

스프링 MVC 웹 요청 처리

- 컨트롤러 만들기 -

컨트롤러 구현

❖ @Controller 어노테이션과 @RequestMapping 어노테이션

- @Controller 어노테이션 : 컨트롤러 클래스를 빈으로 등록
- @RequestMapping 어노테이션 : 클라이언트의 요청을 처리할 메서드에 적용

@Controller

```
public class HelloController {  
    @RequestMapping("/hello") // 요청 URL  
    public String hello() {  
        return "hello";    // 뷰 이름 리턴  
    }  
}
```

컨트롤러 구현

❖ @Controller 어노테이션

- 설정 파일에 컨트롤러 클래스를 빈으로 등록

```
<bean id="helloController"  
      class="com.lecture.springboard.controller.HelloController" />
```

※ 자동 스캔으로 등록하는 경우 <component-scan> 태그의 base-package 지정

- 하위 패키지도 자동 스캔 대상이 됨

```
<context:component-scan base-package="com.lecture.springboard" />
```

- 여러 개의 패키지 등록

```
<context:component-scan  
      base-package="com.lecture.springboard com.lecture.springboard2"  
/>
```

컨트롤러 구현

❖ @RequestMapping

- 컨트롤러 메서드의 HTTP 전송 방식(method) 한정
 - HTTP 전송방식에 따라 다른 메서드 지정 가능

```
@Controller
public class WriteBoardController {
    @RequestMapping(value="/board/write", method=RequestMethod.GET)
    public String form() {
        return "board/board_write";
    }
    @RequestMapping(value="/board/write", method=RequestMethod.POST)
    public String submit(Board board) {

        return "board/board_write_result";
    }
}
```

컨트롤러 구현

❖ @RequestMapping

- 컨트롤러 메서드의 HTTP 전송 방식(method) 한정
 - URL이 동일한 경우 간소화

@Controller

@RequestMapping("/board/write")

```
public class WriteBoardController {  
    @RequestMapping(method=RequestMethod.GET)  
    public String form() {  
        return "board/write";  
    }  
    @RequestMapping(method=RequestMethod.POST)  
    public String submit(Board board) {  
  
        return "board/board_write_result";  
    }  
}
```

컨트롤러 구현

❖ HTML 폼과 커맨드 객체

- 자바빈 객체
 - HTML 폼 항목 이름과 자바빈 클래스의 프로퍼티 이름 일치
- 폼 데이터의 자바빈 객체 전달
 - @RequestMapping 어노테이션 적용 메서드의 파라미터로 자바빈 타입 추가

```
@Controller
@RequestMapping("/board/write")
public class WriteBoardController {
    ...
    @RequestMapping(method=RequestMethod.POST)
    public String submit(Board board) {
        // board.getTitle()      : title 파라미터 값
        // board.getContent(): content 파라미터 값
        // board.getWriter()      : writer 파라미터 값
        ...
        return "board/board_write_result";
    }
    ...
}
```

컨트롤러 구현

❖ HTML 폼과 커맨드 객체

- 뷰에서 커맨드 객체 접근하기
 - 뷰 코드에서 컨트롤러에서 전달받은 커맨드 객체 접근 가능
 - 커맨드 객체는 자동으로 모델에 추가
 - 커맨드 객체의 클래스 이름을 이용해서 커맨드 객체에 접근

- 컨트롤러

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

클래스 첫 글자는 소문자로 변경
(Board → board)

- 뷰

```
<body>
...
제목: ${board.title}
```

컨트롤러 구현

❖ HTML 폼과 커맨드 객체

- 뷰에서 커맨드 객체 접근하기
 - 커맨드 객체의 모델 이름 변경
 - @ModelAttribute 어노테이션으로 지정 가능

- 컨트롤러

```
@RequestMapping(method=RequestMethod.POST)
public String submit(
    @ModelAttribute("command") Board board) {
    ...
}
```

- FORM

```
<body>
...
제목: ${command.title}
```



컨트롤러 구현

❖ @RequestMapping 메서드의 파라미터 타입

파라미터 타입	설명
HttpServletRequest HttpServletResponse HttpSession	서블릿 API
java.util.Locale	현재 요청에 대한 Locale
InputStream, Reader	요청 콘텐츠에 직접 접근할 때 사용
OutputStream, Writer	응답 콘텐츠에 직접 접근할 때 사용
@PathVariable 파라미터	URI 템플릿 변수에 접근할 때 사용
@RequestParam 파라미터	HTTP 요청 파라미터를 매핑
@RequestHeader 파라미터	HTTP 요청 헤더 파라미터를 매핑
@CookieHeader 파라미터	HTTP 쿠키 매핑
@RequestBody 파라미터	HTTP 요청의 몸체 내용에 접근할 때 사용
Map, Model, ModelMap	뷰에 전달할 모델 데이터를 설정할 때 사용
커맨드 객체	HTTP 요청 파라미터를 저장한 객체
Errors, BindingResult	HTTP 요청 파라미터를 커맨드 객체에 저장한 결과
SessionStatus	폼 처리를 완료했음을 처리하기 위해 사용, @SessionAttribute 어노테이션을 명시한 session 속성을 제거하도록 이벤트 발생

