.put("nicknamecheck", true);  
} else {  
    map.put("nicknamecheck", false);  
}
```

# 암호화 적용

## ❖ SpringUserServiceImpl 클래스의 회원 가입 처리 메서드 수정

```
String image = "default.jpg";
if (flag == false && nicknameResult == null) {
    MultipartFile imageFile = request.getFile("image");
    if (imageFile.isEmpty() == false) {
        String filePath =
request.getServletContext().getRealPath("/profile");
        image = UUID.randomUUID() + imageFile.getOriginalFilename();
        filePath = filePath + "/" + image;
        File file = new File(filePath);
        FileOutputStream fos = null;
        try {
            fos = new FileOutputStream(file);
        } catch (FileNotFoundException e) {
            System.out.println("파일 생성 실패");
            System.out.println(e.getLocalizedMessage());
        }
        try {
            fos.write(imageFile.getBytes());
        } catch (Exception e) {
            System.out.println("전송 실패");
            System.out.println(e.getLocalizedMessage());
        }
    }
}
```

# 암호화 적용

## ❖ SpringUserServiceImpl 클래스의 회원 가입 처리 메서드 수정

```
SpringUser user = new SpringUser();
try {
    user.setEmail(CryptoUtil.encryptAES256(email, key));
    user.setPw(BCrypt.hashpw(pw, BCrypt.gensalt()));
    user.setNickname(nickname);
    user.setImage(image);
} catch (Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
int r = springUserDao.join(user);
if (r == 1) {
    map.put("result", true);
}
}
return map;
}
```

# 로그인

❖ **home.jsp** 파일에 로그인 요청을 위한 링크를 생성

```
<li><a href="user/login" class="menu">로그인</a></li>
```

# 로그인

❖ **UserPageController** 클래스에 로그인 요청을 처리하기 위한 메서드를 생성

```
@RequestMapping(value="user/login", method=RequestMethod.GET)
public String login() {
    return "user/login";
}
```



# 로그인

## ❖ user 디렉토리에 login.jsp 파일을 생성하고 작성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>로그인</title>
<meta name="viewport"
    content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no"
/>
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/css/user/login.css">
<script src="${pageContext.request.contextPath}/js/user/login.js"> </script>
</head>
```

# 로그인

❖ user 디렉토리에 login.jsp 파일을 생성하고 작성

```
<body>  
  <h1>로그인</h1>  
  <div class="inner_login">  
    <div class="login_myweb">
```

# 로그인

## ❖ user 디렉토리에 login.jsp 파일을 생성하고 작성

```
<div id="msg" class="msg"> </div>

<form method="post" id="loginform">
    <fieldset>
        <legend
class="screen_out">로그인 </legend>
        <div class="box_login">
            <div class="inp_text">
                <label for="email"
class="screen_out">이메일 </label> <input
                type="email" id="email" name="email" placeholder="이메일을 입력하세요!">
            </div>
            <div class="inp_text">
                <label for="pw"
class="screen_out">비밀번호 </label> <input
                type="password" id="pw" name="pw" placeholder="비밀번호를 입력하세요!">
            </div>
        </div>
    </fieldset>
</form>
```

# 로그인

## ❖ user 디렉토리에 login.jsp 파일을 생성하고 작성

```
<button class="btn_login" id="loginbtn">로그인 </button>

<div class="login_append">
    <div class="inp_chk">
        <input type="checkbox" id="keepLogin"
class="inp_radio"
        name="keepLogin"> <label
for="keepLogin" class="lab_g">
        <span class="img_top
ico_check"> </span> <span class="txt_lab">로그인
        상태 유지 </span>
        </label>
    </div>
</div>

<div class="login_append" align="center">
    <a href="#" class="link_find">아이디 찾기 </a> <a
href="#"
        class="link_find">비밀번호 찾기 </a> <a
href="join" class="link_find">회원가입 </a>
    <a href="../index.html"
class="link_find">메인으로 </a>
</div>
```

# 로그인

- ❖ user 디렉토리에 login.jsp 파일을 생성하고 작성

```
        </div>  
    </div>  
</body>  
</html>
```



# 로그인

❖ webapp/css/user 디렉토리에 login.css 파일을 생성하고 작성

```
@charset "UTF-8";  
h1{  
    text-align:center;  
}  
a {  
    color: #333;  
    text-decoration: none;  
}
```

