

고급웹프로그래밍

**4차 과제**

**학번** : 201710553

**이름** : 강지우

**학번** : 201715570

**이름** : 김민지

**학번** : 201715575

**이름** : 남수진

**학번** : 201710581

**이름** : 조혜원

**학번** : 201715591

**이름** : 주예나

**목차**

1. Spring FrameWork 실행결과

1.1 Spring FrameWork 소개

1.2 IoC

1.3 AOP

1.4 MVC

2. Spring FrameWork 변경 사항

2.1 Spring FrameWork 소개

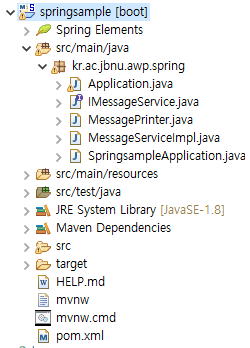
2.2 IoC

2.3 AOP

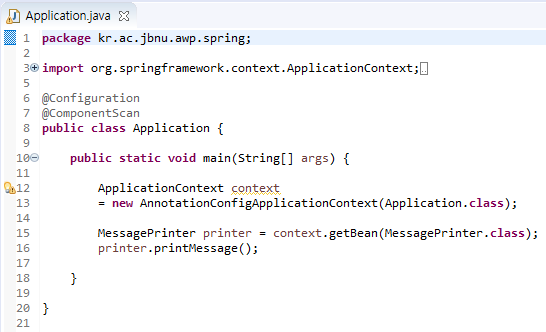
2.4 MVC

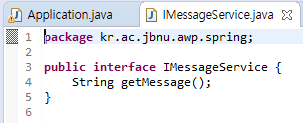
**1.** **Spring FrameWork 실행결과**

**1.1 Spring FrameWork 소개**

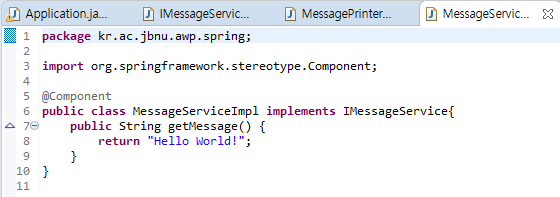


**예제 전체 소스코드**

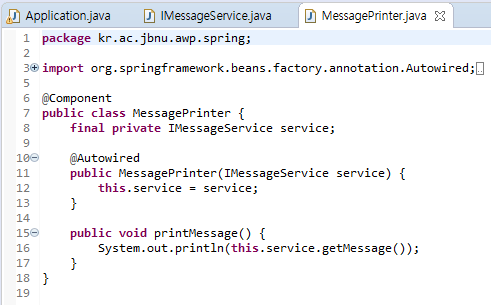




**getMessage() 메소드를 가지는 IMessageService 인터페이스**



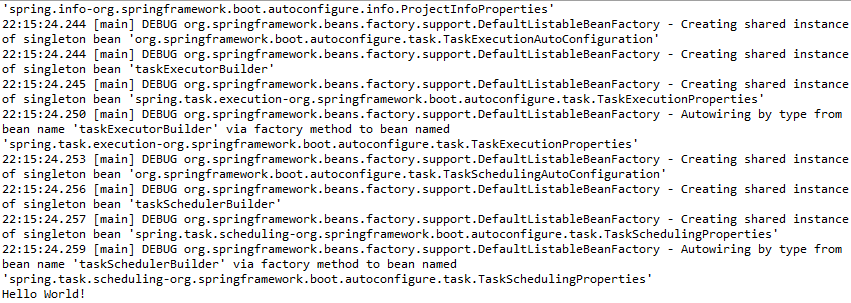
**IMessageService 인터페이스를 구현한 클래스**



**IMessageService 인터페이스를 생성자의 파라미터로 받고, printMessage() 메소드를 호출하면 입력 받은 인터페이스에 따라 다른 문구를 출력하는 클래스.**

@Component: 스프링 컨테이너가 사용하고 관리하기 위한 클래스라는 것을 선언. (없으면 스프링 컨테이너가 사용 불가)

@Autowired: 스프링 컨테이너가 사용하고 관리하는 클래스 중 타입이 맞는 것을 자동으로 입력. Spring 설정 파일을 보고 자동으로 속성의 set 메서드에 해당하는 역할 수행. 별다른 선언 없이 MessageService 클래스의 값을 자동으로 가져오는 것.

