MVC 프레임워크 1단계 구현 실습

—

실습 환경

https://github.com/slipp/jwp-basic 저장소를 자신의 계정으로 fork한다.

- Fork한 저장소를 https://youtu.be/xid_GG8kL_w 동영상 참고해 로컬 개발 환경을 구축한다.
- jwp-basic 저장소의 **step1-user-completed-no-database** 브랜치로 변경한다.
  - 브랜치를 변경하는 방법은 https://youtu.be/VeTjDYI7UVs 동영상을 참고한다.
- src/test/java 디렉토리의 next.WebServerLauncher를 실행한 후 브라우저에서 http://localhost:8080으로 접속해 질문/답변 게시판 서비스 화면이 나타나는지 확인한다.

요구사항

요구사항은 MVC 패턴을 지원하는 프레임워크를 구현하는 것이다.
MVC 패턴을 지원하는 기본적인 구조는 HTTP 웹 서버 리팩토링 단계에서 다양한 분기문을 제거할 때 적용한 방법을 그대로 사용하면 된다.
단, 이와 같은 구조로 변경하려면 모든 요청을 RequestHandler가 받아서 요청 URL에 따라 분기처리했듯이 서블릿도 모든 요청을 하나의 서블릿이 받은 후 요청 URL에 따라 분기처리하는 방식으로 구현하면 된다.

사전 정보 공유

모든 클라이언트 요청을 받는 서블릿을 DispatcherServlet 으로 생성한 후 요청 URL에 따라 해당 컨트롤러에 작업을 위임하도록 구현할 수 있다. @WebServlet으로 URL을 매핑할 때**urlPatterns = "/"** 와 같이 설정하면 모든 요청 URL이 DispatcherServlet으로 연결된다.

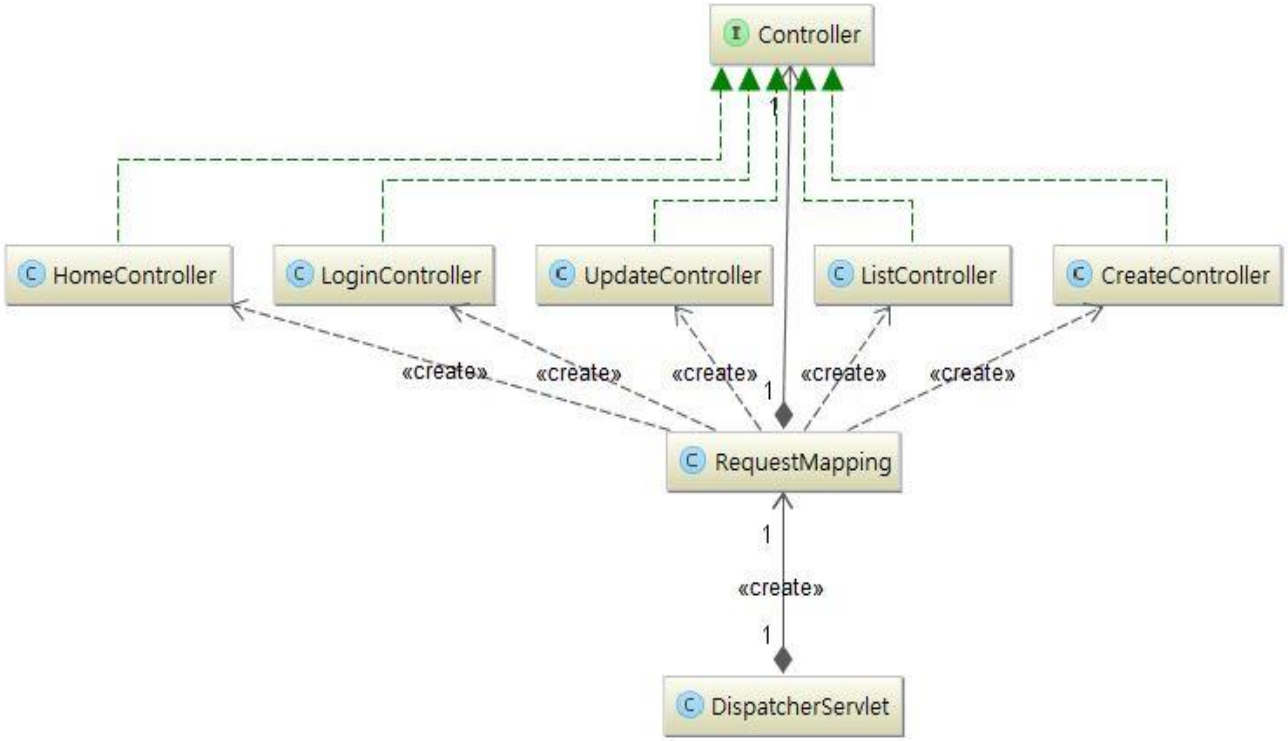
```
@WebServlet(name = "dispatcher", urlPatterns = "/", loadOnStartup = 1)
public class DispatcherServlet extends HttpServlet {
}
```

