

Spring MVC

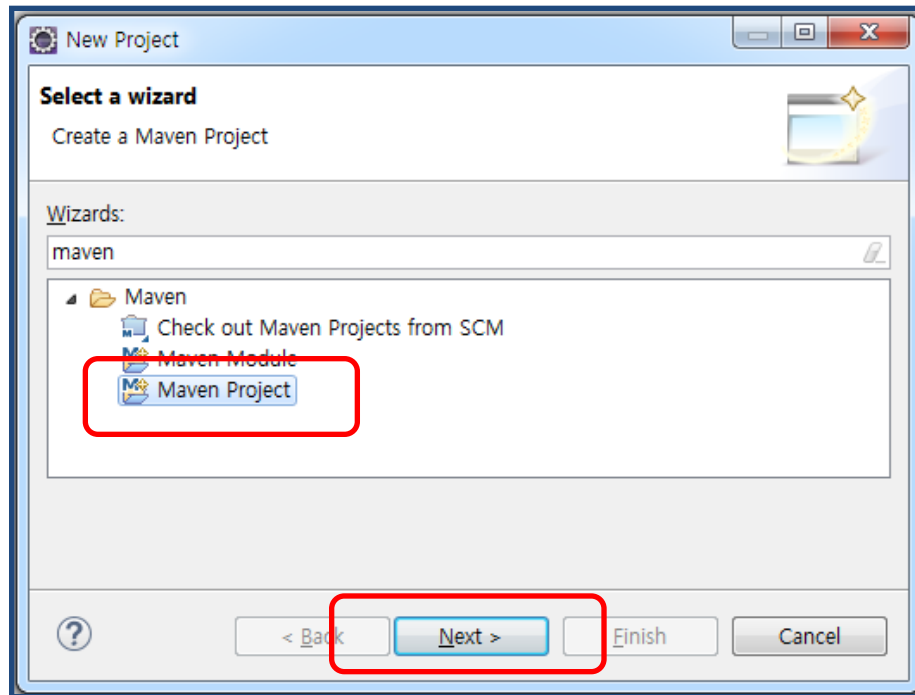
- 첫번째 프로젝트
- **@RequestMapping**
- **Form** 데이터 처리
- **@ResponseBody**
- **@RestController**
- 컨트롤러 없이 페이지 매핑
- 파일 업로드
- 인터셉터(interceptor)
- **@PathVariable**
-

첫번째 MVC 프로젝트

첫번째 만드는 MVC 프로젝트

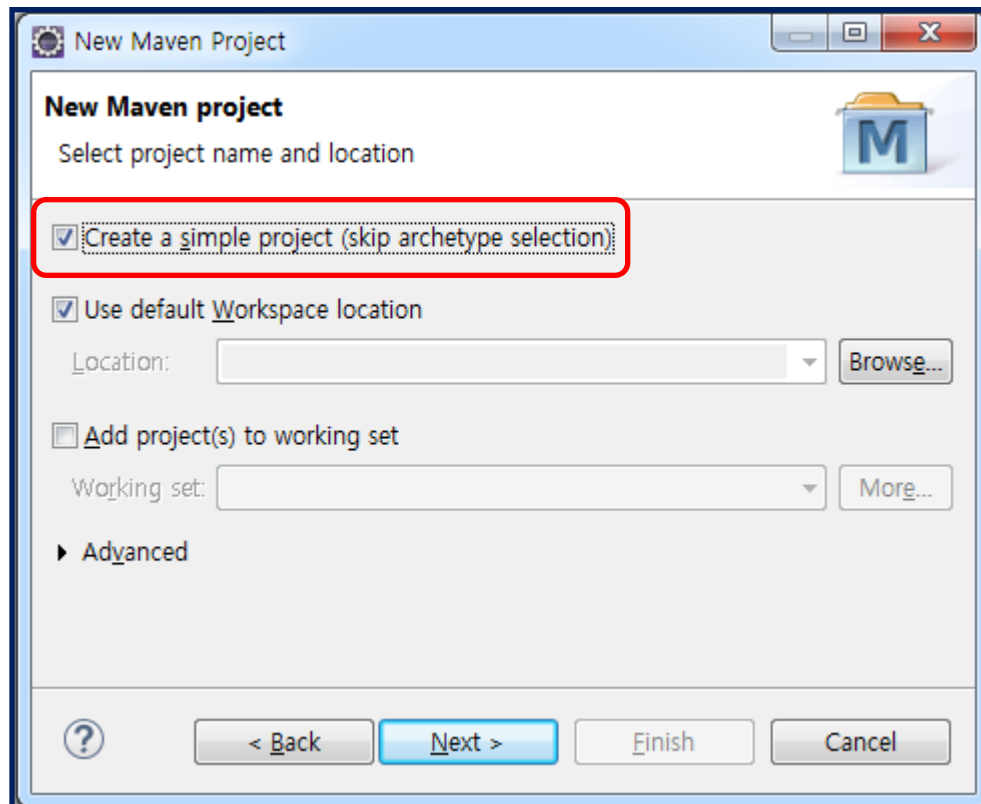
1. 프로젝트 생성

- Maven 프로젝트를 선택 후 Next 버튼을 클릭



첫번째 만드는 MVC 프로젝트

- Create a simple project 체크 후 Next 클릭



첫번째 만드는 MVC 프로젝트

- 입력창에 다음과 같이 입력 후 Finish 클릭

New Maven Project

Configure project

Artifact

Group Id: spring-mvc

Artifact Id: spring-mvc

Version: 0.0.1-SNAPSHOT

Packaging: war

Name: spring-mvc

Description: spring-mvc

Parent Project

Group Id:

Artifact Id:

Version: Browse... Clear

Advanced

< Back Next > Finish Cancel

첫번째 만드는 MVC 프로젝트

2. pom.xml

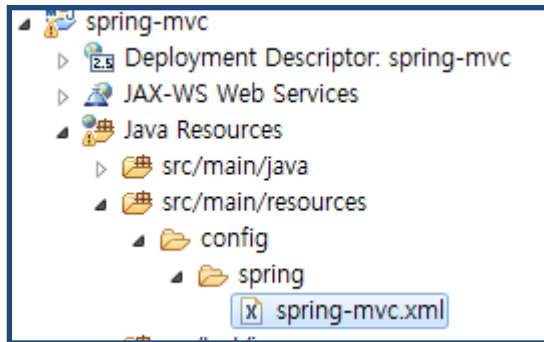
- dependency 추가

```
<dependencies>
  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.2</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
    <scope>provided</scope>
  </dependency>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.1.6.RELEASE</version>
  </dependency>
</dependencies>
```

첫번째 만드는 MVC 프로젝트

3. 스프링 설정 파일 작성

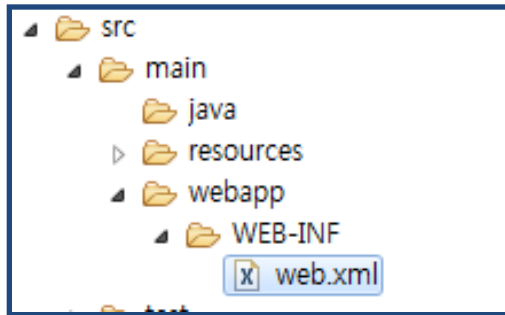
- src \ main \ resources 하위에 config \ spring 폴더 작성
- 제공된 spring-mvc.xml 파일 붙여 넣기



첫번째 만드는 MVC 프로젝트

4. 기본 디렉토리 생성

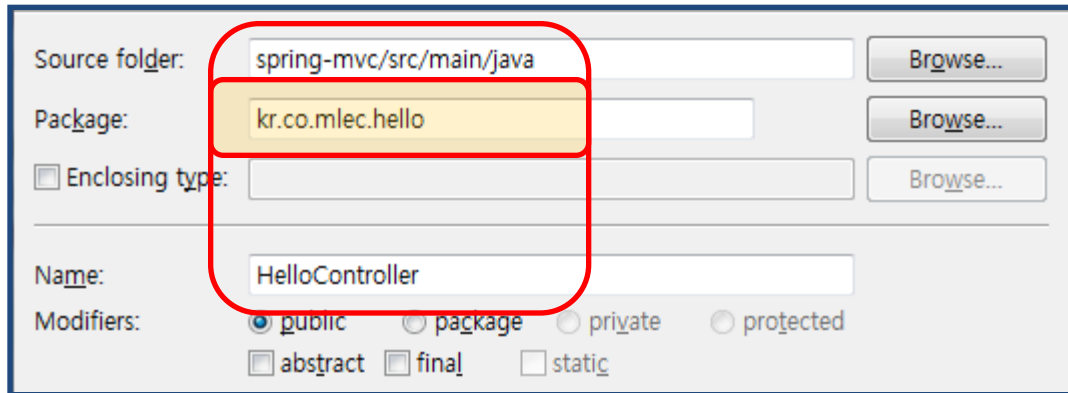
- / spring-mvc / src / main / webapp 에 WEB-INF 폴더 생성
- 제공된 web.xml 파일을 붙여넣기



첫번째 만드는 MVC 프로젝트

5. 컨트롤러 만들기

- / spring-mvc / src / main / java 폴더에 HelloController 생성



The screenshot shows a 'New Class' dialog box with the following fields and options:

- Source folder: spring-mvc/src/main/java
- Package: kr.co.mlec.hello
- Enclosing type: (empty)
- Name: HelloController
- Modifiers: ☒ public, ☐ package, ☐ private, ☐ protected, ☐ abstract, ☐ final, ☐ static

A red rectangle highlights the 'Source folder', 'Package', and 'Name' fields.

첫번째 만드는 MVC 프로젝트

- HelloController 소스 작성
-

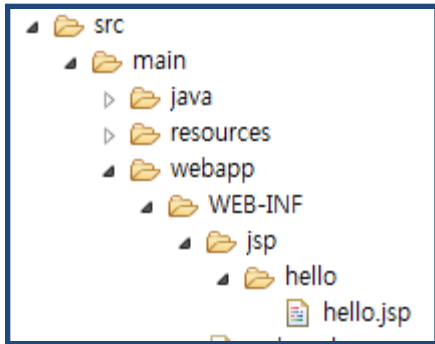
@Controller

```
public class HelloController {  
    @RequestMapping("/hello/hello.do")  
    public ModelAndView hello() {  
        ModelAndView mav = new ModelAndView("hello/hello");  
        mav.addObject("msg", "hi 스프링 MVC~~");  
        return mav;  
    }  
}
```