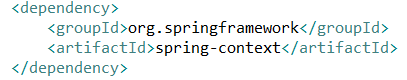
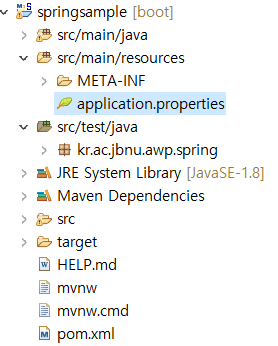


**실행 결과**

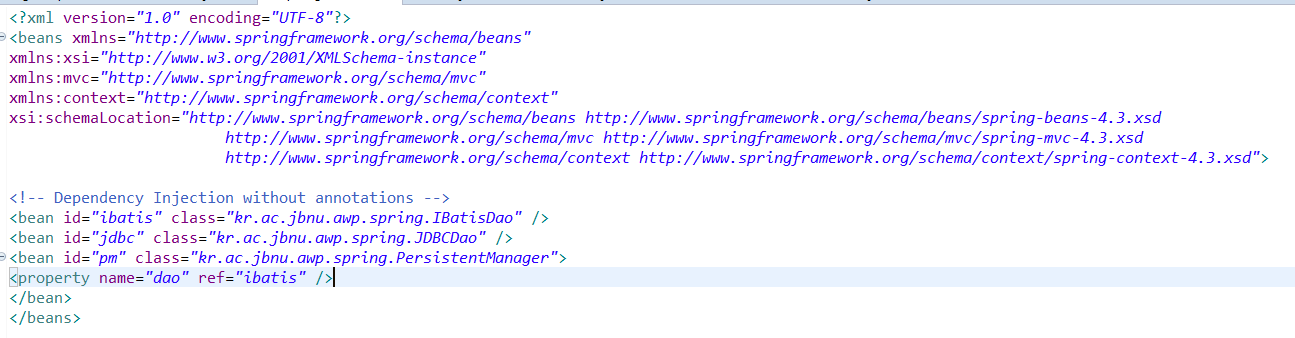
“Hello World!” 가 출력된 모습

**1.2 IoC**

**META-INF 폴더 생성 및 spring-context.xml 파일 생성**



예제를 실습하기 앞서 먼저 META-INF파일을 src/main/resources 아래에 생성해 준 후 pom.xml 파일에 종속성을 설정해준다.

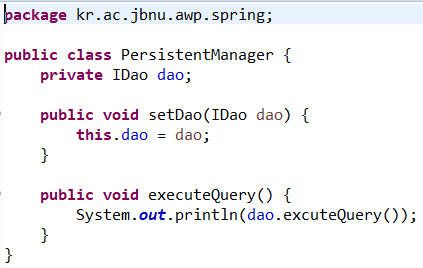


*Spring-context.xml* 파일을 다음과 같이 생성해 준다. 이때 서버 버전과 spring boot의 버전을 확인하여 넣어주어야 한다.

**클래스 생성**

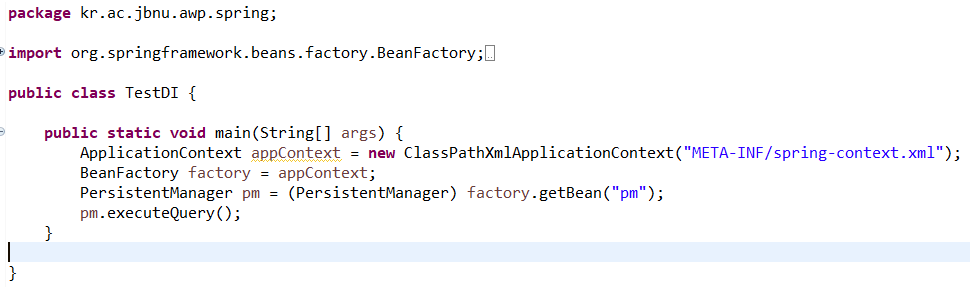
|  |  |
| --- | --- |
| *IBatisDao.java* | *IDao.java(interface)* |
| *JDBCDao.java* |

IDao 인터페이스를 구현하는 각각 다른 클래스 IBatisDao와 JDBCDao 클래스를 만들고 서로 다른 내용을 출력하도록 한다.



*PersistentManager.java*

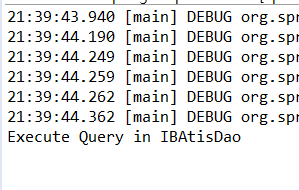
실제 DB를 사용하려는 Manager 객체를 만들어 준다.



