**Spring 내용정리**

□ 기본 구조

Client ↔ Controller ↔ Service ↔ Dao ↔ DB

(Dto) (Dto) (Dto) (Domain, Entity)

↘ ↔ View

□ MVC 역할

∙ Controller : Model과 View 사이의 상호 작용을 관리

- 클라이언트의 요청을 받아, 처리하는 로직을 작성, 해당되는 모델 컴포넌트를 호출

- 클라이언트가 보낸 데이터가 있으면, 모델 호출 시 전달하기 쉽게 데이터를 적절히 가공

- 모델이 업무 수행을 완료하면, 그 결과를 가지고 화면을 생성하도록 뷰에 전달

∙ Model : 애플리케이션의 정보 (데이터, Business Logic 포함)

- 데이터 저장소와 연동하여 사용자가 입력한 데이터나 사용자에게 출력할 데이터를 다룸

- 여러 개의 데이터 변경(추가, 변경, 삭제)을 하나의 작업으로 묶는 트랜잭션을 다룸

- ex) DAO, Service 등의 클래스

∙ View : 사용자에게 제공할 화면

- 클라이언트에게 보여지는 영역. Controller의 처리 결과 화면에 대한 정보를 보유한 객체

□ 역할

∙ Controller

- Model과 View 사이의 상호 작용을 관리. 클라이언트의 요청을 받아 어떻게 처리할지 결정.

Service 등 해당되는 모델 컴포넌트를 호출

- @Controller / @Autowired

- @RequestMapping(value=" ", method= ) : Controller에 들어온 요청을 처리하는 기준점.

주소창에 url + value를 입력하면 method가 수행 / @Override

∙ Service

- 비즈니스 로직을 수행 DB에 접근하는 DAO를 이용해 결과값을 받아옴

- 사용자의 요청에 대해 어떤 처리를 할지 결정. Controller가 받은 요청에 대해 알맞은

정보를 가공해서 Controller에게 다시 데이터를 넘기는 것을 의미

- 여러 번의 DB접근이나, 동시 및 병렬식으로 접근해서 가져와야 함. 여러 Dao를 묶은

트랜잭션의 단위

- 주의할 점은 실제 비즈니스 로직은 Service에서 수행하지 않음. API에서 발생하는 트랜잭션,

Domain의 순서 보장의 역할을 갖고 있으며, 실제 로직은 Domain 패키지에서 수행됨

- @Service / @Autowired / @Resource / @Transactional(rollbackFor = Exception.class)

