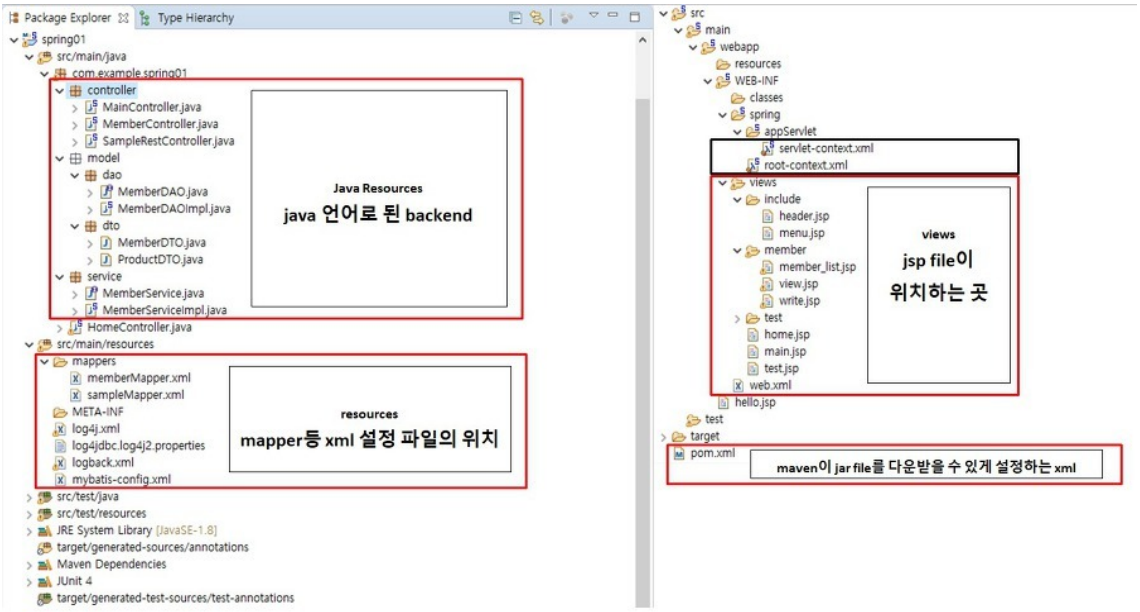


DAO의 정의

Data Access Object의 약자로 간단히 Database의 data에 접근을 위한 객체입니다.
Database에 접근을 하기위한 로직과 비즈니스 로직을 분리하기 위해서 사용을 합니다.
웹 서버와 DB를 연결할 때 커넥션 객체가 필요한데 이 커넥션 객체를 하나씩 생성하고 삭제하면 너무 자원의
소모가 심해서 DB에 접속하는 객체를 전용으로 하나만 만들고 모든 페이지에서 그 객체를 호출할수 있게
해주는 객체가 바로 DAO 객체입니다.

DTO의 정의

Data Transfer Object의 약자이고, 계층간 데이터 교환을 위한 자바빈즈를 말한다.
여기서 말하는 계층은 Controller, View, Business Layer, Persistent Layer를 말한다.
그리고 각 계층간 데이터 교환을 위한 객체를 DTO 또는 VO 라고 부른다.
일반적인 DTO는 로직을 갖고 있지 않는 순수한 데이터 객체이며 속성과 그 속성에
접근 하기 위한 getter, setter 메소드만 가진 클래스를 말합니다.

Service의 정의

Service의 역할은 DAO가 DB에서 받아온 데이터를 전달받아 가공하는 것이다.

Controller의 정의

시스템으로 들어오는 요청과 응답을 담당하는 파일

이렇게 3단계로 package를 설정하는 이유는 기능별로 모듈화 하기 위함이 가장 크다.
모듈화를 하면 계층을 객체 지향적으로 할 수 있고, 유지보수 할 때도 편리하게 사용할 수 있으며 팀 프로젝트를 할 때에도 효율적으로
역할 분담해서 사용 할 수 있다.

모듈화란??

프로그래밍 언어로 프로그램을 제작시 생산성과 최적화, 관리에 용이하게 모듈(기능) 단위로 분할하는 것

프로젝트 실행 과정

1. 스프링 프로젝트를 run(서버 구동) 시키면 브라우저에서 클라이언트가 url을 전송한다.

2. Dispatcher Servlet이 그 url을 가로챈다.

3. web.xml에서 스프링의 환경설정 파일(root-context.xml , servlet-context.xml) 을 모두 읽어들여서 (컨트롤러) bean 생성을 해둔다.