*TestDI.java*

일반적인 의존 관계였다면 Manager가 현재 사용해야 하는 DB가 아닌 다른 DB를 사용해야 할 경우 PersistentManager의 상당 부분을 바꿔야 할 것이다. 하지만 IOC에서는 IDao를 인터페이스로 바꾸고 이 IDao를 구현하는 각각의 DB(IBatisDao, JDBCDao)클래스를 만들었다. 이렇게 하여 Manager는 자신이 DB를 직접 바꾸는게 아니라 상황에 따라 필요한 DB를 TestDI 클래스에서 spring-context.xml을 기반으로 매칭 시켜주고 줌으로써 관계가 역전된 것을 볼 수 있다.

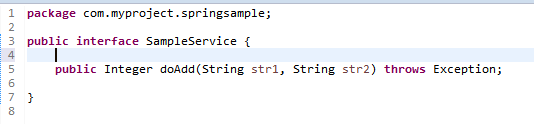
**실행 결과**



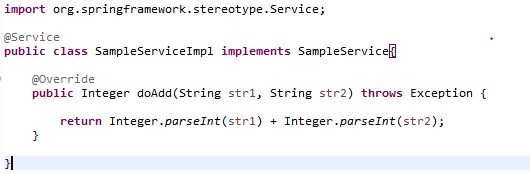
IBAtisDao.java클래스의 메소드가 출력 된 것을 볼 수 있다.

**1.3 AOP**

강의자료에 있는 실행코드를 실행해 보는데 어려움이 있어서 인터넷이서 변수 2개를 받아 더하는 기능의 코드로 AOP예제를 실행해보았다.

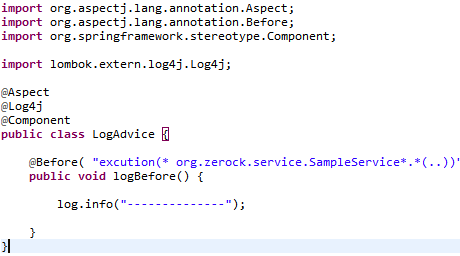


<SampleServie.interface>



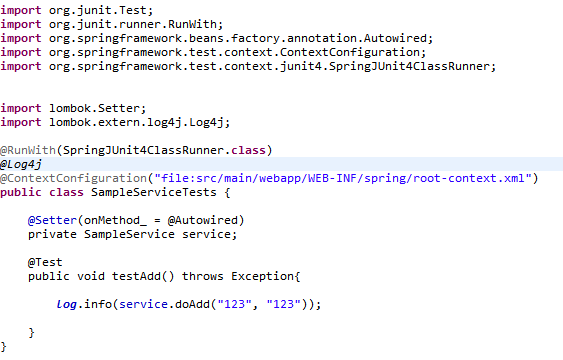
<SampleServieImpl.java>

위 코드는 두 변수 값을 받아 계산하는 메소드를 포함한 코드이다. 그리고 SampleServiceImpl은 SampleService를 상속받고 있다.



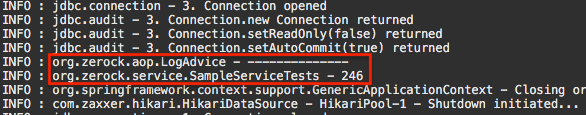
<LogAdvice.java>

Log.info()부분이 보통 반복되어 사용되기 때문에 LogAdvice클래스로 분리하여 작성해 주었다. AOP의 가장 기초가 되는 개념이 필요하지만 중복해서 작성해야 하는 코드 이외는 외부로 뺴놓는 것이기 때문이다.



<SampleServiceTests.java>

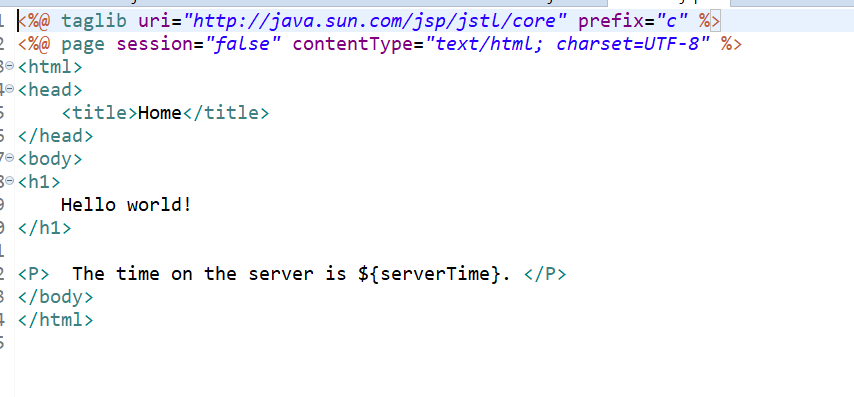
이렇게 테스트 코드를 123와 123 정수값을 doadd()함수에 넣어주고 실행이 되게 된다.



