MVC 프레임워크 3단계 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 **step8-self-check-completed** 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 1

- 지금까지 나만의 MVC 프레임워크를 구현해 잘 활용해 왔다. 그런데 새로운 컨트롤러가 추가될 때마다 매번 RequestMapping 클래스에 요청 URL과 컨트롤러를 추가하는 작업이 귀찮다. 귀찮지만 이 정도는 그래도 참을 수 있다. 하지만 유지보수 차원에서 봤을 때 컨트롤러의 수가 계속해서 증가하고 있으며, 각 컨트롤러의 execute() 메소드를 보니 10라인이 넘어가는 경우도 거의 없다. 새로운 기능이 추가될 때마다 매번 컨트롤러를 추가하는 것이 아니라 메소드를 추가하는 방식이면 좋겠다.
- 또 한 가지 아쉬운 점은 요청 URL을 매핑할 때 HTTP 메소드(GET, POST, PUT, DELETE 등)도 매핑에 활용할 수 있으면 좋겠다. HTTP 메소드에 대한 지원이 가능하다면 URL은 같지만 다른 메소드로 매핑하는 것도 가능할 것이다.

```
@Controller
public class MyController {
    private static final Logger logger = LoggerFactory.getLogger(MyController.class);

    @RequestMapping("/users")
    public ModelAndView list(HttpServletRequest request, HttpServletResponse response) {
        logger.debug("users findUserId");
        return new ModelAndView(new JspView("/users/list.jsp"));
    }

    @RequestMapping(value="/users/show", method=RequestMethod.GET)
    public ModelAndView show(HttpServletRequest request, HttpServletResponse response) {
        logger.debug("users findUserId");
        return new ModelAndView(new JspView("/users/show.jsp"));
    }

    @RequestMapping(value="/users", method=RequestMethod.POST)
    public ModelAndView create(HttpServletRequest request, HttpServletResponse response) {
        logger.debug("users create");
        return new ModelAndView(new JspView("redirect:/users"));
    }
}
```

새로운 MVC 프레임워크 테스트

앞의 기능을 지원하는 애노테이션 기반의 새로운 MVC 프레임워크를 구현해야 한다. 효과적인 실습을 위해 새로운 MVC 프레임워크의 뼈대가 되는 코드(src/main/java 폴더의 core.nmvc 패키지)와 테스트 코드(src/test/java 폴더의 core.nmvc.AnnotationHandlerMappingTest)를 제공하고 있다. AnnotationHandlerMappingTest 클래스의 3개의 테스트가 성공하면 새로운 MVC 프레임워크 구현을 완료한 것으로 생각하면 된다.

요구사항 2

- 새로운 MVC 프레임워크도 추가했으니 이전에 구현되어 있던 컨트롤러를 애노테이션 기반으로 변경하면 된다. 그런데 새로운 MVC 프레임워크를 적용하기 위해 한 번에 모든 컨트롤러를 변경하려면 일정 기간 동안 새로운 기능을 추가하거나 변경하는 작업을 중단해야 한다. 이 같은 문제점을 보완하기 위해 점진적으로 리팩토링이 가능한 구조로 개발해야 한다.

실습을 기반 코드 및 라이브러리

- 앞의 예제에서 컨트롤러를 구현할 때 사용한 @Controller와 @RequestMapping 애노테이션은 다음과 같이 구현해step8-self-check-completed 브랜치의 core.annotation 패키지에서 제공하고 있다.
- 이 실습은 클래스패스로 설정되어 있는 클래스 중에 @Controller 애노테이션이 설정되어 있는 클래스를 찾기 위한 목적으로 reflections(https://github.com/ronmamo/reflections) 라이브러리를 활용할 수 있다. 이 라이브러리를 활용해 @Controller 애노테이션이 설정되어 있는 클래스를 찾은 후 @RequestMapping 설정에 따라 요청 URL과 메소드를 연결하도록 구현할 수 있다.

자바 리플렉션 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 **step8-self-check-completed** 브랜치로 변경한다.
    - 브랜치를 변경하는 방법은 https://youtu.be/VeTjDYl7UVs 동영상을 참고한다.
- **src/test/java의 core.ref 패키지에서 제공한다.**

실습

자바 리플렉션을 활용해Question 클래스의 모든 필드, 생성자, 메소드 정보를 출력한다. core.ref.ReflectionTest의 showClass() 메서드를 구현한다.

실습

자바 리플렉션을 활용해 core.ref.Junit3Test에서 test로 시작하는 모든 메서드를 실행해야 한다. core.ref.Junit3TestRunner 클래스의 run() 메소드를 실행했을 때 Junit3Test에서 메소드 이름이 test로 시작하는 모든 메소드를 실행하면 된다. Junit3Test 코드는 다음과 같다.