첫번째 만드는 MVC 프로젝트

6. 웹 페이지 만들기

- / spring-mvc / webapp / WEB-INF 에 jsp / hello 폴더 생성
- 생성된 폴더에 hello.jsp 생성



- 소스 작성

<body>

서버에서 받은 메시지 : \${msg}

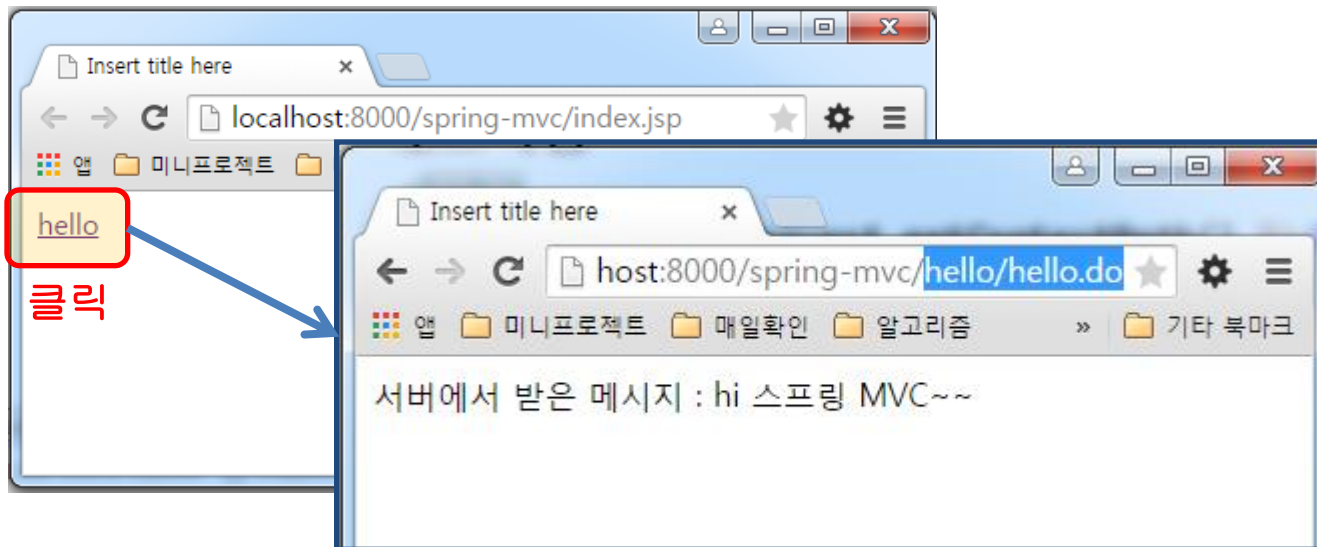
</body>

첫번째 만드는 MVC 프로젝트

7. 호출

- / spring-mvc / webapp 폴더에 index.jsp 파일 생성
- 소스 작성

```
<body>  
  <a href="<%=request.getContextPath() %>/hello/hello.do">  
    hello  
  </a><br />  
</body>
```



RequestMapping

RequestMapping

1. 컨트롤러 만들기

- / spring-mvc / src / main / java 폴더에 MethodController 생성

Source folder: spring-mvc/src/main/java

Package: kr.co.mlec.method

☐ Enclosing type:

Name: MethodController

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

RequestMapping

- MethodController 소스 작성

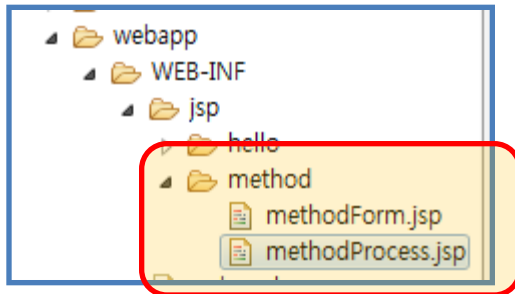
@Controller

```
public class MethodController {  
    @RequestMapping(value="/method/method.do",  
                    method=RequestMethod.GET)  
    public String callGet() {  
        return "method/methodForm";  
    }  
    @RequestMapping(value="/method/method.do",  
                    method=RequestMethod.POST)  
    public String callPost() {  
        return "method/methodProcess";  
    }  
}
```

RequestMapping

2. 웹 페이지 만들기

- / spring-mvc / webapp / WEB-INF / jsp 에 method 폴더 생성
- 생성된 폴더에 methodForm.jsp, methodProcess.jsp 생성



RequestMapping

- 코드 작성

methodForm.jsp

```
<body>
```

```
    <form action="<%= request.getContextPath() %>/method/method.do"
```

```
        method="POST">
```

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

```
    </form>
```

```
</body>
```