# 로그인

## ❖ webapp/css/user 디렉토리에 login.css 파일을 생성하고 작성

```
input {  
    -webkit-writing-mode: horizontal-tb !important;  
    text-rendering: auto;  
    color: initial;  
    letter-spacing: normal;  
    word-spacing: normal;  
    text-transform: none;  
    text-indent: 0px;  
    text-shadow: none;  
    display: inline-block;  
    text-align: start;  
    -webkit-appearance: textfield;  
    background-color: white;  
    -webkit-rtl-ordering: logical;  
    cursor: text;  
    margin: 0em;  
    font: 400 13.3333px Arial;  
    padding: 1px 0px;  
    border-width: 2px;  
    border-style: inset;  
    border-color: initial;  
    border-image: initial;  
}
```

# 로그인

## ❖ webapp/css/user 디렉토리에 login.css 파일을 생성하고 작성

```
.inner_login {  
    position: absolute;  
    left: 50%;  
    top: 30%;  
    margin: -145px 0 0 -160px;  
}
```

```
.login_myweb {  
    position: relative;  
    width: 320px;  
    margin: 0 auto;  
}
```

```
.screen_out {  
    position: absolute;  
    width: 0;  
    height: 0;  
    overflow: hidden;  
    line-height: 0;  
    text-indent: -9999px;  
}
```



# 로그인

## ❖ webapp/css/user 디렉토리에 login.css 파일을 생성하고 작성

```
body, button, input, select, td, textarea, th {  
    font-size: 13px;  
    line-height: 1.5;  
    -webkit-font-smoothing: antialiased;  
}
```

```
fieldset, img {  
    border: 0;  
}
```

```
.login_myweb .box_login {  
    margin: 35px 0 0;  
    border: 1px solid #ddd;  
    border-radius: 3px;  
    background-color: #fff;  
    box-sizing: border-box;  
}
```

# 로그인

## ❖ webapp/css/user 디렉토리에 login.css 파일을 생성하고 작성

```
.login_myweb .inp_text {  
    position: relative;  
    width: 100%;  
    margin: 0;  
    padding: 18px 19px 19px;  
    box-sizing: border-box;  
}  
  
.login_myweb .inp_text+.inp_text {  
    border-top: 1px solid #ddd;  
}  
  
.inp_text input {  
    display: block;  
    width: 100%;  
    height: 100%;  
    font-size: 13px;  
    color: #000;  
    border: none;  
    outline: 0;  
    -webkit-appearance: none;  
    background-color: transparent;  
}
```

# 로그인

## ❖ webapp/css/user 디렉토리에 login.css 파일을 생성하고 작성

```
.btn_login {  
    margin: 20px 0 0;  
    width: 100%;  
    height: 48px;  
    border-radius: 3px;  
    font-size: 16px;  
    color: #fff;  
    background-color: #000;  
}
```

```
.login_append {  
    overflow: hidden;  
    padding: 15px 0 0;  
}
```

```
.inp_chk {  
    display: inline-block;  
    position: relative;  
    margin-bottom: -1px;  
}
```

# 로그인

## ❖ webapp/css/user 디렉토리에 login.css 파일을 생성하고 작성

```
.login_append .inp_chk {  
    float: left;  
}  
  
.inp_chk .inp_radio {  
    position: absolute;  
    z-index: -1;  
    top: 0;  
    left: 0;  
    width: 18px;  
    height: 18px;  
    border: 0;  
}  
  
.inp_chk .lab_g {  
    display: inline-block;  
    margin-right: 19px;  
    color: #909090;  
    font-size: 13px;  
    line-height: 19px;  
    vertical-align: top;  
}
```

# 로그인

## ❖ webapp/css/user 디렉토리에 login.css 파일을 생성하고 작성

```
.inp_chk .ico_check {  
    display: inline-block;  
    width: 18px;  
    height: 18px;  
    margin: 1px 4px 0 0;  
    background-position: -60px 0;  
    color: #333;  
}  
  
.inp_chk .txt_lab {  
    vertical-align: top;  
}  
  
.login_append .link_find {  
    float: left;  
    color: #777;  
    margin-left:.5em;  
    margin-right:.5em;  
}
```

