파라미터 전달

2021-09-12

이승진

**학습목표**

URL을 요청하여, 서버의 컨트롤러 액션 메소드를 호출 할 때,

파라미터(request parameter)를 전달하는 방법을 학습한다.

**목차**

[1. request parameter 전달 2](#_Toc51348004)

[1) request parameter란? 2](#_Toc51348005)

[2) request parameter 종류 2](#_Toc51348006)

[2. request method 3](#_Toc51348007)

[1) request method란? 3](#_Toc51348008)

[2) request method 값의 종류 3](#_Toc51348009)

[3) request method 지정 4](#_Toc51348010)

[4) GET 방식과 request parameter 4](#_Toc51348011)

[3. 추천 책 4](#_Toc51348012)

[4. param 프로젝트 5](#_Toc51348013)

[1) 프로젝트 생성 5](#_Toc51348014)

[2) pom.xml 파일 수정 7](#_Toc51348015)

[3) application.properties 수정 7](#_Toc51348016)

[5. FirstController 8](#_Toc51348017)

[1) FirstController.java 8](#_Toc51348018)

[2) first/test1.jsp 9](#_Toc51348019)

[3) 실행 #1 10](#_Toc51348020)

[4) @RequestParam 11](#_Toc51348021)

[5) 실행 #2 11](#_Toc51348022)

[6) URL encoding 12](#_Toc51348023)

[7) @RequestParam의 속성 12](#_Toc51348024)

[8) 실행 #3 13](#_Toc51348025)

[9) 뷰(view) 파일 공유 13](#_Toc51348026)

[10) @RequestParam 생략 14](#_Toc51348027)

[11) 실행 #4 14](#_Toc51348028)

[6. SecondController 15](#_Toc51348029)

[1) request paramter 형변환 15](#_Toc51348030)

[2) @DateTimeFormat 15](#_Toc51348031)

[3) 액션 메소드의 파라미터가 객체 16](#_Toc51348032)

[4) Person.java 17](#_Toc51348033)

[5) SecondController.java 18](#_Toc51348034)

[6) second/test1.jsp 20](#_Toc51348035)

[7) formatNumber, formatDate 확장 태그 21](#_Toc51348036)

[8) second/test3.jsp 22](#_Toc51348037)

[9) second/test4.jsp 23](#_Toc51348038)

[10) 실행 24](#_Toc51348039)

[7. ThirdController 25](#_Toc51348040)

[1) @RequestMapping의 속성 25](#_Toc51348041)

[2) ThirdController.java 26](#_Toc51348042)

[3) third/test1.jsp 27](#_Toc51348043)

[4) if 확장태그 28](#_Toc51348044)

[5) 실행 28](#_Toc51348045)

[8. 과제 30](#_Toc51348046)

[1) 프로젝트 30](#_Toc51348047)

[2) ThirdController 클래스 수정 30](#_Toc51348048)

[3) URL 30](#_Toc51348049)

# request parameter 전달

## request parameter란?

웹브라우저가 웹서버의 URL을 요청하는 것을 HTTP request이라고 한다.

서버가 Spring Web MVC 앱인 경우, 이 요청에 의해서 컨트롤러의 액션 메소드가 자동으로 호출된다.

이 요청(HTTP Request)에 담겨서 서버에 전달되는 파라미터를, request parameter이라고 한다.

request parameter는 액션 메소드의 파라미터 변수에 전달된다.

## request parameter 종류

다음과 같은 데이터가 request parameter가 되어 서버에 전달된다.

### query string

URL의 query string은 request parameter가 되어 서버에 전달된다.

예:

|  |
| --- |
| http://localhost:8088/first?**id=3&name=tom** |

이 URL에 의해서 다음과 같은 request parameter가 서버에 전달된다.

|  |  |
| --- | --- |
| request parameter name | request parameter value |
| id | 3 |
| name | tom |

### 입력 폼에 입력된 값

|  |
| --- |
| <form method="post">  <input type="text" name="**title**" />  <input type="number" name="**price** />  <button type="submit">저장</button>  </form> |

위 form 태그에 의해, 웹브라우저창에 입력폼이 그려진다.

이 입력폼의 두 칸에 '우유', '1000'을 각각 입력하고 submit 버튼을 클릭하면,

다음과 같은 request parameter가 서버에 전달된다.

|  |  |
| --- | --- |
| request parameter name | request parameter value |
| title | 우유 |
| price | 1000 |

# request method

## request method란?

웹브라우저가 웹서버의 URL을 요청하는 것을 HTTP request이라고 한다.

HTTP request에는 method 속성이 있다.

여기서 method는 객체지향 언어의 method와는 아무 관련이 없다.

HTTP request의 method는, 이 요청이 어떤 종류의 작업을 서버에 요청하는 것인지 구분하기 위한 속성이다.

HTTP request의 method를, **요청 방식** 정도로 해석하자.

## request method 값의 종류

HTTP request method 속성의 값의 종류는 다음과 같다.

|  |  |
| --- | --- |
| 값 | 설명 |
| GET | 서버의 어떤 자료를 요청하는 request의 method는 GET 이어야 한다.  (서버의 데이터 조회 요청) |
| POST | 새 자료를 서버 제출(submit)하는 request의 method는 POST 이어야 한다.  (서버에 데이터 삽입 요청) |
| PUT | 수정된 자료를 서버에 제출(submit)하는 request의 method는 PUT 이어야 한다.  (서버의 데이터 수정 요청) |
| DELETE | 서버의 어떤 자료를 삭제 요청하는 request의 method는 DELETE 이어야 한다.  (서버의 데이터 삭제 요청) |

### 웹 서버 프로그래밍

UI를 포함하여 웹 사이트의 기능의 대부분을 서버에서 구현하는, 웹 서버 프로그래밍에서는

GET, POST 두 요청 방식만 사용하는 것이 보통이다.