methodProcess.jsp

```
<body>
```

```
    <h1>MethodSpring 클래스에서 호출됨</h1>
```

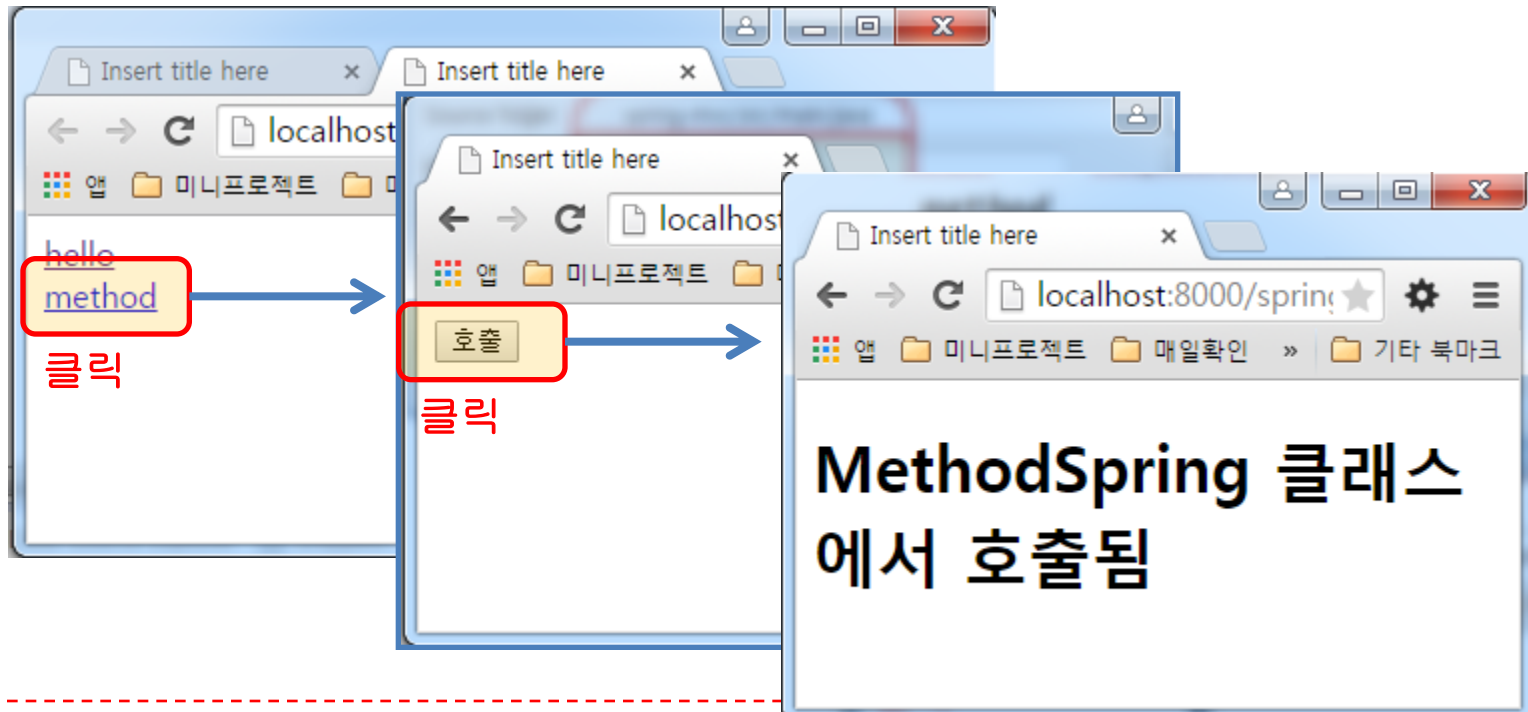
```
</body>
```

RequestMapping

3. 호출

- index.jsp 파일 코드 추가

```
<a href="<%=request.getContextPath() %>/method/method.do">  
    method  
</a><br />
```



RequestMapping

4. 컨트롤러 코드 변경

코드 생략

```
@RequestMapping(value="/method/method.do")
```

```
public class MethodController {
```

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

```
    public String callGet() {
```

코드 생략

```
}
```

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

```
    public String callPost() {
```

코드 생략

```
}
```

```
}
```

RequestMapping

5. 호출

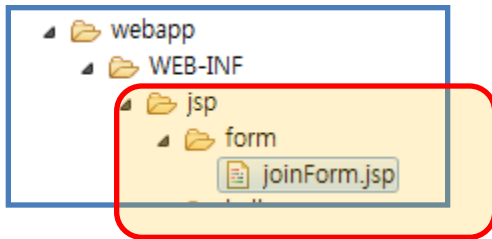
- index.jsp 파일을 실행하여 페이지 결과 확인

Form 데이터 처리

Form 데이터 처리

1. 웹 페이지 만들기

- / spring-mvc / webapp / WEB-INF / jsp 에 form 폴더 생성
- 생성된 폴더에 joinForm.jsp 생성



Form 데이터 처리

2. joinForm.jsp 코드 작성

```
<form action="<%= request.getContextPath() %>/form/join.do"
```

```
    method="POST">
```

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

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

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

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

```
</form>
```

Form 데이터 처리

3. 컨트롤러 만들기

- / spring-mvc / src / main / java 폴더에 MemberController 생성

Source folder: spring-mvc/src/main/java

Package: kr.co.mlec.form

☐ Enclosing type:

Name: MemberController

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Form 데이터 처리

- MemberController 소스 작성

```
@Controller
@RequestMapping("/form")
public class MemberController {
    @RequestMapping("/joinForm.do")
    public String joinForm() {
        return "form/joinForm";
    }
}
```

Form 데이터 처리

4. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="<%=request.getContextPath() %>/form/joinForm.do">  
    form  
</a><br />
```

Form 데이터 처리

5. VO 만들기

- / spring-mvc / src / main / java / form 폴더에 MemberVO 생성

```
private String id;
```

```
private String password;
```

```
private String name;
```

```
set, get 메서드 생성
```

Form 데이터 처리

6. MemberController 코드 추가

```
@RequestMapping("/join.do")
public String join(HttpServletRequest request) {
    String id = request.getParameter("id");
    String password = request.getParameter("password");
    String name = request.getParameter("name");
    MemberVO member = new MemberVO();
    member.setId(id);
    member.setPassword(password);
    member.setName(name);
    request.setAttribute("member", member);
    return "form/memberInfo";
}
```

Form 데이터 처리

7. form 폴더 하위 memberInfo.jsp 페이지 작성

`<body>`

`<h1>회원 정보</h1>`

`id : ${member.id}
`

`password : ${member.password}
`

`name : ${member.name}
`

`</body>`

- index.jsp 페이지 실행 후 테스트 진행
-

Form 데이터 처리

8. @RequestParam - MemberController 소스 수정

```
@RequestMapping("/join.do")
public String join(HttpServletRequest request,
                  @RequestParam("id") String id,
                  @RequestParam("password") String pass,
                  @RequestParam("name") String name) {
    MemberVO member = new MemberVO();
    member.setId(id);
    member.setPassword(pass);
    member.setName(name);
    request.setAttribute("member", member);
    return "form/memberInfo";
}
```

Form 데이터 처리

9. index.jsp 페이지 호출 후 테스트 진행

Form 데이터 처리

10. VO 객체 활용 - MemberController 소스 수정

```
@RequestMapping("/join.do")  
  
public String join(MemberVO member) {  
    System.out.println(member.getId());  
    System.out.println(member.getPassword());  
    System.out.println(member.getName());  
    return "form/memberInfo";  
}
```

Form 데이터 처리

11. form 폴더 하위 memberInfo.jsp 페이지 코드 수정

id : \${memberVO.id}

 password : \${memberVO.password}

 name : \${memberVO.name}

- index.jsp 페이지 실행 후 테스트 진행
-

Form 데이터 처리

12. @ModelAttribute - JSP에서 사용할 공유 객체명 변경

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

```
public String join(
```

```
    @ModelAttribute("member") MemberVO member) {
```

```
    System.out.println(member.getId());
```

```
    System.out.println(member.getPassword());
```

```
    System.out.println(member.getName());
```

```
    return "form/memberInfo";
```

```
}
```

Form 데이터 처리

13. form 폴더 하위 memberInfo.jsp 페이지 코드 수정

id : \${member.id}

 password : \${member.password}

 name : \${member.name}

- index.jsp 페이지 실행 후 테스트 진행
-

Form 데이터 처리

14. ModelAndView - view 페이지 및 view에서 사용할 객체 공유

