**프레임워크 프로그래밍**

**6주차 Report**

**그리기이(가) 표시된 사진

자동 생성된 설명**

**한국산업기술대학교**

**201715404컴퓨터공학과**

**권한길**

|  |
| --- |
| **프론트 컨트롤러 패턴 실행 프로세스를 스프링 MVC의 주요 구성 요소를 통해 설명하시오.** |
| 1. 클라이언트(Client)가 서버에 어떤 요청(Request)을 한다면 스프링에서 제공하는 **DispatcherServlet** 이라는 클래스(일종의 front controller)가 요청을 가로챈다.  (web.xml에 살펴보면 모든 url ( / )에 서블릿 매핑을하여 모든 요청을 DispatcherServlet이 가로채게 해둠(변경 가능))  2. 요청을 가로챈 DispatcherServlet은 **HandlerMapping**(URL 분석등..)에게 어떤 컨트롤러에게 요청을 위임하면 좋을지 물어본다.  (servlet-context.xml에서 @Controller로 등록한 것들을 스캔해서 찾아준다.)  3. 요청에 매핑된 컨트롤러가 있다면 @RequestMapping을 통하여 요청을 처리할 메서드에 도달한다.  4. 컨트롤러에서는 해당 요청을 처리할 Service를 주입(DI)받아 비즈니스로직을 Service에게 위임한다.  5. Service에서는 요청에 필요한 작업 대부분(코딩)을 담당하며 데이터베이스에 접근이 필요하면 DAO를 주입받아 DB처리는 DAO에게 위임한다.  6. DAO는 mybatis(또는 hibernate등) 설정을 이용해서 SQL 쿼리를 날려 DB의 정보를 받아 서비스에게 다시 돌려준다.  (이 때 보통 VO(dto)를 컨트롤러에서 부터 내려받아 쿼리의 결과를 VO에 담는다. (mybatis의 resultType)  7. 모든 로직을 끝낸 서비스가 결과를 컨트롤러에게 넘긴다.  8. 결과를 받은 컨트롤러는 Model객체에 결과물 어떤 view(jsp)파일을 보여줄 것인지등의 정보를 담아 DispatcherServlet에게 보낸다.  9. DispatcherServlet은 ViewResolver에게 받은 뷰의 대한 정보를 넘긴다.  10. ViewResolver는 해당 JSP를 찾아서(응답할 View를 찾음) DispatcherServlet에게 알려준다.  (servlet-context.xml에서 suffix, prefix를 통해 /WEB-INF/views/index.jsp 이렇게 만들어주는 것도 ViewResolver)  11. DispatcherServlet은 응답할 View에게 Render를 지시하고 View는 응답 로직을 처리한다.  12. 결과적으로 DispatcherServlet이 클라이언트에게 렌더링된 View를 응답한다. |

|  |
| --- |
| **@ModelAttribute, @PathVariable, @RequestParam 차이점을 설명하시오.** |
| @PathVariable  @RequestMapping의 URL에 {}로 들어가는 패스 변수를 받는다.  요청 파라미터를 URL의 쿼리 스트링으로 보내는 대신 URL패스로 풀어서 쓰는 방식을 쓰는 경우 매우 유용하다.  타입이 일치하지 않는 값이 들어오면 예외가 발생하며 별다른 예외처리를 해주지 않는다면  클라이언트에 HTTP 400 - Bad Request응답 코드가 전달 됨  @ModelAttibute  이름 그대로 모델로 사용되는 오브젝트이다. 여러개의 @RequestParam을 선언하는 대신에 여러개의 파라미터를 하나의 오브젝트에 담는 것이다. 코드를 훨씬 간결하게 작성할 수 있고 좀더 객체 지향적인 코드 구현이 가능하다. @ModelAttribute가 해주는 한가지 기능이 더 있는데, 그것은 컨트롤러가 리턴하는 모델에 파라미터로 전달한 오브젝트를 자동으로 추가해주는 것이다.  @RequestBody 이 애노테이션이 붙은 파라미터에는 HTTP요청의 본문body부분이 그대로 전달된다. 일반적인 GET/POST의 요청 파라미터라면 @RequestBody를 사용할 일이 없을 것이다. 반면에 XML이나 JSON기반의 메시지를 사용하는 요청의 경우에는 이 방법이 매우 유용하다. |