서버의 데이터 조회 요청의 경우 GET 방식으로 요청하고,

삽입, 수정, 삭제 요청을 구별하지 않고 POST 방식으로 요청한다.

@Controller 컨트롤러는 웹 서버 프로그래밍 기술에 속한다.

@Controller 컨트롤러를 구현할 때는 뷰(view)를 구현해야 한다.

이 뷰(view)가 UI 이다.

### REST API

프론트엔드 기술로 UI를 개발하고, UI를 제외한 기능만 서버에서 구현하는, 백 엔드 개발에서는

GET, POST, PUT, DELETE 요청 방식을 구분해서 사용하는 보통이다.

REST API를 개발할 때, @RestController 컨트롤러를 구현한다.

@RestController 컨트롤러를 구현할 때는, 뷰(view)를 구현하지 않는다.

### 기타 값

GET, POST, PUT, DELETE 외에도

HEAD, CONNECT, OPTIONS, TRACE, PATH 값들도 있다.

이 값들에 대한 설명은 생략한다.

## request method 지정

### form 태그

예:

|  |
| --- |
| <form method="POST"> |

위의 예와 같이 form 태그에 request method를 지정할 수 있다.

위 form 태그 내부의 submit 버튼이 클릭되면, POST 방식의 HTTP request가 서버에 전달된다.

form 태그에 request method를 지정하지 않으면, 디폴트 값은 GET 이다.

### a 태그

a 태그가 클릭되어 서버에 HTTP request가 전달될 때, request method는 GET 이다.

### 기타

웹브라우저의 주소칸에 URL을 입력하고 엔터키를 눌렀을 때,

eclipse 에서 웹브라우저창이 자동으로 열리고 서버에 HTTP request가 요청되었을 때

등등 대부분의 경우 request method는 GET 이다.

## GET 방식과 request parameter

GET 방식의 요청일 때, request parameter는 URL의 query string으로 서버에 전달된다.

GET 방식의 입력 폼에 입력된 값들은,

그 폼의 submit 버튼이 눌려져 서버에 HTTP request가 전달될 때,

URL의 query string으로 서버에 전달된다.

GET 방식이 아닌 다른 방식의 요청일 경우에, request parameter는

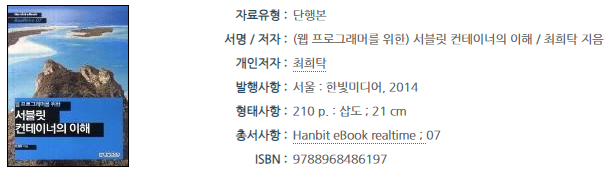
URL의 query string + HTTP request 본문(body)의 데이터 이다.

GET 방식이 아닌 입력 폼에 입력된 값들은,

그 폼의 submit 버튼이 눌려져 서버에 HTTP request가 전달될 때,

HTTP request 본문에 담겨서 서버에 전달된다.

# 추천 책

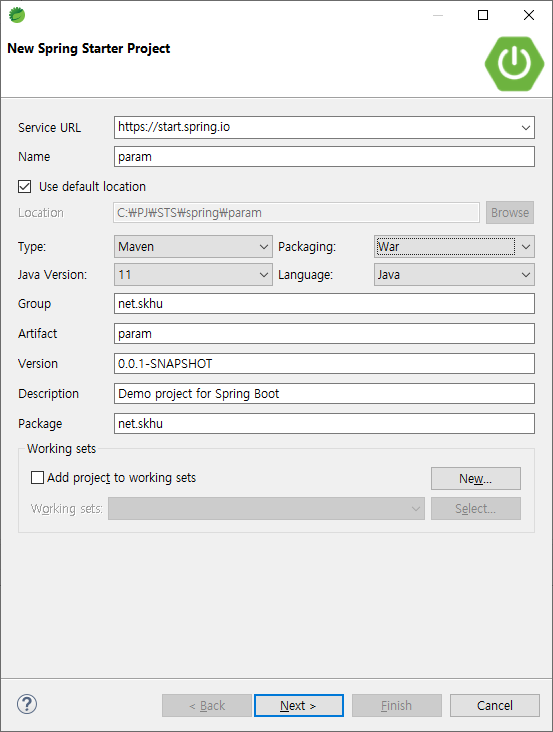


HTTP 프로토콜에 대한 자세한 내용은 위 책을 참고하라.

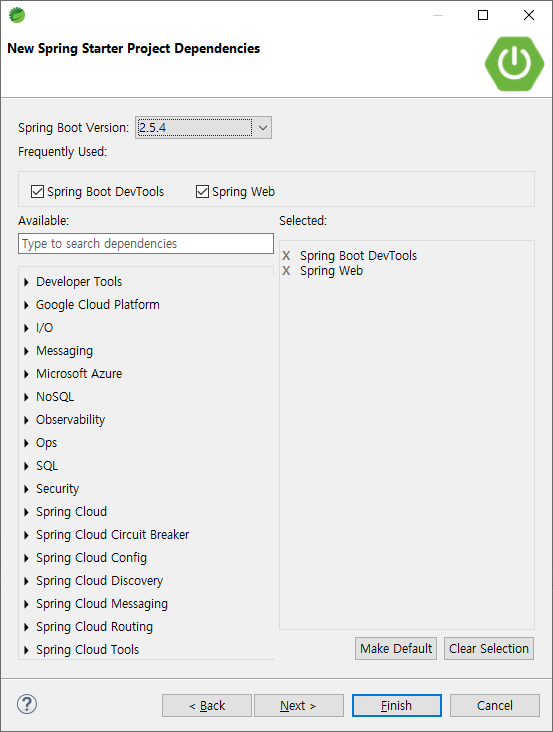
# param 프로젝트

## 프로젝트 생성

메뉴: File - New Spring Starter Propject



|  |  |
| --- | --- |
| Name | param |
| Type | maven |
| Packaging | war |
| Group | net.skhu |
| Artifact | param |
| Package | net.skhu |



Spring Boot DevTools

Spring Web

## pom.xml 파일 수정

|  |
| --- |
| <dependency>  <groupId>jstl</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency>  <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  </dependency> |

pom.xml 파일을 열고 dependencies 태그 내부에 위 내용을 추가하자.

pom.xml 파일은 maven의 설정 파일이다.