단, CSS, 자바스크립트, 이미지와 같은 정적인 자원은 굳이 컨트롤러가 필요 없다. 그런데 위와 같이 매핑할 경우 컨트롤러가 필요없는 CSS, 자바스크립트, 이미지에 대한 요청까지 DispatcherServlet으로 매핑이 되어 버리는 상황이 발생한다. 이 같은 문제점을 해결하기 위해 core.web.filter.ResourceFilter를 추가해 해결했다. 따라서CSS, 자바스크립트, 이미지 요청에 대한 처리는 고려하지 않아도 된다.

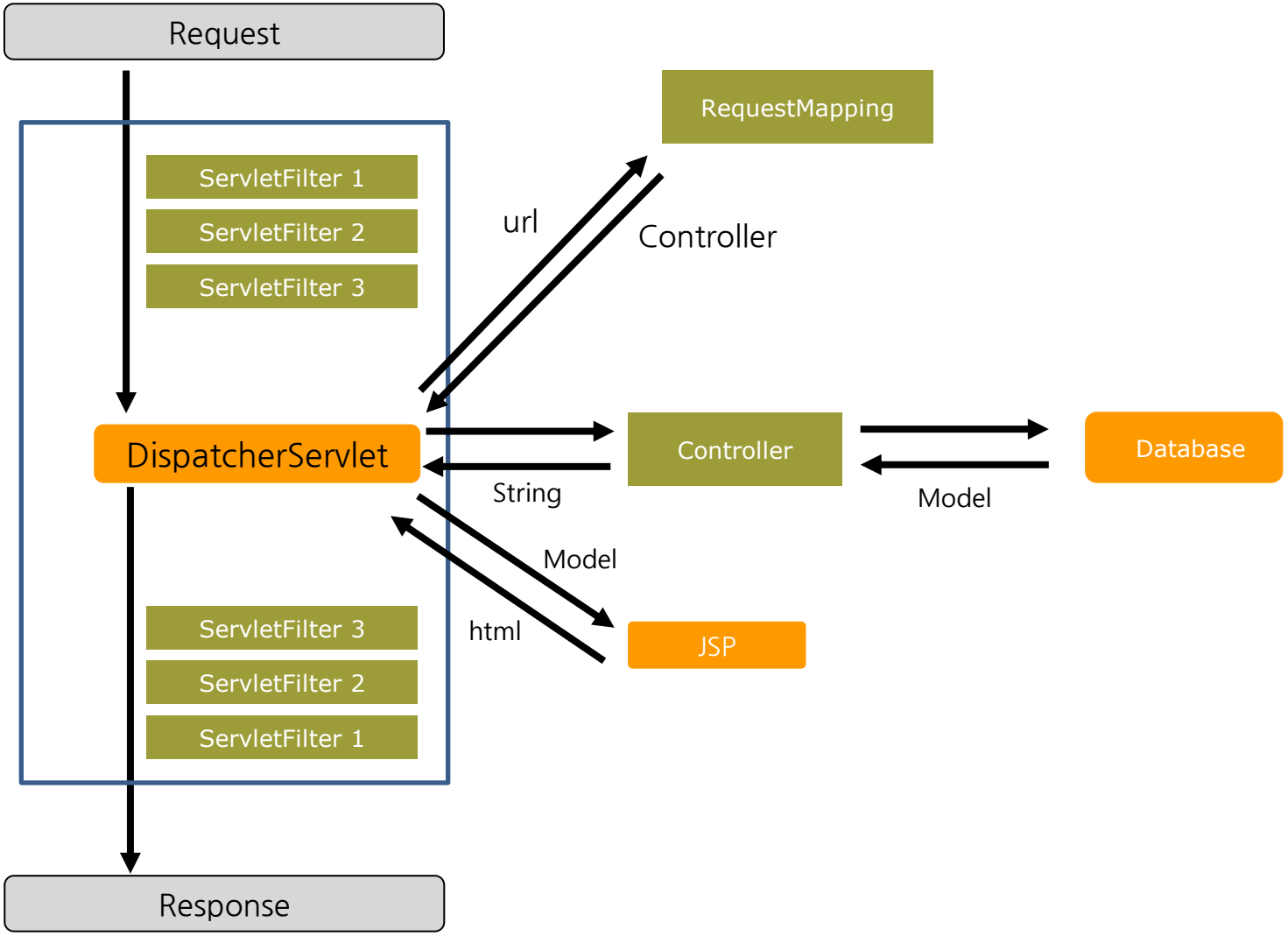
1단계 힌트

—

MVC 프레임워크 구현 완료 후 클래스 다이어그램



MVC 프레임워크 구현 완료 후의 흐름도



요구사항 분리 및 2단계 힌트

—

실습

모든 요청을 서블릿 하나(예를 들어 DispatcherServlet)가 받을 수 있도록 URL 매핑한다.

힌트

@WebServlet(name = "dispatcher", urlPatterns = “/”, loadOnStartup = 1)

loadOnStartup 속성이 무슨 역할을 하는지 학습해 본다.

서블릿은 한 개 이상의 요청 URL로 매핑하는 것이 가능하다.

실습

Controller 인터페이스를 추가한다.

힌트

```
public interface Controller {
    String execute(HttpServletRequest request, HttpServletResponse response) throws Exception;
}
```

execute() 메소드의 반환 값이 String이라는 것을 눈여겨 보자.

실습

서블릿으로 구현되어 있는 회원관리 기능을 앞 단계에서 추가한 Controller 인터페이스를 구현한다. execute() 메소드의 반환 값은 리다이렉트 방식으로 이동할 경우 “redirect:”로 시작하고 포워드 방식으로 이동할 경우 JSP 경로를 반환한다.

힌트

```
public class ListUserController implements Controller {