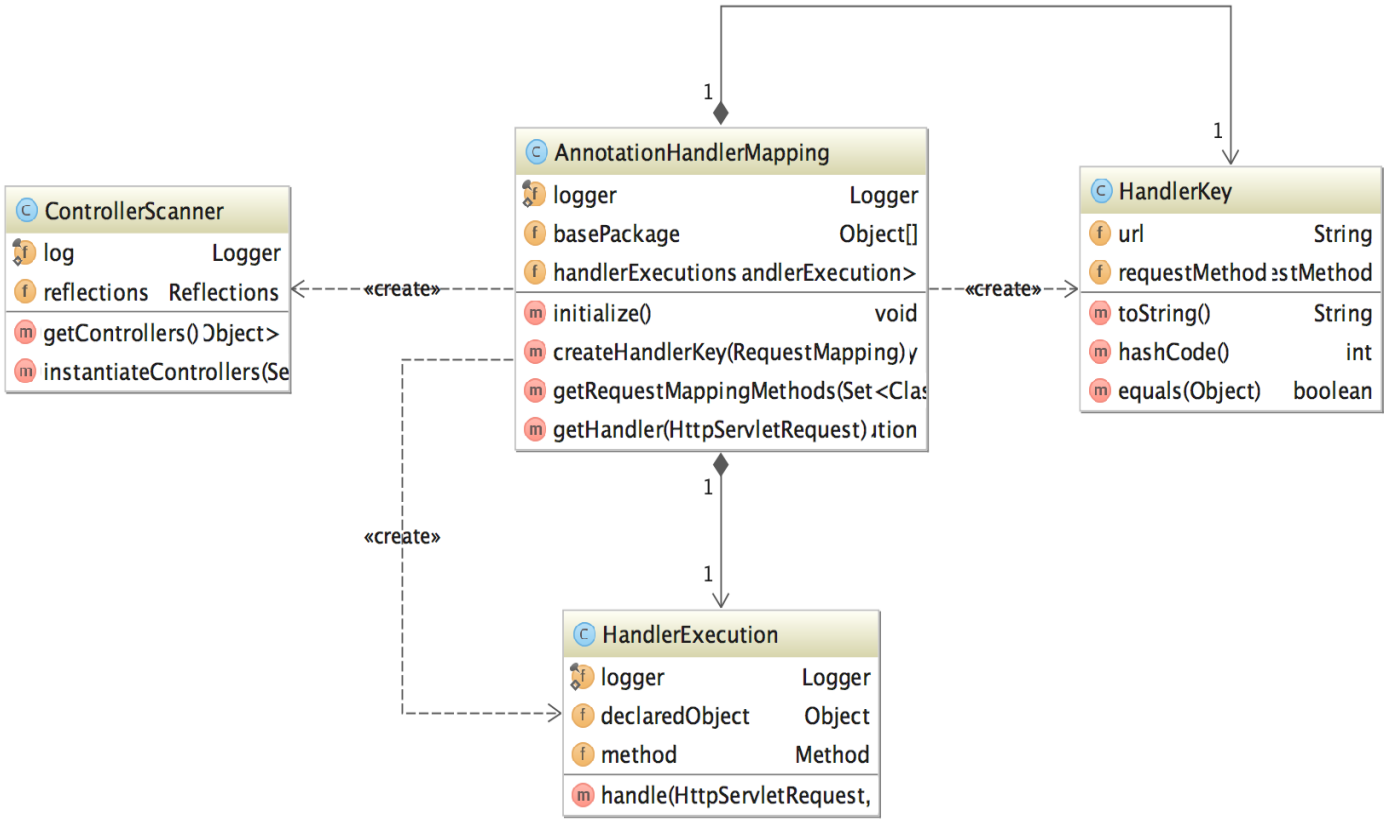
실습

자바 리플렉션을 활용해 core.ref.Junit4Test에서 @MyTest 애노테이션이 설정되어 있는 모든 메소드를 실행해야 한다. 즉, core.ref.Junit4TestRunner 클래스의 run() 메소드를 실행했을 때 Junit4Test에서 @MyTest 애노테이션이 설정되어 있는 모든 메소드를 실행하면 된다.