## application.properties 수정

src/main/resources/application.properties

|  |
| --- |
| spring.mvc.view.prefix=/WEB-INF/views/  spring.mvc.view.suffix=.jsp  server.port=8088 |

스프링 부트의 설정 파일

# FirstController

## FirstController.java

### src/main/java/net/skhu/controller/FirstController.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | package net.skhu.controller;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  @Controller  @RequestMapping("first")  public class FirstController {  @RequestMapping("test1")  public String test1(Model model, @RequestParam("id") int id,  @RequestParam("name") String name) {  model.addAttribute("id", id);  model.addAttribute("name", name);  return "first/test1";  }  @RequestMapping("test2")  public String test2(Model model,  @RequestParam(value="id", required=false, defaultValue="0") int id,  @RequestParam(value="name", required=false, defaultValue="nobody") String name) {  model.addAttribute("id", id);  model.addAttribute("name", name);  return "first/test1";  }  @RequestMapping("test3")  public String test3(Model model, int id, String name) {  model.addAttribute("id", id);  model.addAttribute("name", name);  return "first/test1";  }  } |

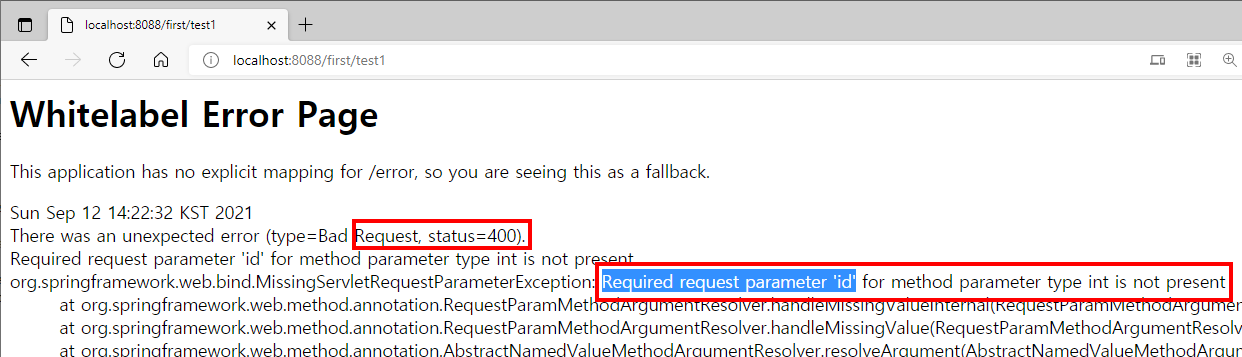
## first/test1.jsp

### src/main/webapp/WEB-INF/views/first/test1.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <style>  table { border-collapse: collapse; }  td { width: 100; padding: 5px; border: 1px solid gray; }  </style>  </head>  <body>  <table>  <tr>  <td>id</td>  <td>${ id }</td>  </tr>  <tr>  <td>name</td>  <td>${ name }</td>  </tr>  </table>  </body>  </html> |

## 실행 #1

### http://localhost:8088/first/test1



### Http Status 코드 400 (Bad Request)

서버에서 에러가 발생했다.

서버에서 HTTP status 코드 400 (Bad Request) 에러가 발생했다.

400 (Bad Request) 에러는, HTTP Request에 문제가 있을 때 발생한다.

HTTP Request에 문제가 있을 경우에, 웹 서버는 웹브라우저에

HTTP status 값이 400 인 HTTP Response를 전송한다.

HTTP status 값이 400 인 HTTP Response를 전달받은 웹브라우저는,

위 화면과 같은 내용을 웹브라우저 창에 출력한다.

### Required request parameter 'id' for method parameter type int is not present

request parameter에 'id' 이름의 데이터가 없어서 발생한 에러이다.

## @RequestParam

|  |
| --- |
| @RequestMapping("test1")  public String test1(Model model, @RequestParam("id") int id,  @RequestParam("name") String name) { |

HTTP Request에 담겨서 서버에 전달된 request parameter는

서버의 액션 메소드의 파라미터 변수에 전달된다.

@RequestParam("파라미터\_이름") 애너테이션이 붙어있는 파라미터 변수에,

그 이름의 request parameter가 전달된다.

@RequestParam("id") int id

"id" 이름의 request parameter 값이 id 파라미터 변수에 자동으로 대입된다.

이 이름의 request parameter 값이 없다면,

서버는 400 (Bad request) HTTP response를 출력한다.

@RequestParam("name") String name

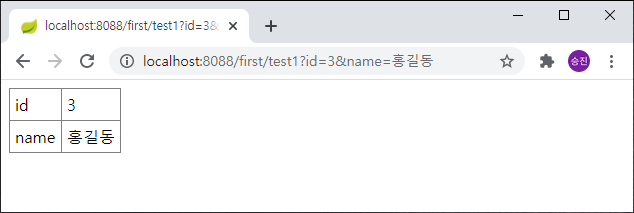
"name" 이름의 request parameter 값이 name 파라미터 변수에 자동으로 대입된다.

이 이름의 request parameter 값이 없다면,

서버는 400 (Bad request) HTTP response를 출력한다.

## 실행 #2

### http://localhost:8088/first/test1?id=3&name=%ED%99%8D%EA%B8%B8%EB%8F%99



## URL encoding

URL에 사용될 수 있는 문자는 다음과 같다.

숫자, 영어 알파벳, 몇몇 기호 문자

한글 문자는 URL에 사용될 수 없다.

URL에 사용될 수 없는 문자는 %ED %8F 형태의 16진수로 인코딩(encoding)되어야 한다.

