6장. 스프링 MVC의 Controller

동의과학대학교
컴퓨터정보과
김진숙

학습내용

- 스프링 Controller의 특징
- 스프링 Controller의 분기
- 스프링 Controller의 파라미터 수집
- Model 객체

스프링 MVC Controller의 특징

- HttpServletRequest, HttpServletResponse를 거의 사용할 필요 없이 필요한 기능 구현 가능
- **URI 분기**
  - GET 방식, POST 방식 등 전송 방식에 대한 URL 처리를 메소드 어노테이션으로 처리 가능
    - @GetMapping
    - @PostMapping
- **파라미터 자동 수집**
  - Request.getParameter("id")의 기능과 유사
- 다양한 타입의 **리턴 타입 사용** 가능
- 상속/인터페이스 방식 대신에 어노테이션만으로도 필요한 설정 가능(오버라이딩 등이 필요없게 됨)

스프링 MVC는 어노테이션을 중심으로 구성된다.

@Controller 패키지 등록

- **servlet-context.xml**
  - 해당 클래스의 인스턴스를 스프링의 빈으로 등록하고 컨트롤러로 사용
  - <component-scan>과 같이 활용(**servlet-context.xml**)

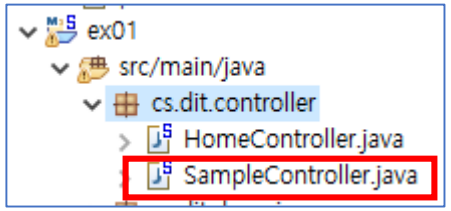
```
<context:component-scan base-package="cs.dit.controller" />
```

관련 어노테이션

어노테이션	설명
@Controller	• 해당 클래스가 Controller임을 나타내기 위한 어노테이션 • 스프링의 빈으로 등록하고 컨트롤러로 사용, <component-scan>과 같이 활용
@RequestMapping	• 들어온 URI 요청을 특정 메서드와 매핑하기 위해 사용(요청 배열로 제시가능) • Class와 Method에 사용 가능 • **클래스 레벨**에서 **공통 조건**을 지정 • **메서드 레벨**에서 이를 세분화 • URI 요청 시 이 둘(클래스 레벨, 메서드 레벨)을 조합하여 최종 조건이 결정
@GetMapping	• 주어진 URI 표현식과 일치하는 HTTP GET 요청을 처리(요청 배열로 제시가능) • HTTP GET 메서드는 특정한 리소스를 가져오도록 요청(**조회, 게시**)
@PostMapping	• 주어진 URI 표현식과 일치하는 HTTP POST 요청을 처리 • POST 요청은 보통 HTML 양식을 통해 서버에 전송(**수정, 등록, 삭제** 등)
@RequestParam	• 파라미터 수집을 명시함 • 파라미터명이 다를 때 사용 • Controller 메소드의 파라미터와 웹요청 파라미터와 맵핑하기 위한 어노테이션
@ModelAttribute	• Controller 메소드의 파라미터나 리턴값을 Model 객체와 바인딩하기 위한 어노테이션 • 이 어노테이션이 있는 파라미터는 파라미터명도 변경하고 화면까지 전달됨

URI 분기

- @RequestMapping("/sample/*")
- 특정한 **URI 분기**에 대한 처리를 해당 컨트롤러나 메서드에서 처리 가능
- 현재 클래스의 모든 메서드들의 **기본적인 URL 경로** 설정



SampleController 클래스

```
package org.zerock.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
@RequestMapping("/sample/*")
@Log4j
public class SampleController {
}
```

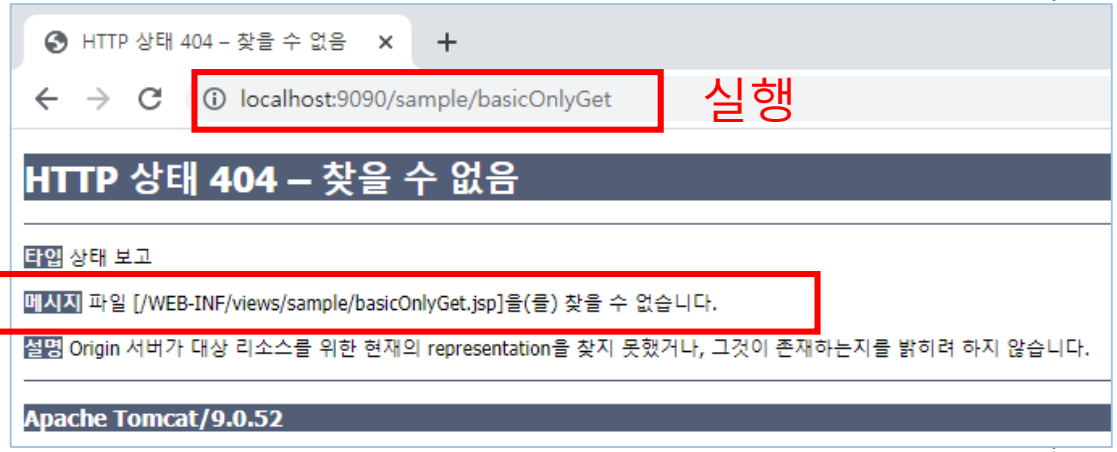
메서드 내에서 viewName을 별도로 설정하지 않으면 @RequestMapping의 path로 설정한 URL이 그대로 viewName으로 설정