# 로그인

❖ webapp/css/user 디렉토리에 login.css 파일을 생성하고 작성

```
.msg{  
    color:red;  
    text-align:center;  
    font-size:1.5em;  
}
```

# 로그인

## ❖ SpringUserMapper 인터페이스에 로그인 처리를 위한 SQL을 생성

//로그인 처리를 위한 메서드

```
@Select("select email, pw, nickname, image from SpringUser")
```

```
public List<SpringUser> login();
```



# 로그인

## ❖ SpringUserService 인터페이스에 로그인 처리를 위한 메서드를 선언

```
public Map<String, Object> login(HttpServletRequest request, HttpServletResponse response);
```





# 로그인

## ❖ SpringUserServiceImpl 클래스에 로그인 처리를 위한 메서드를 구현

```
@Override
public Map<String, Object> login(HttpServletRequest request, HttpServletResponse
response) {
    Map<String, Object> result = new HashMap<String, Object>();
    result.put("result", false);
    String email = request.getParameter("email");
    String pw = request.getParameter("pw");

    List<SpringUser> list = springUserDao.login();
    String key = "itgangpae";
```



# 로그인

❖ SpringUserServiceImpl 클래스에 로그인 처리를 위한 메서드를 구현

```
try {
    for(SpringUser user : list) {
        if(CryptoUtil.decryptAES256(user.getEmail(), key).equals(email)
        && BCrypt.checkpw(pw, user.getPw())) {
            result.put("result", true);
            result.put("email", email);
            result.put("nickname", user.getNickname());
            result.put("image", user.getImage());
            break;
        }
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
}
request.getSession().setAttribute("userinfo", result);
return result;
}
```

# 로그인

## ❖ UserRestController 클래스에 로그인 처리를 위한 메서드를 구현

```
@RequestMapping(value="user/login", method=RequestMethod.POST)
public Map<String, Object> login(HttpServletRequest request, HttpServletResponse
response) {
    Map<String, Object> result = springUserService.login(request, response);
    return result;
}
```



# 로그인

## ❖ login.js 파일을 생성하고 스크립트 코드를 작성

```
window.addEventListener("load", function(event){  
    //DOM 객체 찾아오기  
    var loginform = document.getElementById("loginform");  
    var loginbtn = document.getElementById("loginbtn");  
  
    var email = document.getElementById("email");  
    var pw = document.getElementById("pw");  
  
    var msg = document.getElementById("msg");
```



# 로그인

## ❖ login.js 파일을 생성하고 스크립트 코드를 작성

```
loginbtn.addEventListener("click", function(event){
    var flag = false;

    if (email.value.trim().length < 1) {
        msg.innerHTML = '이메일은 필수 입력입니다.<br/>';
        flag = true;
    }

    if (pw.value.trim().length < 1) {
        msg.innerHTML += '비밀번호는 필수 입력입니다.<br/>';
        flag = true;
    }

    if(flag == true){
        return;
    }
}
```



# 로그인

## ❖ login.js 파일을 생성하고 스크립트 코드를 작성

```
var url="login";
var request=new XMLHttpRequest();
request.open("post", url, true);
var formdata = new FormData(loginform);
request.send(formdata);
request.addEventListener('load', function(e){
    var map = JSON.parse(e.target.responseText);
    if(map.result == true){
        location.href = "../";
    }else{
        msg.innerHTML = "잘못된 이메일이거나 비밀번호
가 틀렸습니다.";
    }
});
});
```



# 로그인

## ❖ home.jsp 파일의 로그인 링크 수정

```
<c:if test="${userinfo.email == null}">
    <li> <a href="user/join" class="menu">회원가입</a> </li>
    <li> <a href="user/login" class="menu">로그인</a> </li>
</c:if>
<c:if test="${userinfo.email != null}">
    <li> ${userinfo.nickname}
님 <a href="user/logout" class="menu">로그아웃</a> </li>
    <li> <a href="user/update" class="menu">회원 정보 수정</a> </li>
</c:if>
```



# 로그아웃

## ❖ **UserPageController** 클래스에 로그아웃 처리를 위한 메서드를 생성

```
@RequestMapping(value="user/logout", method=RequestMethod.GET)
public String logout(HttpSession session) {
    session.invalidate();
    return "redirect:../";
}
```