예를 들어 공백(space) 문자는 URL에 사용될 수 없다.

공백 문자의 ASCII 코드는 32 이다. (16진수로 20)

URL에서 공백 문자는 %20 형태이어야 한다.

이 URL 인코딩/디코딩은 웹브라우저 웹서버가 자동으로 해주기 때문에,

많은 경우 프로그래머는 신경쓰지 않아도 된다. (가끔 신경써서 구현해야 하는 경우도 있다.)

예를 들어, 웹브라우저 주소칸에 다음과 같이 입력하면

|  |
| --- |
| http://localhost:8088/first/test1?id=3&name=홍길동 |

웹브라우저가 이 URL을 아래의 형태로 바꿔서 서버에 요청해 준다.

|  |
| --- |
| http://localhost:8088/first/test1?id=3&name=%ED%99%8D%EA%B8%B8%EB%8F%99 |

서버에서 request parameter가 변수에 대입될 때, 자동으로 디코딩(decoding)된 값이 대입된다.

## @RequestParam의 속성

|  |
| --- |
| @RequestMapping("test2")  public String test2(Model model,  @RequestParam(value="id", required=false, defaultValue="0") int id,  @RequestParam(value="name", required=false, defaultValue="nobody") String name) { |

### @RequestParam(value="id", required=false, defaultValue="0") int id

required=false 이므로, "id" 이름의 request parameter가 HTTP request에 담겨 있지 않아도,

에러가 발생하지 않고, defaultValue 속성으로 지정한 값이, 파라미터 변수에 대입된다.

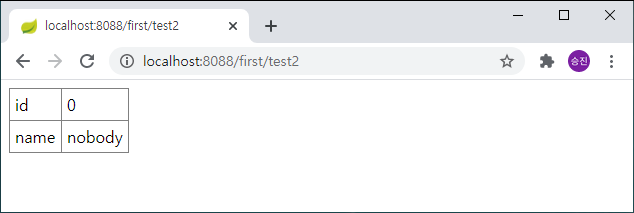
즉 0 값이 id 변수에 대입된다.

defaultValue 속성의 값을 언제나 따옴표로 묶어야 함에 주의하자.

(예: defaultValue=0 컴파일 에러)

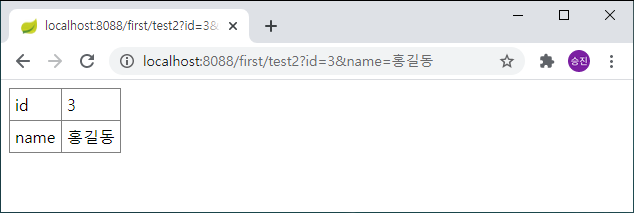
## 실행 #3

### http://localhost:8088/first/test2



파라미터 변수에 defaultValue가 대입되었다.

### http://localhost:8088/first/test2?id=3&name=%ED%99%8D%EA%B8%B8%EB%8F%99



## 뷰(view) 파일 공유

test1, test2, test3 액션 메소드의 리턴 값이 모두 "first/test1" 이다.

출력할 내용(HTML)이 같은 여러 액션 메소드가 뷰 파일 하나를 공유하는 것이 바람직하다.

뷰 파일의 이름이 액션 메소드의 이름과 같을 필요 없고,

request mapping URL과 같을 필요도 없다.

그렇지만, 액션 메소드 이름, request mapping URL, 뷰 파일 이름을 일치시키는 것이

헷갈리지 않아서 좋다.

## @RequestParam 생략

|  |
| --- |
| @RequestMapping("test3")  public String test3(Model model, int id, String name) { |

위 코드는 아래 코드와 유사하다.

|  |
| --- |
| @RequestMapping("test3")  public String test1(Model model, @RequestParam("id") int id,  @RequestParam("name") String name) { |

차이점은 다음과 같다.

위 코드는 id, name 이름의 request parameter가 없어도,

400 Bad request 에러가 발생하지 않고,

액션 메소드 파라미터 변수에 null 값이 대입된다.

옛날 Java에서는 컴파일된 결과 bytecode에서 파라미터 변수와 지역 변수 이름이 제거되었다.

그래서 파라미터 변수 이름과 일치하는 request parameter를 찾는 것이 가능하지 않아서,

찾을 이름을 적어주기 위한 @RequestParam 애너테이션이 꼭 필요했다.

Java 8부터는 컴파일된 결과 bytecode에서 파라미터 변수와 지역 변수 이름이 남아있다.

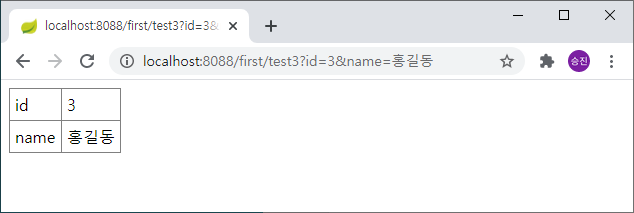
그래서 파라미터 변수 이름과 일치하는 request parameter를 찾는 것이 가능하다.

서버에서는 아직도 Java 7 이하의 환경을 사용하는 경우가 있다.

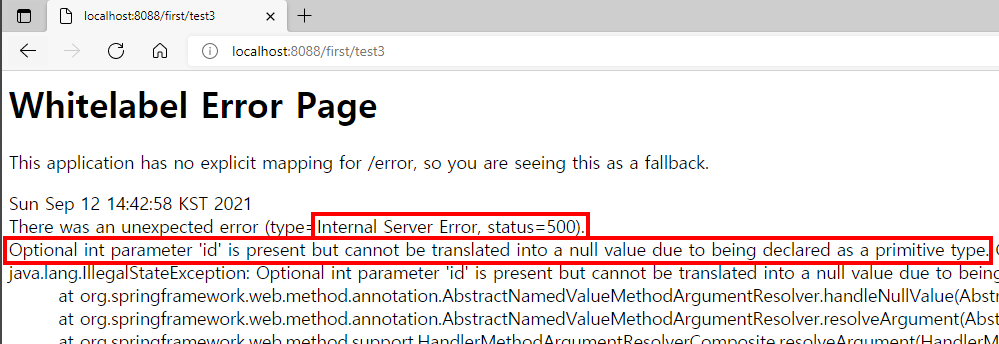
이 경우에 @RequestParam을 생략할 수 없다.