```
@RequestMapping("/join.do")  
  
public ModelAndView join(MemberVO member) {  
    ModelAndView mav = new ModelAndView();  
    mav.setViewName("form/memberInfo");  
    mav.addObject("member", member);  
    return mav;  
}
```

- index.jsp 페이지 실행 후 테스트 진행
-

Form 데이터 처리

15. Model 객체 및 redirect 사용

```
@RequestMapping("/join.do")  
  
public String join(MemberVO member, Model model) {  
    System.out.println(member.getId());  
    System.out.println(member.getPassword());  
    System.out.println(member.getName());  
    model.addAttribute("msg", "가입이 완료되었습니다.");  
    return "form/joinForm";  
    // return "redirect:/form/joinForm.do";  
}
```

Form 데이터 처리

16. joinForm.jsp 페이지 코드 수정

```
<script>  
    if ("${msg}") {  
        alert("${msg}");  
    }  
</script>
```

- index.jsp 페이지 실행 후 테스트 진행
-

Form 데이터 처리

17. redirect 시 1회성 데이터 전송하기

```
@RequestMapping("/join.do")
public String join(
    MemberVO member, RedirectAttributes attr) {
    System.out.println(member.getId());
    System.out.println(member.getPassword());
    System.out.println(member.getName());
    attr.addFlashAttribute("msg", "가입이 완료되었습니다.");
    return "redirect:/form/joinForm.do";
}
```

- index.jsp 페이지 실행 후 테스트 진행

@ResponseBody

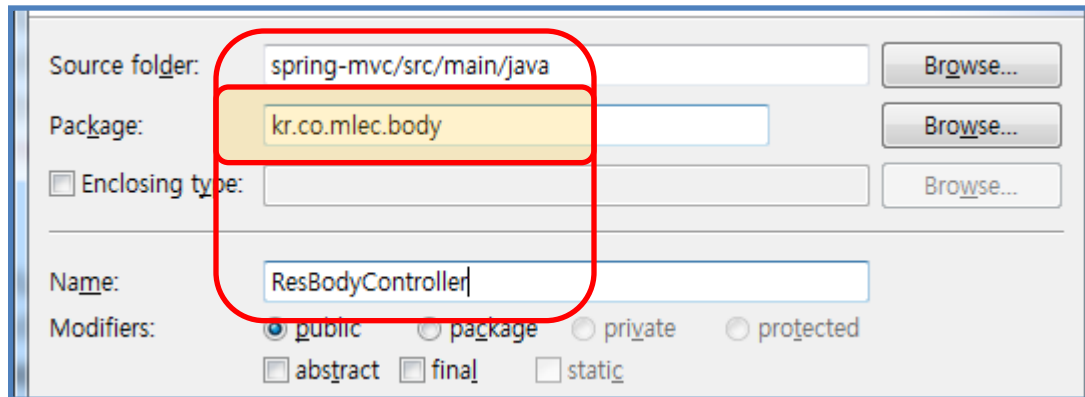
@ResponseBody

@ResponseBody 는 xml 또는 json 과 같은 메시지 기반의 서비스를 만들 경우 사용

예> AJAX 서비스를 제공하는 컨트롤러

1. 컨트롤러 만들기

- / spring-mvc / src / main / java 폴더에 ResBodyController 생성



@ResponseBody

2. 코드 작성

```
@Controller
@RequestMapping("/ajax")
public class ResBodyController {
    @RequestMapping("/resBody.do")
    @ResponseBody
    public String resStringBody() {
        return "OK, 성공";
    }
}
```

@ResponseBody

3. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="<%=request.getContextPath() %>/ajax/resBody.do">  
    문자열 응답  
</a><br />
```

@ResponseBody

4. 한글처리

- spring-mvc.xml 파일 수정 후 페이지 호출 테스트

```
<mvc:annotation-driven>
```

```
  <mvc:message-converters>
```

```
    <bean class=
```

```
      "org.springframework.http.converter.StringHttpMessageConverter">
```

```
      <property name="supportedMediaTypes">
```

```
        <list>
```

```
          <value>text/html; charset=UTF-8</value>
```

```
        </list>
```

```
      </property>
```

```
    </bean>
```

```
  </mvc:message-converters>
```

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

@ResponseBody

5. JSON 응답 처리

- ResBodyController 코드 추가

```
@RequestMapping("/resBody.json")
```

```
@ResponseBody
```

```
public Map<String, String> resJsonBody() {  
    Map<String, String> result = new HashMap<>();  
    result.put("id", "sbc");  
    result.put("name", "hong");  
    result.put("addr", "서울");  
    return result;  
}
```

@ResponseBody

6. JSON 처리 컨버터 등록

- spring-mvc.xml 파일 수정
- `mvc:message-converters`의 하위 태그로 등록

```
<bean class="org.springframework.http.converter
        .json.MappingJackson2HttpMessageConverter">
  <property name="supportedMediaTypes">
    <list>
      <value>text/html; charset=UTF-8</value>
      <value>application/json; charset=UTF-8</value>
    </list>
  </property>
</bean>
```

@ResponseBody

7. *MappingJackson2HttpMessageConverter* 에서 사용하는 라이브러리 다운로드

- pom.xml 파일 dependency 추가

```
<dependency>  
    <groupId>com.fasterxml.jackson.core</groupId>  
    <artifactId>jackson-core</artifactId>  
    <version>2.5.3</version>  
</dependency>
```

```
<dependency>  
    <groupId>com.fasterxml.jackson.core</groupId>  
    <artifactId>jackson-annotations</artifactId>  
    <version>2.5.3</version>  
</dependency>
```

```
<dependency>  
    <groupId>com.fasterxml.jackson.core</groupId>  
    <artifactId>jackson-databind</artifactId>  
    <version>2.5.3</version>  
</dependency>
```


@ResponseBody

8. JSON URL 매핑 등록

- web.xml 파일 수정

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

@ResponseBody

9. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="<%=request.getContextPath() %>/ajax/resBody.json">  
    JSON 응답  
</a><br />
```

@ResponseBody

10. JSON 응답 처리 (VO)

- ResBodyController 코드 추가