# 회원정보 수정

- ❖ **UserPageController** 클래스에 회원정보 수정 처리를 위한 메서드를 생성

```
@RequestMapping(value="user/update", method=RequestMethod.GET)
public String update() {
    return "user/update";
}
```



# 회원정보 수정

## ❖ user 디렉토리에 update.jsp 파일을 생성하고 작성

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport"
    content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-
scalable=no" />
<link rel="stylesheet"
    href="${pageContext.request.contextPath}/css/user/join.css">
<script src="${pageContext.request.contextPath}/js/user/update.js"> </script>
<title>회원가입</title>
</head>
```



# 회원정보 수정

## ❖ user 디렉토리에 update.jsp 파일을 생성하고 작성

```
<body>
  <form id="updateform" enctype="multipart/form-data">
    <h2>정보 수정</h2>
    <div align="center" id="msg"> </div>
    <ul>
      <li><label for="email">이메일</label> <input type="text"
id="email"
                                name="email" class="textinput"
value="\${userinfo.email}"
                                readonly="readonly" /> </li>

      <li><label for="pw">비밀번호</label> <input
type="password" id="pw"
                                name="pw" class="textinput" placeholder="비밀
번호는 8-15자로 만들어야 합니다." />
                                </li>
      <div id="pwmsg"> </div>
      <li><label for="pw1">비밀번호확인</label> <input
type="password"
                                id="pw1" class="textinput"
                                placeholder="영문 대소문자와 특수문자와 숫자 1
개 이상으로 만들어야 합니다." /> </li>
```

# 회원정보 수정

## ❖ user 디렉토리에 update.jsp 파일을 생성하고 작성

```
<li> <label for="nickname">별명</label> <input type="text"
      id="nickname" name="nickname"
class="textinput" readonly="readonly"
      value="${userinfo.nickname}" /> </li>
<div id="nicknamemsg"> </div>
<li> <label for="image">이미지</label> <input type="file"
id="image"
      class="fileinput" accept="image/*" name="image"
/> 
</li>
<li class="buttons"> <input type="button" value="정보수정"
      id="updatebtn" /> <input type="button" value="
메인" id="mainbtn" /> <input
      type="button" value="로그인" id="loginbtn"
/> </li>
</ul>
</form>
</body>
</html>
```

# 회원정보 수정

## ❖ SpringUserMapper 인터페이스에 데이터 수정을 위한 sql을 작성

//회원정보 수정을 위한 메서드

```
@Update("update SpringUser set PW=#{pw}, IMAGE=#{image} where nickname =  
#{nickname}")
```

```
public int update(SpringUser springUser);
```



# 회원정보 수정

- ❖ **SpringUserDao** 클래스에 회원정보 수정을 위한 메서드를 작성

```
public int update(SpringUser springUser) {  
    return sqlSession.update("springuser.update", springUser) ;  
}
```



# 회원정보 수정

❖ **SpringUserService** 인터페이스에 회원정보 수정을 위한 메서드를 구현

```
public Map<String, Object> update(MultipartHttpServletRequest request,  
    HttpServletResponse response);
```



# 회원정보 수정

## ❖ SpringUserServiceImpl 클래스에 회원정보 수정을 위한 메서드를 구현

```
@Override
public Map<String, Object> update(MultipartHttpServletRequest request,
HttpServletResponse response) {
    // 결과 초기화
    Map<String, Object> map = new HashMap<String, Object>();
    map.put("result", false);