@RequestMapping의 변화

- 스프링 4.3전까지 : @RequestMapping(method ='get') 방식 사용
- 스프링 4.3이후는 : @GetMapping, @PostMapping등으로 간단히 표현

SampleController 클래스의 일부

```
@RequestMapping(value = "/basic", method = {RequestMethod.GET, RequestMethod.POST})
public void basicGet() {
    log.info("basic.....................");
}

@GetMapping("/basicOnlyGet")
public void basicGet2() {
    log.info("basic.....................");
}
```




```
INFO : cs.dit.controller.SampleController - basic ----------------------
```

STS 실행 메세지

Controller 메소드의 리턴타입

리턴 타입	설명
String	• jsp를 이용하는 경우에는 jsp 파일의 **경로와 파일로 분기**하기 위해서 사용
void	• 호출하는 **URI**과 동일한 이름의 jsp를 의미(url 지정할 필요 없음)
VO, DTO	• 주로 JSON 타입의 데이터를 만들어서 반환하는 용도로 사용 (추가적인 라이브러리 필요, ajax)
ResponseEntity	• response할 때 Http 헤더 정보와 내용을 가공하는 용도로 사용 (추가적인 라이브러리 필요)
Model, ModelAndView	• Model로 데이터를 반환하거나 화면까지 같이 지정하는 경우에 사용 (최근 많이 사용하지 않음)
HttpHeaders	• 응답에 내용 없이 Http 헤더 메시지만 전달하는 용도로 사용(ResponseEntity랑 같이 사용)

void 타입

- 호출하는 URI와 동일한 이름의 jsp 파일명
- 경로도 클래스 공통경로와 요청을 합해서 만들어짐

```
@RequestMapping("")
public void basic() {
        log.info("basic---------");

}
```

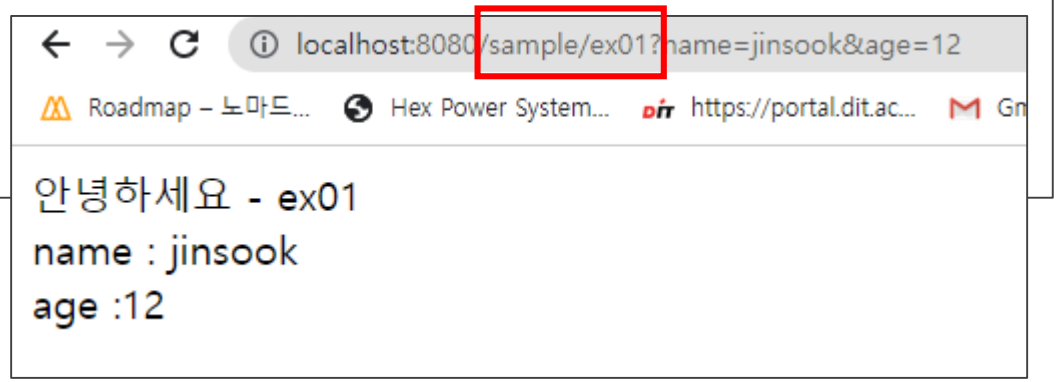
```
@RequestMapping("/ex01")
public void ex01() {
        log.info("ex01---------");

}
```

```
@RequestMapping({"/ex01", "/ex011"})
public void ex02() {
        log.info("ex02---------");

}
```

String 타입

- 상황에 따라 다른 화면을 보여줄 필요가 있을 경우에 유용하게 사용
- String 타입에는 다음과 같은 특별한 키워드를 붙여서 사용할 수 있음
  - redirect: 리다이렉트 방식으로 처리하는 경우
    - 게시판 기능에서 게시글을 입력한 뒤 리스트 보기로 이동할 때 등에 많이 사용됨
  - forward: 포워드 방식으로 처리하는 경우(기본 처리로 암묵적으로 사용됨)

```java
@GetMapping("ex01")
public String ex01(@ModelAttribute("name") String name, @ModelAttribute("age") int age) {
    //기본 데이터타입의 값은 화면까지 전달 되지 않기 때문에 @ModelAttribute를 사용하여 화면까지 전달
    log.info("name :" + name);
    log.info("age :" + age);

    return "/sample/ex01";
}
```