```
@RequestMapping("/resV0Body.json")
@ResponseBody
public MemberVO resJsonV0Body() {
    MemberVO vo = new MemberVO();
    vo.setId("sbc");
    vo.setName("sbc");
    vo.setPassword("1234");
    return vo;
}
```

@ResponseBody

11. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="<%=request.getContextPath() %>/ajax/resVOBody.json">  
    JSON VO 응답  
</a><br />
```

@ResponseBody

12. JSON 응답 처리 (List<String>)

- ResBodyController 코드 추가

```
@RequestMapping("/resStringListBody.json")
@ResponseBody
public List<String> resJsonStringListBody() {
    List<String> list = new ArrayList<>();
    for (int i = 1; i < 4; i++) {
        list.add(String.valueOf(i));
    }
    return list;
}
```

@ResponseBody

13. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="...../ajax/resStringListBody.json">  
    JSON List(문자열) 응답  
</a><br />
```

@ResponseBody

14. JSON 응답 처리 (List<VO>)

- ResBodyController 코드 추가

```
@RequestMapping("/resV0ListBody.json")
```

```
@ResponseBody
```

```
public List<MemberVO> resJsonV0ListBody() {  
    List<MemberVO> list = new ArrayList<>();  
  
    for (int i = 1; i < 4; i++) {  
        MemberVO vo = new MemberVO();  
        vo.setId("sbc");  
        vo.setName("sbc");  
        vo.setPassword("1234");  
        list.add(vo);  
    }  
    return list;  
}
```

@ResponseBody

15. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="...../ajax/resVOListBody.json">  
    JSON List(VO) 응답  
</a><br />
```


@RestController

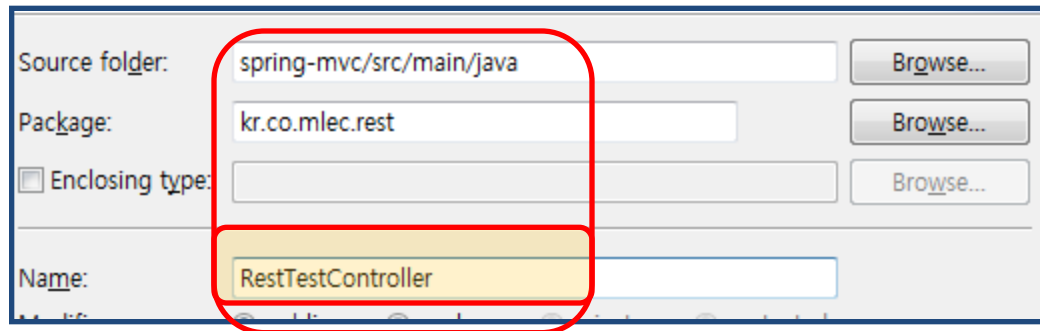
@RestController

@RestController는 rest 방식의 메시지 기반의 서비스를 만들 경우 사용
스프링 4 부터 지원함

예> AJAX 서비스를 제공하는 컨트롤러

1. 컨트롤러 만들기

- **kr.co.mlec.rest.RestBodyController** 생성



@RestController

2. 코드 작성

```
@RestController
@RequestMapping("/rest")
public class RestTestController {
    @RequestMapping("/resBody.do")
    public String resStringBody() {
        return "OK, 성공";
    }
}
```

- 기존 작성된 ResBodyController 클래스의 메서드를 복사한 후 붙여넣기 한다.
 - 복사한 메서드에서 @ResponseBody 어노테이션을 제거
-

@RestController

3. 호출

- index.jsp 파일에 아래의 링크 추가 후 테스트
- @ResponseBody 테스트 링크를 복사한 후 ajax -> rest 로 변경하여 추가

```
<a href="/생략/rest/resBody.do">RestController 문자열 응답</a>
```

```
<a href="/생략/rest/resBody.json">
```

```
RestController JSON 응답</a><br />
```

```
<a href="/생략/rest/resVOBody.json">
```

```
RestController JSON VO 응답</a><br />
```

```
<a href="/생략/rest/resStringListBody.json">
```

```
RestController JSON List(문자열) 응답</a><br />
```

```
<a href="/생략/rest/resVOListBody.json">
```

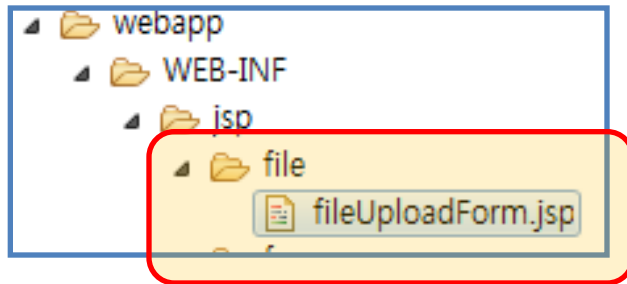
```
RestController JSON List(VO) 응답</a><br />
```

컨트롤러 없이 페이지 매핑

컨트롤러 없이 페이지 매핑

1. 웹 페이지 만들기

- / spring-mvc / webapp / WEB-INF / jsp 에 file 폴더 생성
- 생성된 폴더에 fileUploadForm.jsp 생성



컨트롤러 없이 페이지 매핑

2. fileUploadForm.jsp 코드 작성

```
<h2>파일 업로드 테스트</h2>
```

```
<form method="post" enctype="multipart/form-data"  
      action="...../file/upload.do">
```

```
  <input type="text" name="id" value="test" /><br />
```

```
  <input type="file" name="attachFile1" /><br />
```

```
  <input type="file" name="attachFile2" /><br />
```

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

```
</form>
```

컨트롤러 없이 페이지 매핑

3. spring-mvc.xml 파일에 내용 추가

```
<mvc:view-controller path="/file/uploadForm.do"  
view-name="file/fileUploadForm"/>
```