1단계 힌트

—

새로운 MVC 프레임워크 구현 완료 후 클래스 다이어그램



2개의 MVC 프레임워크를 통합하기 위한 기반 코드

```
@WebServlet(name = "dispatcher", urlPatterns = { "", "/" }, loadOnStartup = 1)
public class DispatcherServlet extends HttpServlet {
    private LegacyHandlerMapping lhm;
    private AnnotationHandlerMapping ahm;

    @Override
    public void init() throws ServletException {
        lhm = new LegacyHandlerMapping();
        lhm.initMapping();
        ahm = new AnnotationHandlerMapping("next.controller");
        ahm.initialize();
    }

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        Controller controller = lhm.findController(req.getRequestURI());
        if (controller != null) {
            render(req, resp, controller.execute(req, resp));
        } else {
            HandlerExecution he = ahm.getHandler(req);
            render(req, resp, he.handle(req, resp));
        }
    }

    […]
}
```

새로운 MVC 구현 힌트

—

실습

reflections 라이브러리를 활용해 @Controller 애노테이션이 설정되어 있는 모든 클래스를 찾고, 각 클래스에 대한 인스턴스 생성을 담당하는 ControllerScanner 클래스를 추가한다.

힌트

- @Controller 애노테이션이 설정되어 있는 모든 클래스를 찾는다.

```
Reflections reflections = new Reflections("my.project");
Set<Class<?>> annotated = reflections.getTypesAnnotatedWith(Controller.class);
```

- 앞 단계에서 찾은 클래스에 대한 인스턴스를 생성해Map<Class<?>, Object>에 추가한다.

```
Class clazz = … ;
clazz.newInstance();
```

실습

애노테이션 기반 매핑을 담당할 AnnotationHandlerMapping 클래스를 추가한 후 초기화한다.

힌트

- ControllerScanner를 통해 찾은 @Controller 클래스의 메소드 중 RequestMapping 애노테이션이 설정되어 있는 모든 메소드를 찾는다. reflections 라이브러리를 활용한다.

```
ReflectionUtils.getAllMethods(clazz, ReflectionUtils.withAnnotation(RequestMapping.class));
```

- 앞 단계에서 찾은java.lang.reflect.Method 정보를 바탕으로Map<HandlerKey, HandlerExecution>에 각 요청 URL과 URL과 연결되는 메소드 정보를 값으로 추가한다.
- Map의 HandlerKey는 RequestMapping 애노테이션이 가지고 있는 URL과 HTTP 메소드 정보를 가진다.

```
private HandlerKey createHandlerKey(RequestMapping rm) {
    return new HandlerKey(rm.value(), rm.method());
}
```

- HandlerExecution은 자바 리플렉션에서 메소드를 실행하기 위해 필요한 정보를 가진다. 즉, 실행할 메소드가 존재하는 클래스의 인스턴스 정보와 실행할 메소드 정보(java.lang.reflect.Method)를 가져야 한다.

실습

AnnotationHandlerMapping 클래스에 클라이언트 요청 정보(HttpServletRequest)를 전달하면 요청에 해당하는 HandlerExecution을 반환하는 메소드를 구현한다.

힌트

- HandlerExecution getHandler(HttpServletRequest request); 메소드를 구현한다.

레거시 MVC와 새로운 MVC 통합 힌트
—

실습

RequestMapping, AnnotationHandlerMapping은 요청 URL과 실행할 컨트롤러 클래스 또는 메소드를 매핑하는 역할은 같다. 단지 다른 점이라면RequestMapping은 개발자가 수동으로 등록하고, AnnotationHandlerMapping은 애노테이션을 설정하면 자동으로 매핑한다는 점이다. 두 클래스의 공통된 부분을 인터페이스로 추상화한다.

힌트

- HandlerMapping 이름으로 인터페이스를 추가한다.

```
public interface HandlerMapping {
    Object getHandler(HttpServletRequest request);
}
```

- RequestMapping 클래스를 LegacyHandlerMapping과 같은 형태로 클래스 이름을 변경한다.

실습

DispatcherServlet의 초기화(init() 메소드) 과정에서LegacyHandlerMapping, AnnotationHandlerMapping 모두 초기화한다. 초기화한 2개의 HandlerMapping을 List로 관리한다.

실습

DispatcherServlet의 service() 메소드에서는 앞에서 초기화한 2개의 HandlerMapping에서 요청 URL에 해당하는 Controller를 찾아 메소드를 실행한다.

힌트

```
Object handler = getHandler(req);
if (handler instanceof Controller) {
    ModelAndView mav = ((Controller)handler).execute(req, resp);
} else if (handler instanceof HandlerExecution) {
    ModelAndView mav = ((HandlerExecution)handler).handle(req, resp);
} else {
    // throw exception
}
```

실습

기존 컨트롤러를 새로 추가한 애노테이션 기반으로 설정한 후 정상적으로 동작하는지 테스트한다. 테스트에 성공하면 기존의 컨트롤러를 새로운 MVC 프레임워크로 점진적으로 변경한다.