## 실행 #4

### http://localhost:8088/first/test3?id=3&name=%ED%99%8D%EA%B8%B8%EB%8F%99



### http://localhost:8088/first/test3



request parameter에 'id' 이름의 값이 없어서, int id 변수에 null 을 대입해야 하는데,

int 변수에 null 을 대입할 수 없으므로 500 Internal Server Error 발생

# SecondController

## request paramter 형변환

### Sring

request parameter 값은 기본적으로 문자열 텍스트로 전달된다.

액션 메소드의 파라미터 타입이 String 이라면, 형변환이 필요없다.

request paramter 값도 문자열이기 때문이다.

### 기본 자료형

request paramter 값이 액션 메소드의 파라미터 변수에 대입될 때,

파라미터 변수의 타입이 기본자료형(byte, int, long, float, double, boolean, char)이라면,

자동으로 형변환 된다.

예:

|  |
| --- |
| @RequestParam("weight") double weight |

### 기타

액션 메소드의 파라미터 타입이,

String도 아니고, 기본 자료형도 아니라면, 즉 객체라면,

자동 형변환이 되지 않는다.

## @DateTimeFormat

액션 메소드의 파라미터 타입이 Date 인 경우에, request paramter 값이 형변환되어 대입될 수 있으려면

@DateTimeFormat 애너테이션이 필요하다.

예

|  |
| --- |
| @RequestParam("birthday") @DateTimeFormat(pattern="yyyy-MM-dd") Date birthday |

birthday 이름의 request paramter 값이, Date 타입으로 형변환되어, birthday 파라미터 변수에 대입된다.

이때 request paramter 값이 "yyyy-MM-dd" 패턴이 아니라면 형변환 에러가 발생한다.

(HTTP status 400 Bad Request)

## 액션 메소드의 파라미터가 객체

|  |
| --- |
| @RequestMapping("test3")  public String test3(Model model, Person person) { |

test3 액션 메소드의 파라미터가 Person 객체이다.

### request paramter 값 전달

액션 메소드의 파라미터가 객체인 경우에, request parameter 값이 그 객체 내부의 속성에 대입된다.

이때 request paramter의 이름과 일치하는 객체의 속성에 대입된다.

Java 객체의 속성은 멤버 변수가 아니고, getter와 setter 이다. (get 메소드, set 메소드)

객체의 속성값을 읽어야 할 때, get 메소드가 호출되고,

객체의 속성에 값을 대입해야 할 때, set 메소드가 호출된다.

이름이 일치하는 객체의 속성이 없어서 값이 전달될 수 없는 request paramter가 있어도 에러가 발생하지 않는다. 그 reuqest paramter는 무시된다.

이름이 일치하는 request paramter가 없어서 값이 대입되지 않는 객체의 속성이 있어도 에러가 발생하지 않는다.

### model attribute

액션 메소드의 파라미터가 객체인 경우에, 그 객체는 자동으로 모델(model)에 저장되어 뷰(view)에 전달된다.

예를 들어

|  |
| --- |
| @RequestMapping("test3")  public String test3(Model model, Person person) { |

이 Person 객체에 대해서 아래 코드가 자동으로 실행된다.

|  |
| --- |
| model.addAttribute("person", person); |

이 때, 이 데이터의 이름 "person"은, 파라미터 변수 이름과 아무 상관 없고,

이 객체의 클래스 이름에서 첫 문자만 소문자로 변경한 것이다.

## Person.java

### src/main/java/net/skhu/dto/Person.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46 | package net.skhu.dto;  import java.util.Date;  import org.springframework.format.annotation.DateTimeFormat;  public class Person {  String name;  double weight;  Date birthday;  public Person() {  }  public Person(String name, double weight, Date birthday) {  this.name = name;  this.weight = weight;  this.birthday = birthday;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public double getWeight() {  return weight;  }  public void setWeight(double weight) {  this.weight = weight;  }  public Date getBirthday() {  return birthday;  }  @DateTimeFormat(pattern="yyyy-MM-dd")  public void setBirthday(Date birthday) {  this.birthday = birthday;  }  } |

Person 클래스에, name 속성, weight 속성, birthday 속성이 구현되었다.

Java 객체의 속성은 멤버 변수가 아니고, getter와 setter 이다. (get 메소드, set 메소드)

예를 들어 name 속성은 아래 두 메소드로 구현된다.

|  |
| --- |
| public String getName()  public void setName(String name) |

request parameter가 채워질 때, 자동으로 형변환이 될 수 있도록

setBirthday 메소드에 @DateTimeFormat 애너테이션을 붙였다.