<실행결과>

이 화면이 그렇게 실행한 테스트 코드의 결과이다. Before advice에 맞게 logbefore()가 먼저 실행되고 doadd()가 실행된 것을 확인할 수 있다.

**1.4 Spring MVC**

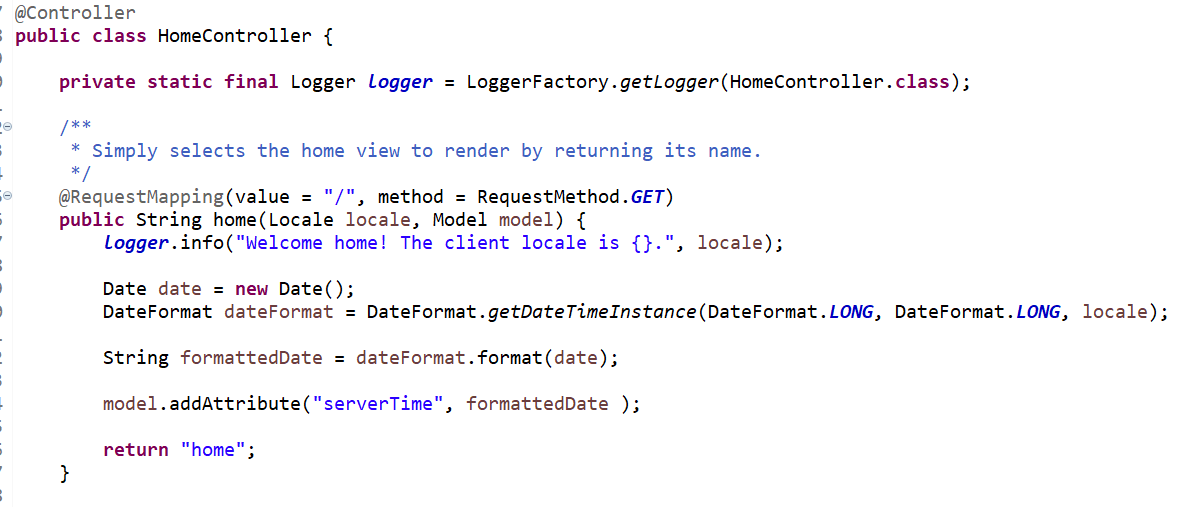


<home.jsp>

Home.jsp파일은 뷰를 담당하는 파일이다.  <P> The time on the server is ${serverTime}. </P>는 브라우저에서 ‘2019년 11월 18일 (월) 오후 2시 32분 17초. ‘와같이 보이도록 하는 부분이다. ${serverTime}을 EL(Expression Language)라고 한다. serverTime 변수의 값을 읽어서 출력(${변수})을 하는 역할을 수행한다. home.jsp 파일의 위치를 살펴보자.

WEB-INF/폴더 아래는 브라우저에서 접근하지 못하는 경로이다. WEB-INF/views/안에 있기 때문에 직접 브라우저에서 호출할 수 없다. 보통은 스프링 프레임워크를 타고 jsp를 호출하게 된다.