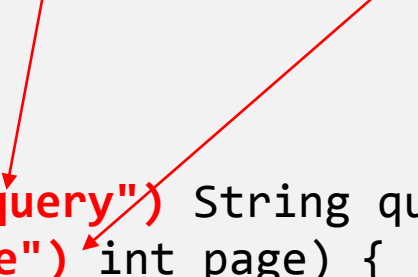
컨트롤러 구현

❖ 컨트롤러 메서드의 파라미터 타입

- @RequestParam 어노테이션을 이용한 파라미터 매핑
 - HTTP 요청 파라미터를 메서드의 파라미터로 전달받을 때 사용

http://localhost/SpringBoard/board/search?**query**=spring&**page**=2

```
@Controller
public class SearchController {
    @RequestMapping("/board/search")
    public String search(    @RequestParam("query") String query,
                          @RequestParam("page") int page) {
        :
        return "board/search_result";
    }
}
```



- 메서드의 파라미터 타입이 String이 아닌 경우
 - 실제 타입에 따라 알맞게 타입 변환 수행
 - 타입이 맞지 않는 경우 잘못된 요청을 의미하는 400 응답 코드를 웹 브라우저에 전송

컨트롤러 구현

❖ 컨트롤러 메서드의 파라미터 타입

- @RequestParam 어노테이션을 이용한 파라미터 매핑
 - 필수가 아닌 파라미터인 경우
 - required 속성을 false로 지정(기본값은 true)

```
@RequestMapping("/board/search")
public ModelAndView search (
    @RequestParam(value="query", required=false) String query,
    @RequestParam(value="page", required=false) int page) {
    ...
}
```

- 필수가 아닌 요청 파라미터의 값이 존재하지 않을 경우 null 값 할당
- 기본 데이터 타입은 null을 가질 수 없으므로 예외 발생
- 기본값 할당 설정 defaultValue 속성이용

```
@RequestMapping("/board/search")
public ModelAndView search (
    @RequestParam(value="query", required=false) String query,
    @RequestParam(value="page", defaultValue="1") int page) {
    ...
}
```

컨트롤러 구현

❖ 컨트롤러 메서드의 파라미터 타입

- @CookieValue 어노테이션을 이용한 쿠키 매핑
 - 쿠키 값을 파라미터로 전달받음
 - auth 쿠키의 값을 authValue 파라미터를 통해 전달받기

```
@RequestMapping("/cookie/view.do")
public String view( @CookieValue("auth") String authValue) {
    . . .
    return "cookie/view";
}
```

- 해당 쿠키가 존재하지 않으면 500에러 발생
- 필수가 아닌 경우 required 속성을 false로 지정(기본값은 true)

```
@RequestMapping("/cookie/view.do")
public String view(
    @CookieValue(value="auth", required="false") String
    authValue) {
    . . .
}
```

컨트롤러 구현

❖ 컨트롤러 메서드의 파라미터 타입

- @CookieValue 어노테이션을 이용한 쿠키 매핑
 - 쿠키 값을 파라미터로 전달받음
 - defaultValue 속성으로 기본값 지정

```
@RequestMapping("/cookie/view.do")
public String view(
    @CookieValue(value="auth", defaultValue="0") String authValue)
{
    . . .
}
```

컨트롤러 구현

❖ 컨트롤러 메서드의 파라미터 타입

- @RequestHeader 어노테이션을 이용한 헤더 매핑
 - HTTP 요청 헤더의 값을 메서드의 파라미터로 전달

```
@Controller
public class HeaderController {
    @RequestMapping("/header/check.do")
    public String check(
        @RequestHeader("Accept-Language") String languageHeader) {
        System.out.println(languageHeader);

        return "header/pass";
    }
}
```

- 해당 헤더가 존재하지 않으면 500 응답 에러 코드 전송
- required, defaultValue 속성 이용 가능

컨트롤러 구현

❖ 컨트롤러 메서드의 파라미터 타입

- 서블릿 API 직접 사용
 - 사용가능 파라미터
 - javax.servlet.http.HttpServletRequest/javax.servlet.ServletRequest
 - javax.servlet.http.HttpServletResponse/javax.servlet.ServletResponse
 - javax.servlet.http.HttpSession

컨트롤러 구현

❖ 컨트롤러 메서드의 파라미터 타입

- 서블릿 API 직접 사용
 - 사용하면 편리한 경우
 - HttpSession의 생성을 직접 제어해야 하는 경우
 - 컨트롤러에서 쿠키를 생성해야 하는 경우
 - 서블릿 API 사용을 선호하는 경우

```
@RequestMapping("/someUrl")
public ModelAndView process(HttpServletRequest request, ...) {
    if(someCondition) {
        HttpSession session = request.getSession();
    }
    ...
}
```


컨트롤러 구현

❖ 컨트롤러 메서드의 파라미터 타입

- 서블릿 API 직접 사용
 - 사용하면 편리한 경우
 - HttpSession 타입의 파라미터를 가질 경우 세션은 항상 생성

```
@RequestMapping("/someUrl")  
public ModelAndView process(HttpSession session, ...) {  
    ...  
}
```

컨트롤러 구현

❖ Url을 바로 뷰로 매핑하기

○ servlet-context.xml

```
<beans:bean
  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <beans:property name="mappings">
    <beans:props>
      <beans:prop key="/home2">urlController</beans:prop>
      <beans:prop key="/test/test">urlController</beans:prop>
    </beans:props>
  </beans:property>
</beans:bean>

<beans:bean id="urlController"
  class="org.springframework.web.servlet.mvc.UrlFilenameViewController"/>
```

- /home2 → home2
- /test/test → test/test