이 애너테이션이 없으면 형변환 에러가 발생한다 (HTTP status 400 Bad Request)

## SecondController.java

### src/main/java/net/skhu/controller/SecondController.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54 | package net.skhu.controller;  import java.util.Date;  import org.springframework.format.annotation.DateTimeFormat;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  @Controller  @RequestMapping("second")  public class SecondController {  @RequestMapping("test1")  public String test1(Model model, @RequestParam("name") String name,  @RequestParam("weight") double weight,  @RequestParam("birthday") String birthday) {  model.addAttribute("name", name);  model.addAttribute("weight", weight);  model.addAttribute("birthday", birthday);  return "second/test1";  }  @RequestMapping("test2")  public String test2(Model model,  @RequestParam("name") String name,  @RequestParam("weight") double weight,  @RequestParam("birthday") @DateTimeFormat(pattern="yyyy-MM-dd") Date birthday) {  model.addAttribute("name", name);  model.addAttribute("weight", weight);  model.addAttribute("birthday", birthday);  return "second/test1";  }  @RequestMapping("test3")  public String test3(Model model, Person person) {  model.addAttribute("name", person.getName());  model.addAttribute("weight", person.getWeight());  model.addAttribute("birthday", person.getBirthday());  return "second/test3";  }  @RequestMapping("test4")  public String test4(Model model, Person person) {  model.addAttribute("person", person);  return "second/test4";  }  @RequestMapping("test5")  public String test5(Model model, Person p) {  return "second/test4";  }  } |

test1 액션 메소드의 birthday 파라미터는 String 타입이므로 형변환이 필요없다.

test2 액션 메소드의 birthday 파라미터는 Date 타입이므로 형변환이 필요하다.

test3 액션 메소드에서 "name", "weight", "birthday" 이름의 request paramter 값들이

Person 객체 내부에 채워져서 전달된다.

test4 액션 메소드 본문의 아래 코드는, 자동으로 실행되므로 생략해도 된다.

|  |
| --- |
| model.addAttribute("person", person); |

test5 액션 메소드에서 Person p 파라미터 변수가 model에 저장될 때, 이 데이터의 이름은

파라미터 변수 이름 "p"가 아니고, 클래스 이름에서 첫 문자만 소문자로 변경한 것이다.

즉 "person" 이다.

test4 액션 메소드와 test5 액션 메소드가 하는 일은 같다.

## second/test1.jsp

### src/main/webapp/WEB-INF/views/second/test1.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <style>  table { border-collapse: collapse; }  td { width: 100; padding: 5px; border: 1px solid gray; }  </style>  </head>  <body>  <table>  <tr>  <td>name</td>  <td>${ name }</td>  </tr>  <tr>  <td>weight</td>  <td>${ weight }</td>  </tr>  <tr>  <td>birthday</td>  <td>${ birthday }</td>  </tr>  </table>  </body>  </html> |

## formatNumber, formatDate 확장 태그

JSP에서 숫자나 날짜를 정해진 포맷으로 출력할 때,

formatNumber, formatDate 확장 태그를 사용한다.

### JSTL

이 확장 태그들은 JSTL 라이브러리에 들어있다.

따라서 프로젝트의 pom.xml 파일에 JSTL dependency가 들어있어야 한다.

|  |
| --- |
| <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %> |

위 확장 태그들을 사용하기 위해 필요한 선언이다.

이 선언이 JSP 파일 선두에 있어야 한다.

### formatNumber

|  |
| --- |
| <fmt:formatNumber value="${ weight }" pattern="#.#" /> |

weight 값이 소숫점 한자리까지 출력된다.

그 아래 자리는 반올림된다.

|  |
| --- |
| <fmt:formatNumber value="${ weight \* 1000 }" pattern="#,###" /> |

weight \* 1000 식의 값이 출력된다.

3자리마다 콤마가 출력된다. (예: 1,234,567)

formatNumber 확장 태그의 pattern은, Java의 DecimalFormat 클래스의 그것과 같다.

### formatDate

|  |
| --- |
| <fmt:formatDate pattern="yyyy-MM-dd" value="${ birthday }" /> |

birthday 값을 "yyyy-MM-dd" 형태로 출력한다.

birthday 값은 Date 타입이어야 한다.

formatDate 확장 태그의 pattern은, Java의 SimpleDateFormat 클래스의 그것과 같다.

패턴의 예:

"yyyy-MM-dd HH:mm:ss"

## second/test3.jsp

### src/main/webapp/WEB-INF/views/second/test3.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <style>  table { border-collapse: collapse; }  td { width: 100; padding: 5px; border: 1px solid gray; }  </style>  </head>  <body>  <table>  <tr>  <td>name</td>  <td>${ name }</td>  </tr>  <tr>  <td>weight</td>  <td>  <fmt:formatNumber value="${ weight }" pattern="#.#" />kg  <fmt:formatNumber value="${ weight \* 1000 }" pattern="#,###" />g  </td>  </tr>  <tr>  <td>birthday</td>  <td><fmt:formatDate pattern="yyyy-MM-dd" value="${ birthday }" /></td>  </tr>  </table>  </body>  </html> |

## second/test4.jsp

### src/main/webapp/WEB-INF/views/second/test4.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <style>  table { border-collapse: collapse; }  td { width: 100; padding: 5px; border: 1px solid gray; }  </style>  </head>  <body>  <table>  <tr>  <td>name</td>  <td>${ person.name }</td>  </tr>  <tr>  <td>weight</td>  <td>  <fmt:formatNumber value="${ person.weight }" pattern="#.#" />kg  <fmt:formatNumber value="${ person.weight \* 1000 }" pattern="#,###" />g  </td>  </tr>  <tr>  <td>birthday</td>  <td><fmt:formatDate pattern="yyyy-MM-dd" value="${ person.birthday }" /></td>  </tr>  </table>  </body>  </html> |

${ person.name }

모델의 "person" 이름의 객체가 들어있어야 한다.