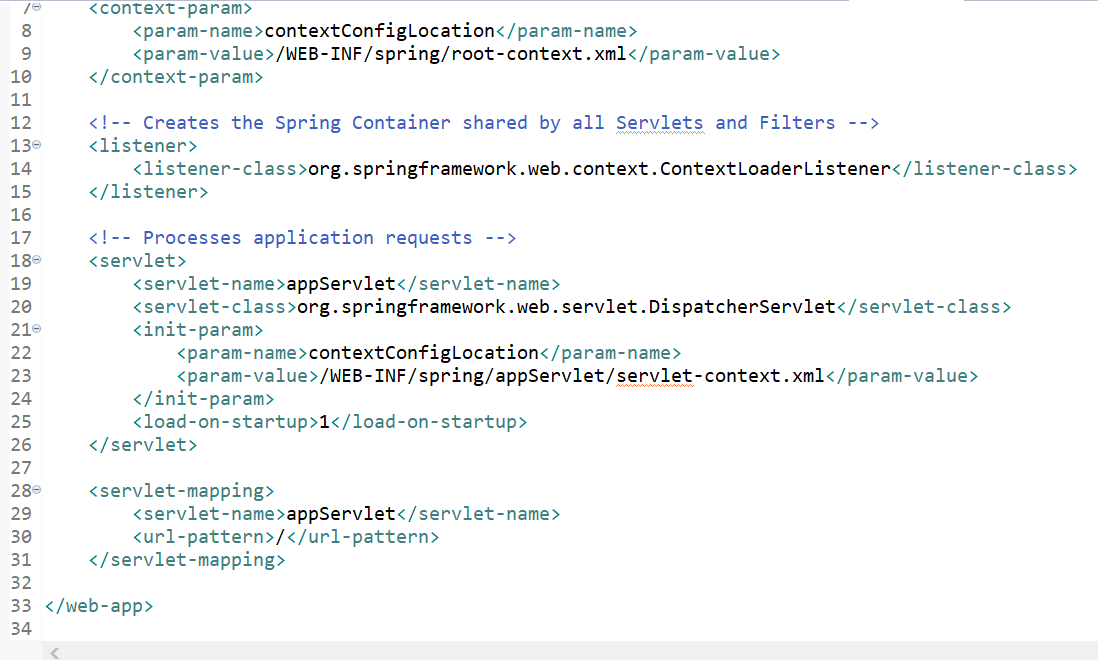


<HomeController.java>

위는 뷰를 담당하는 jsp파일을 호출하는 HomeController.java이다. 그 위에 있는 model.addAttribute("serverTime", formattedDate ); 문장에서 "serverTime" 이라는 명칭으로 formattedDate 값이 model에 속성으로 추가되었다. 다시 올라가 model을 보면 public String home(Locale locale, Model model) 메소드에서 온 변수이다. model을 통해서 Controller에서 정한 값이 JSP에 전달되고 EL로 표시된다고 알 수 있다.

메소드 선언 위에 주석처럼 존재하는 것은 어노테이션(annotation)이라고 한다. @RequestMapping(value = "/", method = RequestMethod.GET) 요청 매핑을 정의하고 있다. 브라우저에 "/" 주소로 들어오고 GET방식이라면 아래 메소드를 실행하라는 의미이다.

@Controller 어노테이션으로 이 클래스가 Controller 역할을 하게 된다.



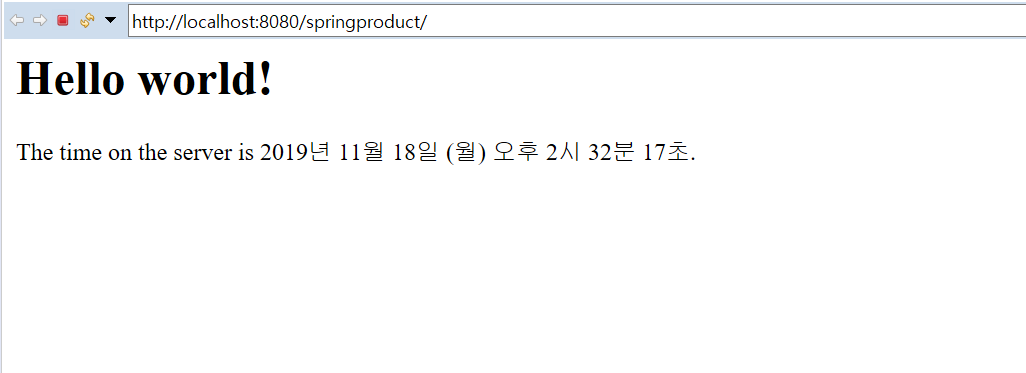
**<web.xml>**

브라우저의 요청을 가장 먼저 처리하는 곳이 web.xml이다. web.xml 중간을 보면 DispatcherServlet이라고 되어있는 것을 볼 수 있다. 브라우저의 요청이 오면 먼저 Controller로 요청이 넘어가기 전에 DispatcherServlet이 그 요청을 가로챈다. 가로챈 요청은 아래의 <init-param> 부분에 있는 servlet-context.xml로 전달된다. 즉 web.xml은 웹 애플리케이션의 실행 환경에 관련된 각종 설정 정보가 담겨있는 파일이다.