Controller의 파라미터 자동 수집기능

- 스프링 MVC의 컨트롤러는 메서드의 다양한 파라미터를 자동으로 수집, 변환하는 편리한 기능을 제공
  - 클라이언트에게서 전달되는 데이터를 자동으로 수집
  - Request.getParameter(“id”)의 기능과 유사

[실습]

- cs.dit.controller 패키지에 SampleController 작성
- 기본 URL경로를 @RequestMapping("/sample/*")로 설정
- 요청 방법
  - http://localhost:8080/sample/basic

@RequestMapping("")에 세분화된 URI 요청이 없으면 메소드명으로 실행된다.

```java
package cs.dit.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

import cs.dit.domain.SampleDto;
import lombok.extern.log4j.Log4j;

@Controller
@RequestMapping("/sample/*") //기본 URL 경로 설정
@Log4j
public class SampleController {

    //기본 URL 경로에서 실행, 기본은 get방식, 리턴은 기본url로 이동
    @RequestMapping("")
    public void basic() {
        log.info("basic.....................");
    }
```

[실습]

- SampleController 에 작성
- 기본 URL경로를 @RequestMapping("/sample/*")로 설정되어 있음
- @RequestParam()은 **파라미터명 변경**에 사용
- 요청 방법
  - http://localhost:8080/sample/ex01
  - http://localhost:8080/sample/ex02?name=홍길동
  - http://localhost:8080/sample/ex03?id=홍길동

```
//get방식, return type이 String, /sample/ex01.jsp로 이동
@GetMapping("/ex01")
public String ex01() {
    log.info("ex01 ..........................");

    return "/sample/ex01";
}
//get방식, 매개변수가 있음
@GetMapping("/ex02")
public String ex02(String name) {
    log.info("ex02 ..........................");
    log.info("name :" + name);

    return "/sample/ex02";
}
//get방식 가져오는 매개변수이름과 다를 때 @RequestParam() 사용
@GetMapping("/ex03")
public void ex03(@RequestParam("id") String name) {
    log.info("ex03 ..........................");
    log.info("name :" + name);

}
```

객체 데이터를 파라미터로 전달

- Java Beans 규칙에 맞게 작성되어야 함
  - 생성자
  - 올바른 규칙으로 만들어진 Getter/Setter 등

=> 객체는 기본적으로 화면까지 정보가 전달됨

객체 데이터를 파라미터로 전달

SampleDto

```
@Data
public class SampleDto {

 private String name;
 private int age;
}
```

SampleController 일부

```
 @GetMapping("/ex06")
 public  void ex06(SampleDto dto) {
        log.info("ex06 ..........");
        log.info(dto.getName());
        log.info(dto.getAge());
        log.info(dto);
}
```

반환되는 문자열로 Servlet-context.xml 의 InternalResourceViewResolver가 viewPage를 생성

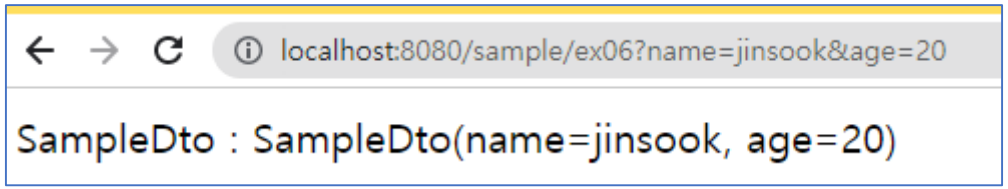
실행 로그

```
INFO : cs.dit.controller.HomeController - Welcome home! The client locale is ko_KR.
INFO : cs.dit.controller.SampleController - ex06 ..........
INFO : cs.dit.controller.SampleController - 홍길동
INFO : cs.dit.controller.SampleController - 20
INFO : cs.dit.controller.SampleController - SampleDto(name=홍길동, age=20)
```