    // Parameter 읽기
    String email = request.getParameter("email");
    String pw = request.getParameter("pw");
    String nickname = request.getParameter("nickname");
    String image = (String)((HashMap)
request.getSession().getAttribute("userinfo")).get("image");
```

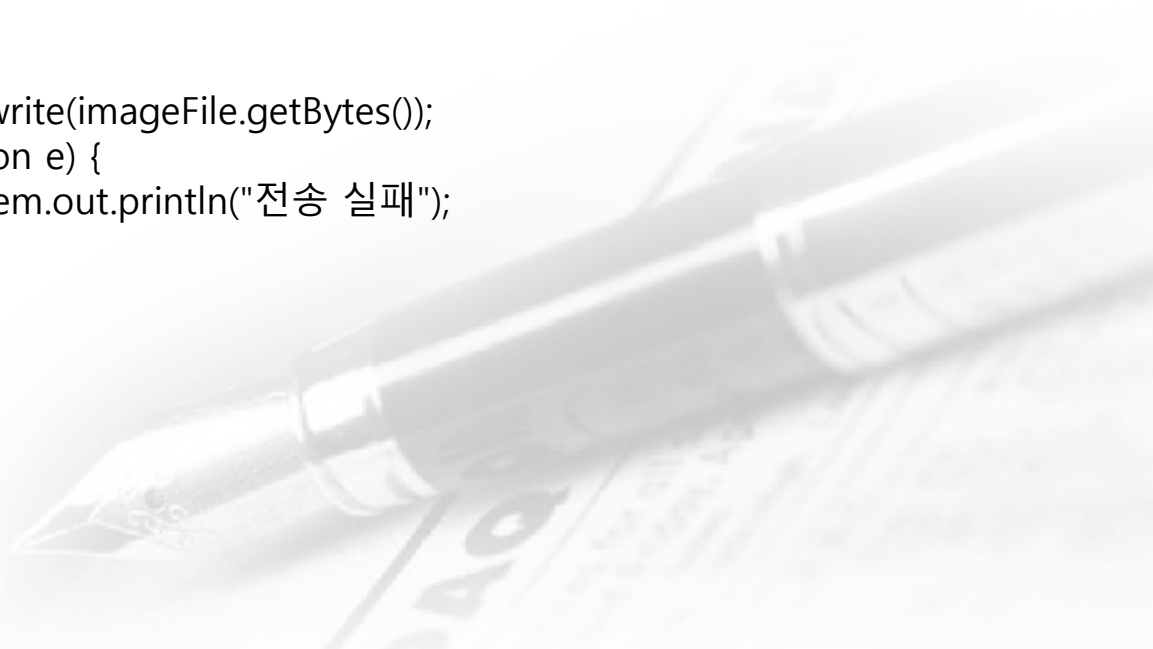




# 회원정보 수정

## ❖ SpringUserServiceImpl 클래스에 회원정보 수정을 위한 메서드를 구현

```
MultipartFile imageFile = request.getFile("image");
if (imageFile.isEmpty() == false) {
    String filePath = request.getServletContext().getRealPath("/profile");
    image = UUID.randomUUID() + imageFile.getOriginalFilename();
    filePath = filePath + "/" + image;
    File file = new File(filePath);
    FileOutputStream fos = null;
    try {
        fos = new FileOutputStream(file);
    } catch (FileNotFoundException e1) {
    }
    try {
        fos.write(imageFile.getBytes());
    } catch (Exception e) {
        System.out.println("전송 실패");
    }
}
```



# 회원정보 수정

## ❖ SpringUserServiceImpl 클래스에 회원정보 수정을 위한 메서드를 구현

```
SpringUser user = new SpringUser();
try {
    user.setEmail(email);
    user.setPw(BCrypt.hashpw(pw, BCrypt.gensalt()));
    user.setNickname(nickname);
    user.setImage(image);
} catch (Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
int r = springUserDao.update(user);
if (r == 1) {
    map.put("result", true);
}
return map;
}
```



# 회원정보 수정

## ❖ **UserRestController** 클래스에 회원정보 수정을 처리하기 위한 메서드를 생성

```
@RequestMapping(value="user/update", method=RequestMethod.POST)
public Map<String, Object> update(MultipartHttpServletRequest request,
HttpServletResponse response) {
    Map<String, Object> result = springUserService.update(request, response);
    request.getSession().invalidate();
    return result;
}
```



# 회원정보 수정

- ❖ webapp/js/user/update.js 파일을 생성하고 스크립트 코드 작성

```
window.addEventListener("load", function(event){  
    //DOM 객체 찾아오기  
    var updateform = document.getElementById("updateform");  
  
    var email = document.getElementById("email");  
    var pw = document.getElementById("pw");  
    var pw1 = document.getElementById("pw1");  
    var nickname = document.getElementById("nickname");  
    var image = document.getElementById("image");  
  
    var msg = document.getElementById("msg");  
    var pwmsg = document.getElementById("pwmsg");
```



# 회원정보 수정

- ❖ webapp/js/user/update.js 파일을 생성하고 스크립트 코드 작성