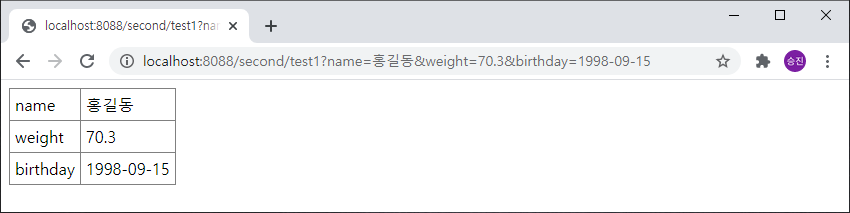
예:

|  |
| --- |
| model.addAttribute("person", person); |

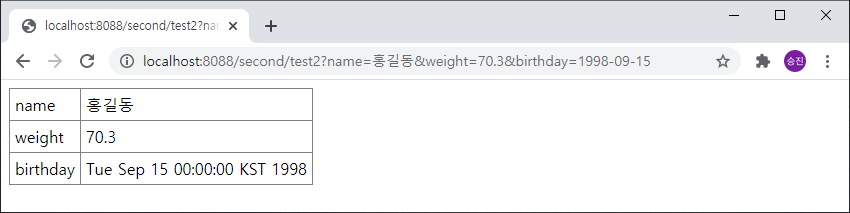
이 객체의 name 속성값을 출력한다. (getName() 메소드의 리턴값)

## 실행

http://localhost:8088/second/test1?name=홍길동&weight=70.3&birthday=1998-09-15



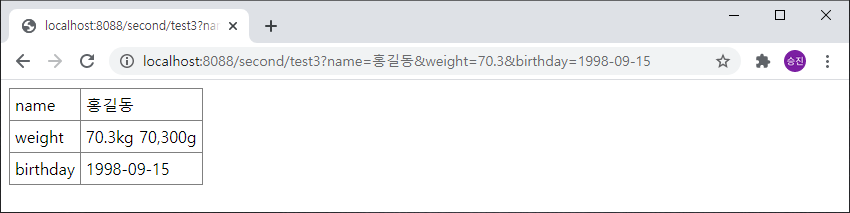
http://localhost:8088/second/test2?name=홍길동&weight=70.3&birthday=1998-09-15



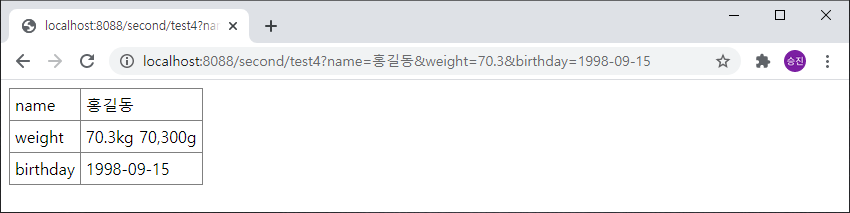
Date 타입인 birthday 값을 아래의 코드를 출력하면, birthday.toString() 메소드의 리턴값이 출력된다.

|  |
| --- |
| ${ birthday } |

http://localhost:8088/second/test3?name=홍길동&weight=70.3&birthday=1998-09-15



http://localhost:8088/second/test4?name=홍길동&weight=70.3&birthday=1998-09-15



# ThirdController

## @RequestMapping의 속성

|  |
| --- |
| @RequestMapping("test1") |

request method 값과 무관하게, "test1" URL이 요청되면, 이 액션 메소드가 호출된다.

|  |
| --- |
| @RequestMapping(value="test1", method=RequestMethod.GET) |

GET 방식의 요청인 경우에만, 이 액션 메소드가 호출된다.

위 코드는 아래 코드와 동일하다.

|  |
| --- |
| @GetMapping("test1") |

GET 방식의 요청인 경우에만, 이 액션 메소드가 호출된다.

|  |
| --- |
| @RequestMapping(value="test1", method=RequestMethod.POST) |

POST 방식의 요청인 경우에만, 이 액션 메소드가 호출된다.

위 코드는 아래 코드와 동일하다.

|  |
| --- |
| @PostMapping("test1") |

POST 방식의 요청인 경우에만, 이 액션 메소드가 호출된다.

## ThirdController.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | package net.skhu.controller;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PostMapping;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  @Controller  @RequestMapping("third")  public class ThirdController {  @RequestMapping(value="test1", method=RequestMethod.GET)  public String test1(Model model) {  model.addAttribute("person", new Person());  return "third/test1";  }  @RequestMapping(value="test1", method=RequestMethod.POST)  public String test1(Model model, Person person) {  // TODO: 저장하는 코드를 구현해야 함.  model.addAttribute("message", "저장되었습니다");  return "third/test1";  }  @GetMapping("test2")  public String test2(Model model) {  model.addAttribute("person", new Person());  return "third/test1";  }  @PostMapping("test2")  public String test2(Model model, Person person) {  // TODO: 저장하는 코드를 구현해야 함.  model.addAttribute("message", "저장되었습니다");  return "third/test1";  }  } |

## third/test1.jsp

### src/main/webapp/WEB-INF/views/third/test1.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <style>  div.container { width: 600px; margin: auto; }  form { box-shadow: 0 0 4px lightgray, 2px 2px 4px gray;  overflow: auto; font-size: 10pt; }  div.title { font-size: 20pt; padding: 10px; background-color: #eee; }  table { margin: 20px; }  td { min-width: 100; padding: 5px; }  td:nth-child(1) { text-align: right; }  input { padding: 4px; }  button { padding: 0.4em 2em; margin: 5px 0 20px 20px; }  div.message { border-radius: 10px; padding: 12px; margin: 10px;  background-color: #ffa; border: 1px solid #cc0; }  </style>  </head>  <body>  <div class="container">  <form method="post">  <div class="title">person 정보</div>  <table>  <tr>  <td>name</td>  <td><input type="text" name="name" value="${ person.name }" /></td>  </tr>  <tr>  <td>weight</td>  <td><input type="number" name="weight" value="${ person.weight }" step="0.1" /></td>  </tr>  <tr>  <td>birthday</td>  <fmt:formatDate pattern="yyyy-MM-dd" value="${ person.birthday }" var="b" />  <td><input type="date" name="birthday" value="${ b }" /></td>  </tr>  </table>  <button type="submit">저장</button>  <c:if test="${ not empty message }">  <div class="message">  ${ message }  </div>  </c:if>  </form>  </div>  </body>  </html> |

## if 확장태그

|  |
| --- |
| <c:if test="${ 조건식 }">  . . .  </c:if> |

조건식이 true 인 경우에만, 이 확장 태그 사이의 내용이 웹브라우저로 출력된다.

|  |
| --- |
| <c:if test="${ not empty message }">  . . .  </c:if> |

모델에 "message" 이름의 데이터가 있을 경우에만, 이 확장 태그 사이의 내용이 웹브라우저로 출력된다.