    public String execute(HttpServletRequest req, HttpServletResponse resp) throws
Exception {
        if (!UserSessionUtils.isLogined(req.getSession())) {
            return "redirect:/users/loginForm";
        }

        req.setAttribute("users", DataBase.findAll());
        return "/user/list.jsp";
    }
}
```

실습

RequestMapping 클래스를 추가해 요청 URL과 컨트롤러 매핑을 설정한다.

힌트

요청 URL과 컨트롤러를 매핑할 때 Map<String, Controller>에 설정한다.

실습

컨트롤러를 추가하다보니 회원가입 화면(/user/form.jsp), 로그인 화면(/user/login.jsp)과 같이 특별한 로직을 구현할 필요가 없는 경우에도 매번 컨트롤러를 생성하는 것은 불합리하다는 생각이 든다. 이와 같이 특별한 로직 없이 뷰(JSP)에 대한 이동만을 담당하는 ForwardController를 추가한다.

forward로 이동할 JSP 경로 정보는 생성자로 전달할 수 있도록 구현한다.

실습

DispatcherServlet에서 요청 URL에 해당하는 Controller를 찾아 execute() 메소드를 호출해 실질적인 작업을 위임한다.

실습

Controller의 execute() 메소드 반환 값 String받아 서블릿에서 JSP로 이동할 때의 중복을 제거한다.

힌트

반환 값이 "redirect:"로 시작할 경우 sendRedirect()로 이동하고, "redirect:"이 아닌 경우 RequestDispatcher의 forward 방식으로 이동한다. 예를 들어 "redirect:/user/list"라면 "/user/list" URL로 리다이렉트하도록 구현한다.