<servlet-context.xml>

servlet-context는 브라우저의 요청으로부터 그 요청을 처리할 Controller를 이어주는 작업을 한다. 즉 servlet-context에서 HandlerMapping, HandlerAdapter의 작업이 진행된다.

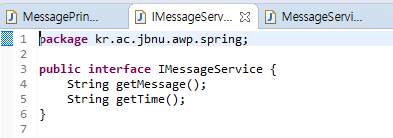


<결과>

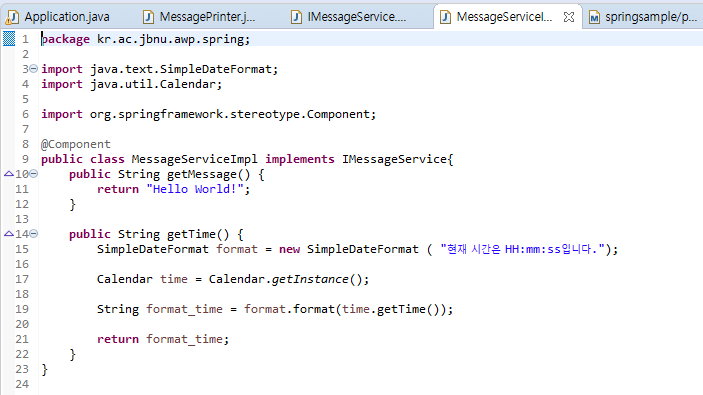
**2. Spring FrameWork 변경사항**

**2.1 Spring FrameWork 소개**

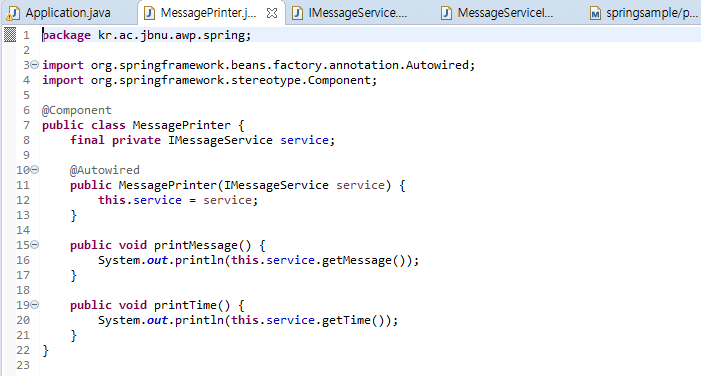
예제에서는 Hello World 구분을 출력하는 실습을 해 보았다. 여기에 현재 시간을 출력하는 구문을 추가해보았다.



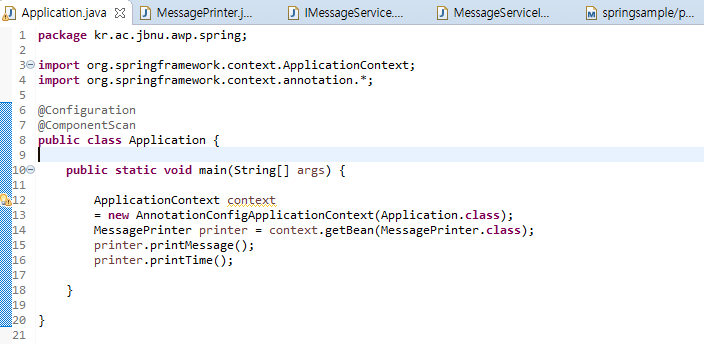
IMessageService 인터페이스에 getTime() 선언



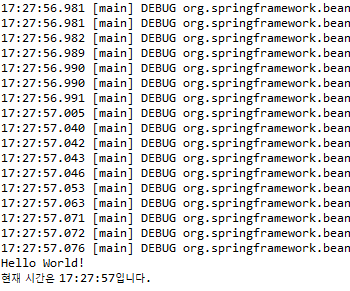
IMessageService 인터페이스를 구현한 MessageServiceImpl 클래스에 현재 시간을 가져오는 getTime() 메소드 구현