이 확장 태그를 사용하려면, 아래 선언이 필요하다.

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

## 실행

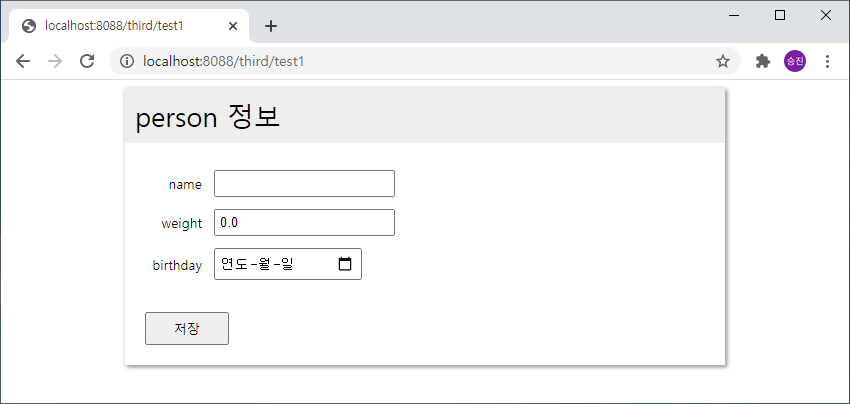
http://localhost:8088/third/test1 URL을 처음 요청하는 것은 GET 방식이다.

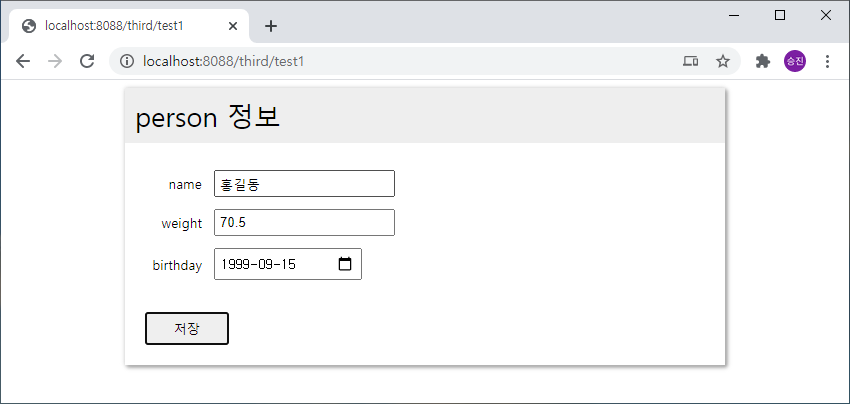
따라서 아래 메소드가 호출된다.

|  |
| --- |
| @RequestMapping(value="test1", method=RequestMethod.GET)  public String test1(Model model) {  model.addAttribute("person", new Person());  return "third/test1";  } |

모델에 저장된 Person 객체의 속성에는 null 이나 0 이 들어있다.

이 Person 객체의 속성값들이, third/test1.jsp 뷰에 의해서 다음과 같이 출력된다.





위와 같이 입력하고 '저장' 버튼을 클릭하면,

http://localhost:8088/third/test1 URL이 서버에 다시 요청된다.

이 요청은 POST 방식이다.

|  |
| --- |
| <form method="post"> |

입력폼에 입력된 데이터는 HTTP request 본문(body)에 담겨서 서버에 전송된다. (request paramter)

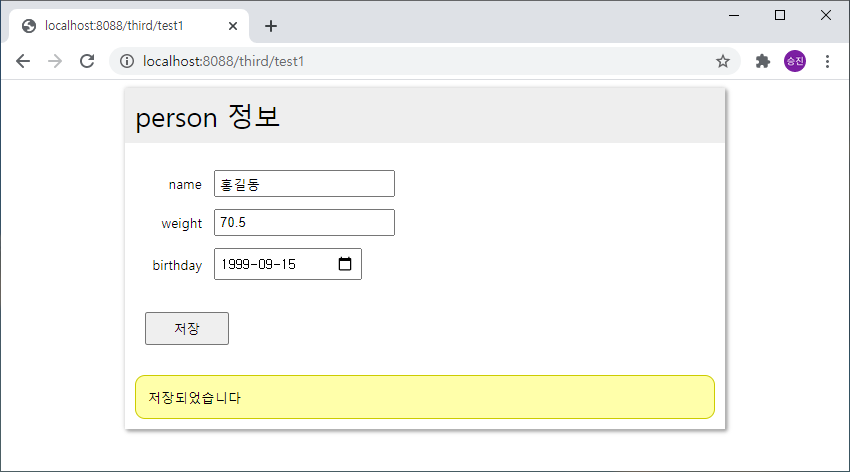
|  |
| --- |
| @RequestMapping(value="test1", method=RequestMethod.POST)  public String test1(Model model, Person person) {  // TODO: 저장하는 코드를 구현해야 함.  model.addAttribute("message", "저장되었습니다");  return "third/test1";  } |

서버에서 위 액션 메소드가 호출된다.

입력폼에 입력되어 서버에 전달된 데이터는, Person 객체의 속성에 자동으로 대입된다.

그리고 이 Person 객체는 자동으로 모델에 저장된다.

이 Person 객체의 속성값들이, third/test1.jsp 뷰에 의해서 다음과 같이 출력된다.



# 과제

지난 시간의 구현 과제를 수정하라.

## 프로젝트

hwA학번

예: hwA201132045

## ThirdController 클래스 수정

### test3 액션 메소드 구현

위의 강의 예제에서 Person 정보 입력 폼과 동일한 구조로

Student 입력 폼을 구현하라.

## URL

test3 액션 메소드를 호출하기 위한 URL은 다음과 같아야 한다.

http://localhost:8088/third/test3