|  |
| --- |
| **@RequestMapping, @RestController, @RequestBody, @ResponseBody 의미를 설명하시오.** |
| **@RequestMapping**  @RequestMapping 어노테이션은 URL을 컨트롤러의 메서드와 매핑할 때 사용하는 스프링 프레임워크의 어노테이션이에요.  클래스나 메서드 선언부에 @RequestMapping과 함께 URL을 명시하여 사용하죠.  URL외에도 HTTP 요청 메서드나 헤더값에 따라 매핑되도록 -0=옵션을 제공하는데요. 메서드 레벨에서 정의한 @RequestMapping은 타입 레벨에서 정의된 @RequestMapping의 옵션을 상속 받아요.  참고로, 메서드 내에서 viewName을 별도로 설정하지 않으면 @RequestMapping의 path로 설정한 URL이 그대로 viewName으로 설정됩니다..  간단하게 정리하면, 클라이언트는 URL로 요청을 전송하고, 요청 URL을 어떤 메서드가 처리할지 여부를 결정하는 것이 바로 "@RequestMapping" 라고 할수 있어요. **@RestController**  @RestController는 Spring MVC Controlle에 @ResponseBody가 추가된 것입니다. 당연하게도 RestController의 주용도는 Json/Xml 형태로 객체 데이터를 반환하는 것입니다. 개인적으로는 VueJS + Spring boot 프로젝트를 진행할 때 Spring boot를 API 서버로 활용할 때 그리고 Android 앱 개발을 하면서 데이터를 반환할 때 사용하였습니다.  **@RequestBody**  이 어노테이션이 붙은 파라미터에는 HTTP 요청의 본문 body 부분이 그대로 전달된다. **RequestMappingHandlerAdapter**에는 **HttpMessageConverter**타입의 메시지 변환기(message converter)가 여러 개 등록되어 있다. @RequestBody가 붙은 파라미터가 있으면 HTTP 요청의 미디어 타입과 파라미터의 타입을 먼저 확인한다. (dispatcher-servlet.xml 에서 확인) 메시지 변환기 중에서 해당 미디어 타입과 파라미터 타입을 처리할 수 있다면, HTTP 요청의 본문 부분을 통째로 변환해서 지정된 메소드 파라미터로 전달해준다. 쉽게 말하자면 @RequestBody 어노테이션을 이용하면 HTTP 요청 Body를 자바 객체로 전달받을 수 있다.  **@ResponseBody**  @ResponseBody는 @RequestBody와 비슷한 방식으로 동작한다. @ResponseBody가 메서드 레벨에서 부여되면 메서드가 리턴하는 오브젝트는 뷰를 통해 결과를 만들어내는 모델로 사용하는 대신, 메시지 컨버터를 통해 바로 HTTP 응답의 메시지 본문으로 변환된다.  간단히 이야기 하자면, 요청한 형태에 맞춰서 메시지 변환기를 통해 결과값을 반환한다. @ResponseBody는 @RequestBody가 선택한 형식으로 결과값을 변환하여 반환한다고 보면 된다. 또한 @ResponseBody을 이용하면 자바 객체를 HTTP 응답 body로 전송할 수 있다. |

|  |
| --- |
| **Mvc-simple.zip을 참고하여 실습하고 생성한 프로젝트 결과 출력 페이지를 PrintScreen하여 첨부하시오.(MemberController, MemberRestController)** |
| **< 학생 목록 추가 소스>**  스크린샷, 컴퓨터, 노트북이(가) 표시된 사진  자동 생성된 설명  **<학생 목록 추가 결과>**  스크린샷이(가) 표시된 사진  자동 생성된 설명  **<rest API 사용하여 Jason 형태로 출력>**  스크린샷이(가) 표시된 사진  자동 생성된 설명  **<Terminal에서 curl, Get방식으로 정보 출력 >**  **스크린샷이(가) 표시된 사진  자동 생성된 설명**  **<Advanced Rest Client 사용하여 POST방식으로 정보 등록>스크린샷이(가) 표시된 사진  자동 생성된 설명** |