getTime()으로 받아온 시간 값을 출력하는 메소드 printTime() 추가



메인 함수에서 printTime() 호출



실행 화면

**2.2 IoC**

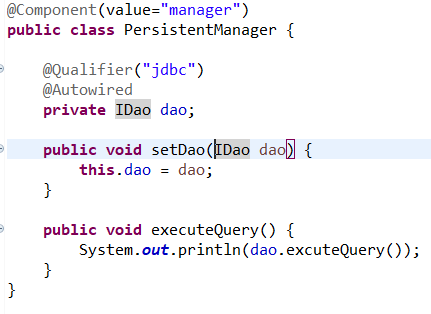
위의 예제는 spring-context.xml 파일을 이용하여 annotation 없이 의존성을 주입해 주었다. 따라서 위의 예제 코드를 annotation을 사용하여 의존성을 주입해 주도록 변경해 보았다.

|  |  |
| --- | --- |
| *IBatisDao.java* | *JDBCDao.java* |

@Component를 통해 클래스자체를 각각 ‘ibatis’ 와 ‘jdbc’라는 이름으로 Bean으로 등록해준다.

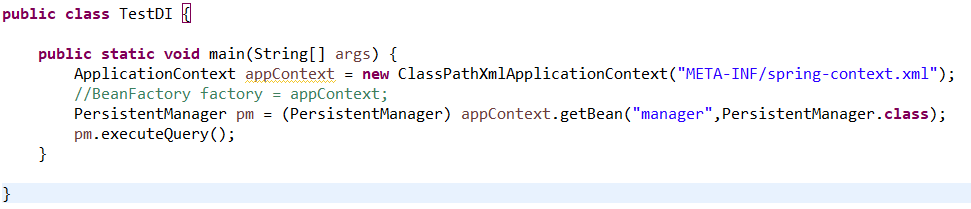


Pom.xml 파일에 component-scan 태그로 component로 등록된 클래스를 인식해준다.



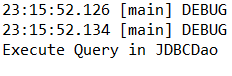
PersistentManager 클래스는 의존성을 부여받는 클래스 이지만 @Component를 이용해 빈즈에 등록하여 준다.

이후 특정 빈을 타입의 세트로 한정시켜서 가져올 수 있는 @Qualifier 어노테이션으로 빈즈를 가져와 @AutoWired 어노테이션으로 의존성을 자동으로 부여 시켜준다.



ApplicationContext 객체를 통해 빈즈를 가져와 의존성이 부여된 객체 pm의 메소들을 호출한다.

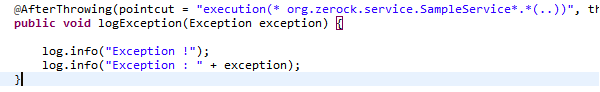
실행 결과



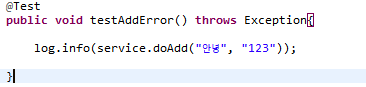
자동으로 PersistentManager객체에 JDBCDao.java클래스로 의존성을 부여 받아 해당 메소드가 출력 된 것을 볼 수 있다.

**2.3 AOP**

위에서 찾아보고 실행해본 코드는 before에 관한 부분에 대해서만 실행해 보아서 다른 advice인 지정된 대상이 예외가 발생한 후 동작하면서 문제를 찾아주는after throwing으로 변경하여 실행해보았다.