Ex06.jsp 화면

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>ex06</title>
8 </head>
9 <body>
10  SampleDto : ${sampleDto}
11 </body>
12 </html>
```

Model객체에 담지 않아도 화면까지 전달됨
객체명의 첫문자를 소문자로 만들어가 함



@ModelAttribute

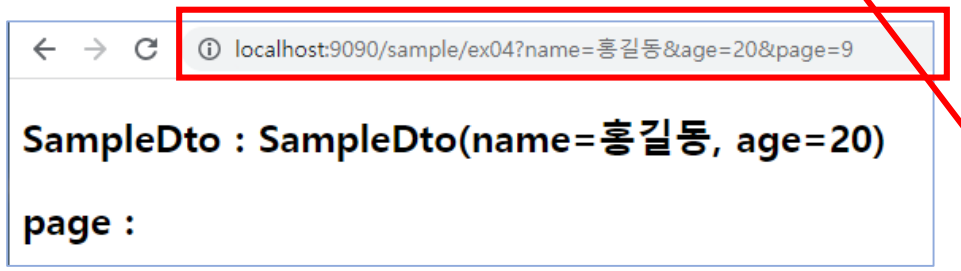
- 컨트롤러에서 메서드의 파라미터는 기본자료형을 제외한 객체형 타입은 다시 화면으로 전달(java beans 규칙에 맞는 것만)
- @ModelAttribute는 명시적으로 이름을 변경(필요 시)하여 화면까지 전달되도록 지정한 것

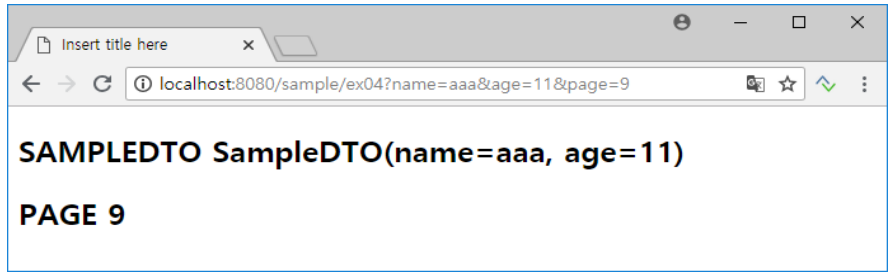
```
@GetMapping("/ex04")
public String ex04(SampleDTO dto, int page) {

  log.info("dto: " + dto);
  log.info("page: " + page);

  return "/sample/ex04";
}
```

SampleController.java

```
<h2>SAMPLEDTO ${sampleDto }</h2>
<h2>PAGE ${page }</h2>
```

ex04.jsp





int page 앞에 **@ModelAttribute("page")** 를 넣어주면 Model 객체에 담아 전달된다

[실습]

- SampleController 에 작성
- 기본 URL경로를 @RequestMapping("/sample/*")로 설정되어 있음
- @ModelAttribute()로 화면까지 데이터를 보낼 수 있음
- "redirect:/";는 response.sendRedirect()와 같음
- 요청 방법
  - http://localhost:8080/sample/ex04?id=홍길동
  - http://localhost:8080/sample/ex05?name=홍길동

- /Web-INF/views/sample/ex04.jsp 파일을 만들고 데이터가 전달되는지 확인할 것

```
//화면까지 데이터가 전달되고 파라미터명을 변경하고자 할 때 @RequestParam() 사용
@GetMapping("/ex04")
public void ex04(@ModelAttribute("id") String name) {
    log.info("ex04 ..........................");
    log.info("name :" + name);

}
@GetMapping("/ex05")
public String ex05(String name) {
    log.info("ex05 ..........................");
    log.info("name :" + name);

    return "redirect:/";
}
```

RedirectAttribute

- 화면에 한번만 전달되는 파라미터를 처리하는 용도
- 내부적으로 **HttpSession객체**에 담아서 한번만 사용되고, 폐기

```
response.sendRedirect("/home?name=aaa&age=10");
```

와 유사

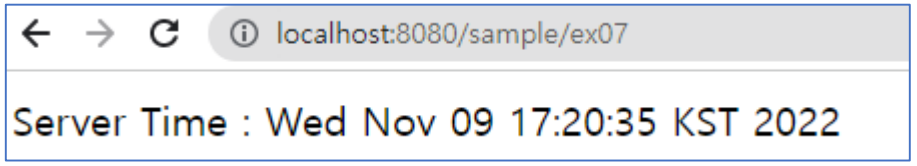
```
@PostMapping(“/modify”)
Public String modify(BoardVO board, RedirectAttributes rttr){
    rttr.addFlashAttribute("name", “aaa");
    rttr.addFlashAttribute("age", 10);

    return "redirect:/";
}
```

Model 객체 : 데이터 전달자

- Model 객체는 컨트롤러에서 생성된 데이터를 담아서 JSP에 전달하는 역할 담당
- 메서드의 파라미터에 Model 타입이 지정된 경우, 스프링 MVC는 자동으로 Model 객체를 만들어 메서드에 주입
- Model에 담는 데이터
  - 파라미터가 아니라 게시글 리스트와 같이 DB 등에서 발생한 데이터를 담기 위한 용기

```java
//Model 객체 사용
@GetMapping("/ex07")
public void ex07(Model model) {
    log.info("ex07 ..........................");

    model.addAttribute("serverTime", new java.util.Date());
}
```

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>ex07</title>
</head>
<body>
 Server Time : ${serverTime }
</body>
</html>
```



모델 2 방식에서 사용하는 request.setAttribute( )와 유사한 역할