/ @Override

∙ DAO

- DB에 접속하여 비즈니스 로직 실행에 필요한 쿼리를 호출. 주로 단일 데이터 로직에 사용

- DB로직을 객체 하나에 메서드로 구현하고, 이를 호출하여 사용하도록 만든 것

- DAO는 1개의 Connection으로 다수의 요청을 모두 수행하기 때문에 오버헤드를 줄여,

안정된 운영을 보장

∙ DTO

- 각 계층이 데이터를 주고받을 때 사용하는 객체. VO(Value Object)와 유사

- Client가 요청할 양식과 요청을 처리하는 과정에서 기준이 되는 틀

∙ DispatcherServlet

- 클라이언트의 요청을 처리하고, 그 결과를 Client에 전송하는 기술

- FrontController를 담당하며, 모든 HTTP 요청을 받아들여 다른 객체들 사이의 흐름을 제어

- Servlet의 요청에 관련된 객체를 정의하는 곳이 Servlet-context.xml이 됨. 즉 Controller,

Annotation, ViewResolver 등을 설정

∙ servlet-context.xml

- 적합한 Controller Annotation을 인식하도록 <annotation-driven />을 사용. HandlerMapping

역할을 함

- HTTP GET 요청을 통해 Spring에서 정적인 리소스 인 CSS, HTML 등 파일들을 처리할 수

있도록 등록

- Controller가 반환한 View name을 기반으로 적합한 View(JSP)를 찾을 수 있도록 경로를

지정. ViewResolver 역할을 함

- component-scan은 XML에 Bean을 일일이 지정하지 않고 @Component를 통해 자동으로

Bean을 등록시킴. base-package 경로를 기반으로 탐색, Annotation을 식별하여 Bean 생성

∙ web.xml

- Servlet은 Client의 요청을 처리하고 그 결과를 Client에게 전송하는 기술

- Servlet 특징 : MVC 패턴에서 Controller로 이용 / Client의 요청에 대해 동적으로 작동

/ HTML을 사용하여 요청에 응답 / Java Thread를 이용하여 동작

- DispatcherServlet은 Front Controller를 담당하고, 모든 HTTP 요청을 받아들여 다른 객체들

사이의 흐름을 제어. Servlet의 요청에 관련된 객체를 정의하는 곳인 servlet-context.xml에서

Controller나 Annotation, ViewResolver 등을 설정

- 별칭을 통해 DispatcherServlet을 Mapping 해줌. url-pattern를 '/' 설정하였기에 Root경로로

들어온 모든 요청을 처리할 수 있게 됨

□ 동작

∙ 순서

1. Client로부터 요청이 들어오면 DispatcherServlet이 호출됨

2. DispatcherServlet은 받은 요청을 HandlerMapping에게 던져주고, 요청받은 URL을 분석하여

HandlerMapping 적합한 Controller를 선택하여 반환

3. DispatcherServlet는 다음으로 HandlerAdapter를 호출하고, HandlerApdater는 해당하는

Controller 중 요청한 URL에 맞는 적합한 Method를 찾아줌

4. Controller는 Business Logic을 처리하고, 해당하는 결과를 View에 전달할 객체를 Model에

저장

5. Controller는 View name을 DispatcherServlet에게 리턴

6. DispatcherServlet은 ViewResolver를 호출하여 Controller가 리턴한 View name을 기반으로

적합한 View를 찾아줌

7. DispatcherServlet은 View 객체에 처리결과를 넘겨 최종 결과를 보여주도록 요청

8. View 객체는 해당하는 View를 호출하며, View는 Model 객체에서 화면 표시에 필요한

객체를 가져와 화면 표시를 처리하고 Client에게 넘겨줌

(이러한 설정은 web.xml에 기술되어 있음)

□ 테스트케이스

∙ ADD

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 테스트 케이스ID | | | COM-EDU-MINICALCURATOR-ADD | | 테스트 케이스명 | 두 수를 합함 | | |
| 작성자 | | | 김용원 | | 작성일자 | 2023-10-06 | | |
| 수행자 | | | 김용원 | | 수행일자 | 2023-10-06 | | |
|  | | | | | | | | |
| 선행 조건 | | 테스트케이스가 수행되기 전에 필요한 것이 있는가? | | | | | | |
| NO | Action | | | 예상응답 | | | 결과 | 비고 |
| 1 | byte a = -128  byte b = 127  byte addTest1 = (byte)(a + b) | | | -1 | | | true |  |
| 2 | byte a = -128  byte b = 127  byte addTest2 = (byte)(b + a) | | | -1 | | | true |  |

∙ SUB

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 테스트 케이스ID | | | COM-EDU-MINICALCURATOR-SUB | | 테스트 케이스명 | 두 수를 뺌 | | |
| 작성자 | | | 김용원 | | 작성일자 | 2023-10-06 | | |
| 수행자 | | | 김용원 | | 수행일자 | 2023-10-06 | | |
|  | | | | | | | | |
| 선행 조건 | | - | | | | | | |
| NO | Action | | | 예상응답 | | | 결과 | 비고 |
| 1 | byte a = -128  byte b = 127  int subTest1 = (int)(a - b) | | | -255 | | | true |  |
| 2 | byte a = -128  byte b = 127  int subTest2 = (int)(b - a) | | | 255 | | | true |  |

∙ MUL

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 테스트 케이스ID | | | COM-EDU-MINICALCURATOR-MUL | | 테스트 케이스명 | 두 수를 곱함 | | |
| 작성자 | | | 김용원 | | 작성일자 | 2023-10-06 | | |
| 수행자 | | | 김용원 | | 수행일자 | 2023-10-06 | | |
|  | | | | | | | | |
| 선행 조건 | |  | | | | | | |
| NO | Action | | | 예상응답 | | | 결과 | 비고 |
| 1 | byte a = -128  byte b = 127  int mulTest1 = (int)(a \* b) | | | -16256 | | | true |  |
| 2 | byte a = -128  byte b = 127  int mulTest2 = (int)(b \* a) | | | -16256 | | | true |  |