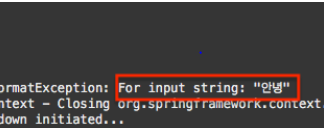


<변경된 LogAdvice.java>



<변경된 테스트 코드>

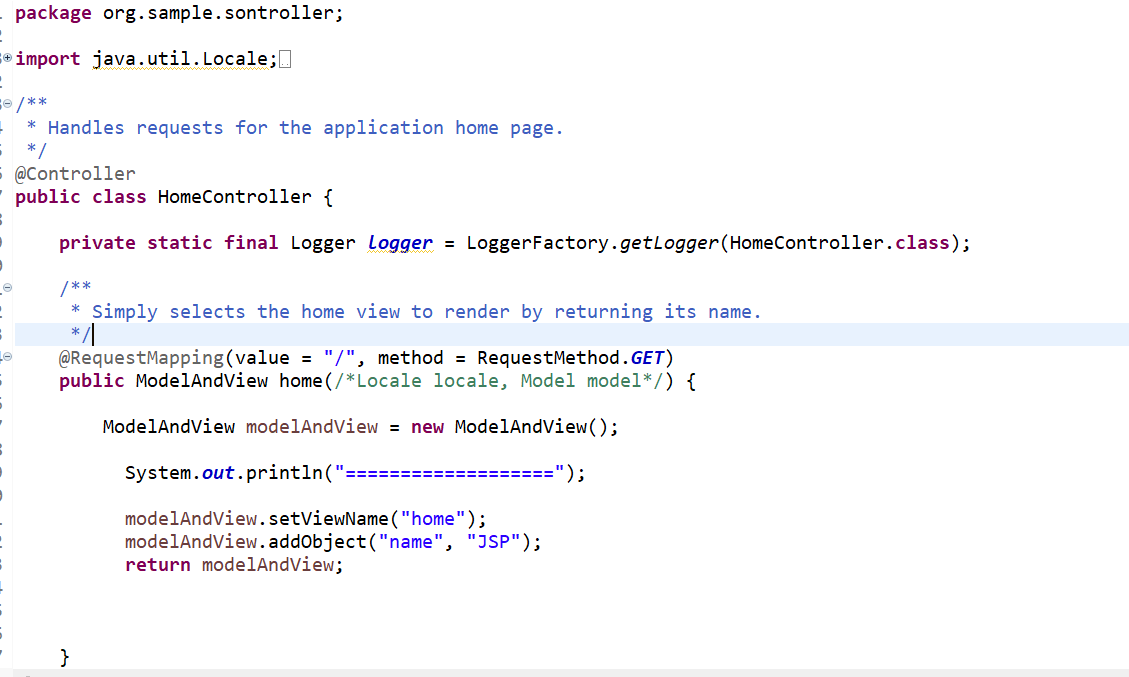
SampleServiceTests 클래스에서 @AfterThrowing에 맞게 테스트 해볼 수 있도록 코드를 수정하였다.



<변경된 실행 결과>

위의 실행결과 같이 input이 exception을 발생시켰는지 확인할 수 가 있다. 즉, 지정한 대상이 아닌 안녕이라는 값이 출력이된것이다.

**2.4 Spring MVC**



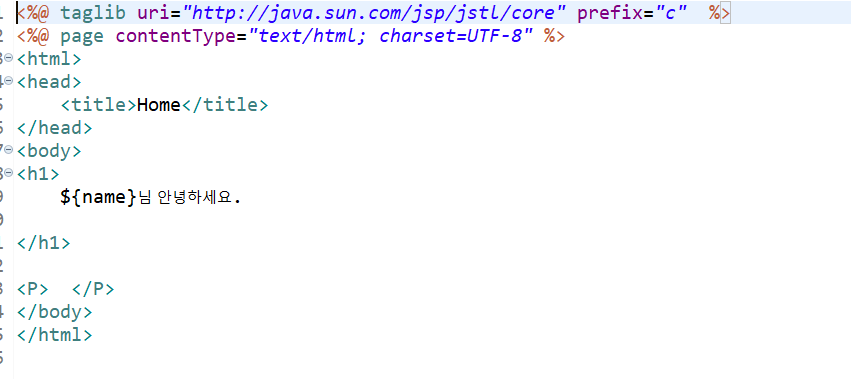
<HomeController.java>

ModelAndView를 사용

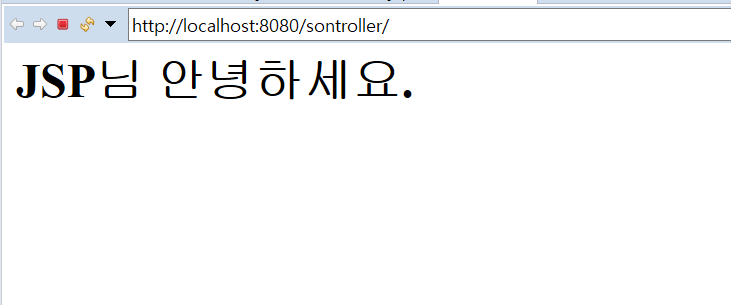
- Model과 View를 동시에 설정이 가능하며 컨트롤러는 ModelAndView객체만 리턴하지만 Model과 View가 모두 리턴 가능

- addObject("key",value) 데이터 입력

Model을 사용할 때는 String 리턴값으로 넘긴 페이지를 설정하였지만 ModelAndView에서는 setViewName()메소드를 이용한다. addObject(String name, Object value)는view에 전달할 값을 설정하는 메소드이다. 반환 값으로는 modelandview의 객체를 반환한다.



<home.jsp>



<결과>