컨트롤러 없이 페이지 매핑

4. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href="...../file/uploadForm.do">  
    파일 업로드  
</a><br />
```

파일 업로드

파일 업로드

1. spring-mvc.xml 파일에 내용 추가

```
<bean id="multipartResolver"
      class="org.springframework.web.multipart
              .commons.CommonsMultipartResolver">
    <!-- 최대 업로드 파일 사이즈 : 10MB -->
    <property name="maxUploadSize" value="10485760" />
</bean>
```

파일 업로드

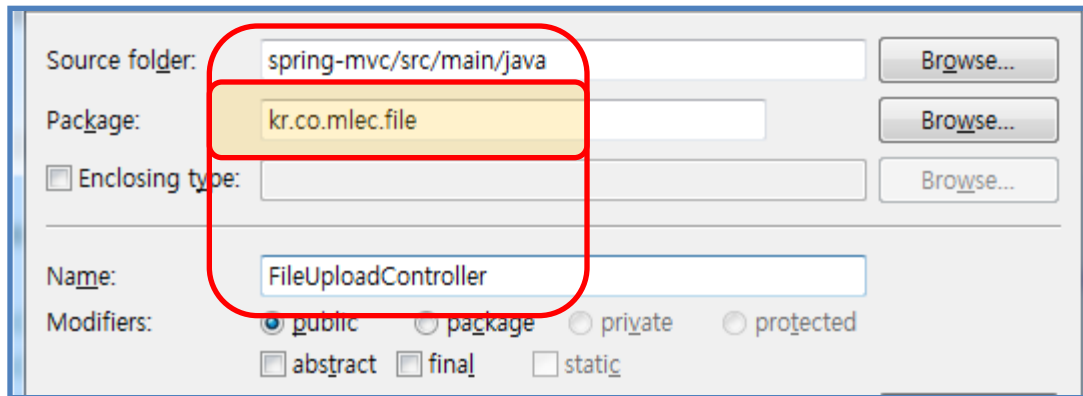
2. pom.xml 파일 dependency 추가

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

파일 업로드

3. 컨트롤러 만들기

- / spring-mvc / src / main / java 폴더에 UploadController 생성

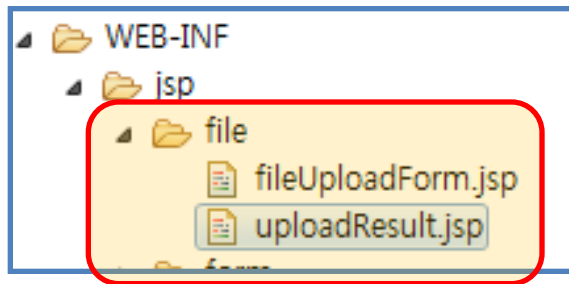


- 제공된 UploadController 자바파일의 내용을 복사해서 붙여넣기 한다.
-

파일 업로드

4. 결과 페이지 만들기

- / spring-mvc / webapp / WEB-INF / jsp 에 file 폴더에 uploadResult.jsp 생성



- 코드 작성

<body>

파일 업로드 성공

</body>

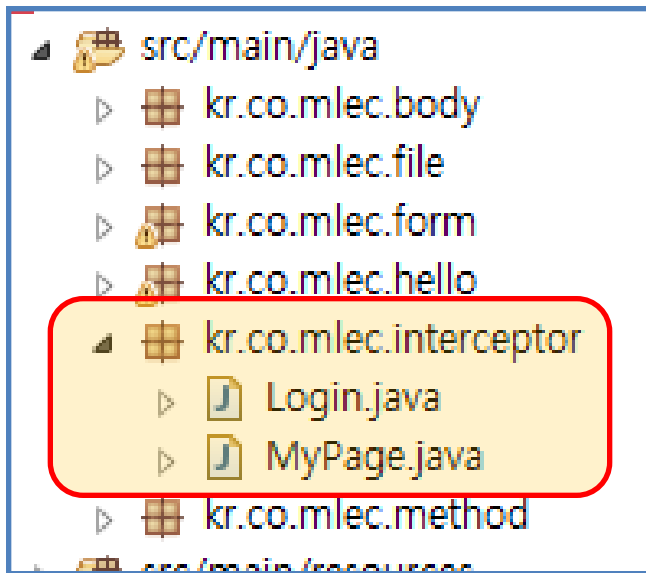
- index.jsp 페이지를 실행하여 결과 확인
-

인터셉터

인터셉터 이해하기

1. src 폴더에 파일 복사하기

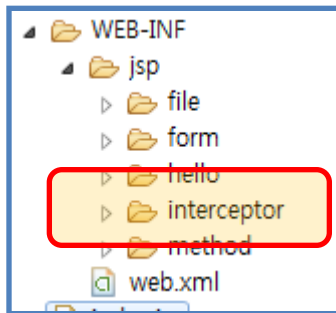
- 제공된 파일 중 src 폴더 하위의 kr 폴더를 복사
- 복사한 폴더를 프로젝트의 src 폴더에 붙여



인터셉터 이해하기

2. jsp 폴더에 파일 복사하기

- 제공된 파일 중 jsp 폴더 하위의 interceptor 폴더를 복사
- 복사한 폴더를 프로젝트의 jsp폴더에 붙여



인터셉터 이해하기

3. index.jsp 파일에 링크 추가 후 실행하여 동작 결과 확인

- `<a "... /interceptor/LoginForm.do">로그인폼
`

인터셉터 이해하기

4. MyPage.java 로그인 확인 코드 추가