```
var updatebtn = document.getElementById("updatebtn");
var mainbtn = document.getElementById("mainbtn");
var loginbtn = document.getElementById("loginbtn");

mainbtn.addEventListener("click", function(event){
    location.href="../";
});

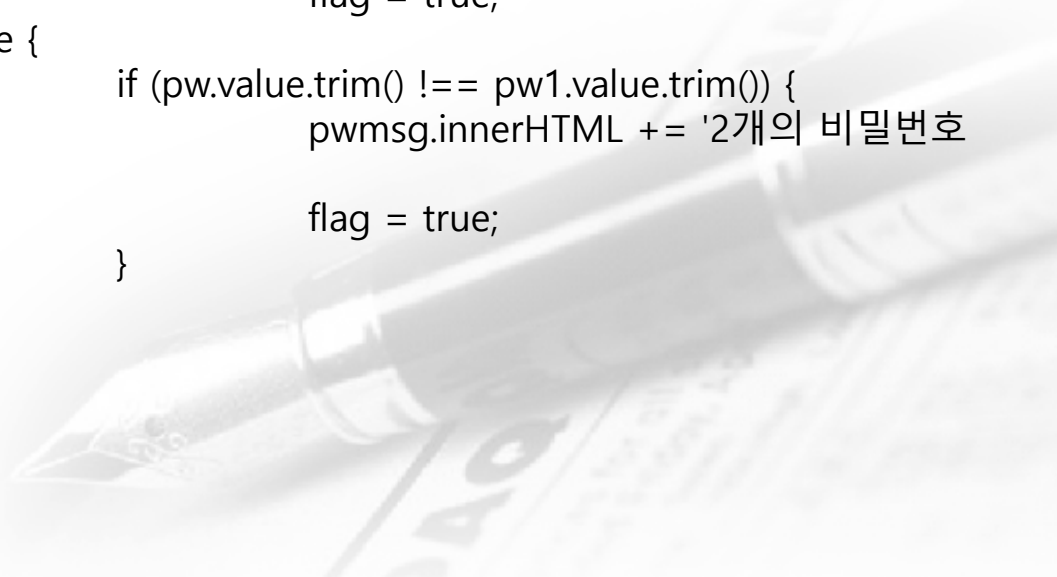
loginbtn.addEventListener("click", function(event){
    location.href="login";
});

image.addEventListener("change", function(event) {
    if (this.files && this.files[0]) {
        var filename = this.files[0].name;
        var reader = new FileReader();
        reader.addEventListener("load", function(e) {
            img.src = e.target.result;
        });
        reader.readAsDataURL(this.files[0]);
    }
})
```

# 회원정보 수정

- ❖ webapp/js/user/update.js 파일을 생성하고 스크립트 코드 작성

```
var flag = false;
updatebtn.addEventListener("click", function(event){
    if (pw.value.trim().length < 1) {
        pwmsg.innerHTML += '비밀번호는 필수 입력입니다.<br/>';
        flag = true;
    } else {
        var pwRegExp = /^(?=.*(?=^.{8,15}$)(?=.*\Wd)(?=.*[a-z])(?=.*[A-Z])(?=.*[!@#$%^&+=]).*$/;
        if (!pwRegExp.test(pw.value.trim())) {
            pwmsg.innerHTML = '비밀번호는 숫자
와 영문 대소문자 그리고 특수문자가 포함되어야 합니다.<br/>';
            flag = true;
        } else {
            if (pw.value.trim() !== pw1.value.trim()) {
                pwmsg.innerHTML += '2개의 비밀번호
는 같아야 합니다.<br/>';
                flag = true;
            }
        }
    }
}
```



# 회원정보 수정

- ❖ webapp/js/user/update.js 파일을 생성하고 스크립트 코드 작성

```
    if (flag == true) {  
        return;  
        event.preventDefault();  
    }  
  
    var url="update";  
    var request=new XMLHttpRequest();  
  
    request.open("post", url, true);  
    var formdata = new FormData(updateform);  
    request.send(formdata);  
    request.addEventListener('load', function(e){  
        var map = JSON.parse(e.target.responseText);  
        if(map.result == true){  
            location.href = "../";  
        }else{  
            msg.innerHTML = "수정 실패";  
        }  
    })  
});  
  
});
```