4. 들어온 url로 mapping (@RequestMapping 어노테이션을 사용한 클래스) 되어 있는 controller가 있는지 확인 (Handler Mapping)한다.

5. Dispatcher Servlet가 Controller를 불러 요청한 작업을 수행
   controller는 String을 리턴하여 경로를 알려줄 수도 있다.

6. 작업을 수행한 후, ModelAndView를 리턴한다.

7. View Resolver를 통해 controller에서 들어온 view의 이름을 찾는다.

8. 찾은 view를 요청사항 (model)과 함께 송신한다.

결론 : Dispatcher Servlet가 모든것을 제어한다. 이것이 제어의 역전 (Inversion of Control)
jsp 파일이 아닌 프로젝트 자체를 Run on Server시켜야 처음에 Project가 생성되면서 Dispatcher Servlet가 일을 할 수 있다.

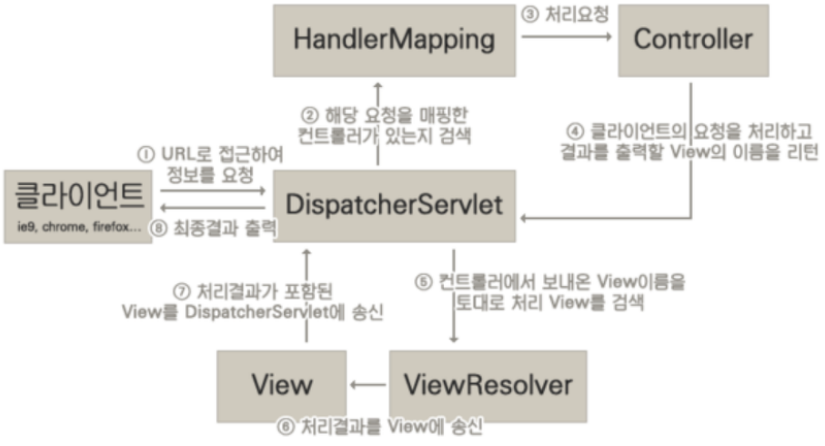
쓰레드	핸들러 (handler)
컴퓨터 프로그램 수행 시 프로세스 내부에 존재하는 수행경로, 즉 일련의 실행코드. 프로세스는 단순히 껍데기이고, 실제 작업은 스레드가 담당한다.	다른 객체가 보낸 메시지 수신, 처리하는 객체 서브 스레드가 보낸 메시지를 받아 UI변경을 한다.

Dispatcher-Servlet 이란?

클라이언트로부터 어떠한 요청이 들어오면 Tomcat (톰캣) 과 같은 서블릿 컨테이너가 요청을 받는데 이때
제일 앞에서 서버로 들어오는 모든 요청을 처리하는 *프론트 컨트롤러를 Spring에서 정의하였고, 이를
Dispatcher - Servlet이라고 합니다.
그래서 공통처리 작업을 Dispatcher 서블릿이 처리한 후, 적절한 세부 컨트롤러로 작업을 위임해준다.
물론 Dispatcher-Servlet가 처리하는 url 패턴을 지정해주어야 하는데 일반적으로 /*.do 같음..
/로 시작하며 .do로 끝나는 url 패턴에 대해서 처리하라고 지정해줍니다.

Dispatcher-Servlet의 장점

Spring MVC는 Dispatcher Servlet가 등장함에 따라 web.xml의 역할을 상당히 축소시켜주었습니다.
기존에는 모든 서블릿에 대해 URL 매핑을 활용하기 위해서 web.xml에 모두 등록해주어야했지만, dispatcher-servlet가
해당 어플리케이션으로 들어오는 모든 요청을 핸들링해주면서 작업을 상당히 편하게 할 수 있게 되었습니다.



Dispatcher-Servlet의 단점

모든 요청을 처리하다보니 이미지나 HTML 파일을 불러오는 요청마다 전부 Controller로 넘겨버립니다.
게다가 JSP 파일 안의 JavaScript나 StyleCSS 파일들에 대한 요청들 까지도 모두 디스패치 서블릿이 가로채는 까닭에 자원을 불러오지 못하는 상황도 발생하곤 했습니다.
이에 대한 해결책은 두가지가 있는데 첫번째는 클라이언트의 요청을 2가지로 분리하여 구분하는 것입니다.

1. /apps의 URL로 접근하면 Dispatcher Servlet가 담당한다.

2. /resources의 URL로 접근하면 Dispatcher Servlet이 컨트롤할 수 없으므로 담당하지 않는다.