```
@RequestMapping("/myPage.do")  
  
public String info(HttpSession session, Model model) {  
    MemberVO member = (MemberVO)session.getAttribute("user");  
    if (member == null) {  
        return "redirect:loginForm.do";  
    }  
    model.addAttribute("msg", "반갑습니다");  
    return "interceptor/myPage";  
}
```

모든 로그인
확인이 필요한 경우
동일한 코드가
반복된다.

인터셉터 이해하기

5. index.jsp 파일 실행하여 동작 결과 확인

- `<a "... /interceptor/LoginForm.do">로그인폼
`

인터셉터 작성하기

6. AuthInterceptor 작성하기

- interceptor 패키지 하위에 AuthInterceptor 클래스 생성

The image shows a 'New Class' dialog box from an IDE. The 'Source folder' is set to 'spring-mvc/src/main/java'. The 'Package' is set to 'kr.co.mlec.interceptor', which is highlighted with a yellow background and a red rounded rectangle. The 'Enclosing type' checkbox is unchecked. The 'Name' is set to 'AuthInterceptor', also highlighted with a red rounded rectangle.

Source folder:	spring-mvc/src/main/java
Package:	kr.co.mlec.interceptor
<input type="checkbox"/> Enclosing type:	
Name:	AuthInterceptor

인터셉터 작성하기

- 코드 작성

```
public class AuthInterceptor extends HandlerInterceptorAdapter {  
    @Override  
    public boolean preHandle(HttpServletRequest request,  
        HttpServletResponse response, Object handler)  
        throws Exception {  
        HttpSession session = request.getSession();  
        MemberVO member = (MemberVO)session.getAttribute("user");  
        if (member != null) {  
            return true;  
        }  
        response.sendRedirect("loginForm.do");  
        return false;  
    }  
}
```

인터셉터 이해하기

7. MyPage.java 로그인 확인 코드 삭제

```
@RequestMapping("/myPage.do")  
  
public void info(HttpSession session, Model model) {  
    model.addAttribute("msg", "반갑습니다");  
}
```

인터셉터 설정하기

8. AuthInterceptor 설정하기

- spring-mvc\src\main\resources\config\spring\spring-mvc.xml

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

```
<mvc:interceptors>
```

```
  <mvc:interceptor>
```

```
    <mvc:mapping path="/interceptor/**" />
```

```
    <mvc:exclude-mapping path="/interceptor/login*.do" />
```

```
    <bean class="kr.co.mlec.interceptor.AuthInterceptor" />
```

```
  </mvc:interceptor>
```

```
</mvc:interceptors>
```

```
<mvc:view-controller path="/file/uploadForm.do"  
                      view-name="file/fileUploadForm" />
```

Ant 경로 패턴 이해하기

- * : 0개 이상의 글자
- ? : 1개 글자
- ** : 0 개 이상의 디렉토리를 표현

예>

```
<mvc:mapping path="/interceptor/**" />
```

/interceptor 로 시작하는 모든 경로에 인터셉터 적용

```
<mvc:exclude-mapping path="/interceptor/Login*.do" />
```

/interceptor 하위의 login으로 시작하는 경로는 인터셉터 제외

인터셉터 이해하기

9. index.jsp 파일 실행하여 동작 결과 확인

- `<a "... /interceptor/LoginForm.do">로그인폼
`

@PathVariable

@PathVariable

@PathVariable는 Rest에서 많이 사용되는 방식으로 경로의 특정 부분을 변수 처럼 사용하게 하여 가변적인 값의 사용이 가능
특정 부분 파라미터를 대체하기도 함

1. 컨트롤러 만들기

-

Source folder: spring-mvc/src/main/java

Package: kr.co.mlec.path

☐ Enclosing type:

Name: PathVariableController

테스트 코드 작성하기

- 기본 파라미터 접근 방식

```
@RequestMapping("/paramMemberInfo.do")  
  
public void paramMemberinfo(String id, Model model) {  
    Map<String, MemberVO> data = new HashMap<>();  
    MemberVO m1 = new MemberVO();  
    m1.setId("a 사용자");  
    data.put("a", m1);  
    MemberVO m2 = new MemberVO();  
    m2.setId("b 사용자");  
    data.put("b", m2);  
    model.addAttribute("member", data.get(id));  
}
```

테스트 코드 작성하기

- PathVariable 접근 방식

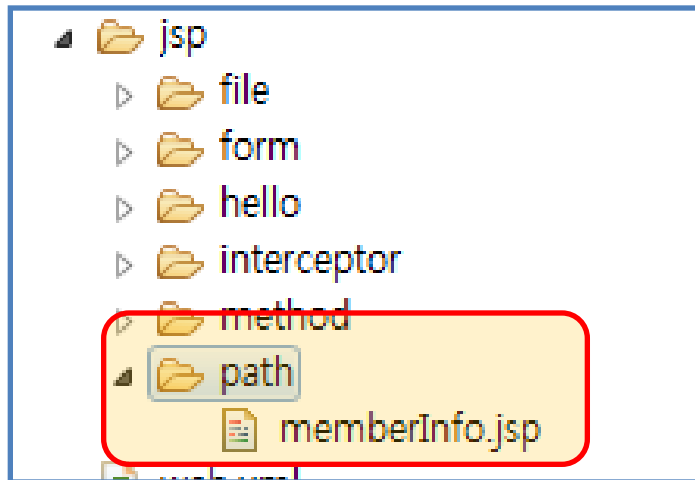
```
@RequestMapping("/{id}/pathMemberInfo.do")
public String pathMemberInfo(@PathVariable String id, Model model) {
    Map<String, MemberVO> data = new HashMap<>();
    MemberVO m1 = new MemberVO();
    m1.setId("a 사용자");
    data.put("a", m1);
    MemberVO m2 = new MemberVO();
    m2.setId("b 사용자");
    data.put("b", m2);

    model.addAttribute("member", data.get(id));
    return "path/memberInfo";
}
```

@PathVariable

2. 웹 페이지 만들기

- / spring-mvc / webapp / WEB-INF / jsp 에 path 폴더 생성
- 생성된 폴더에 memberInfo.jsp 생성



- 코드 작성

<body> 회원 정보 : \${member.id} </body>

테스트

3. 호출

- index.jsp 파일 코드 추가 후 테스트

```
<a href=" ../path/paramMemberInfo.do?id=a">
```

PathVariable 파라미터

```
</a><br />
```

```
<a href=" ../path/b/pathMemberInfo.do">
```

PathVariable 경로처리

```
</a><br />
```