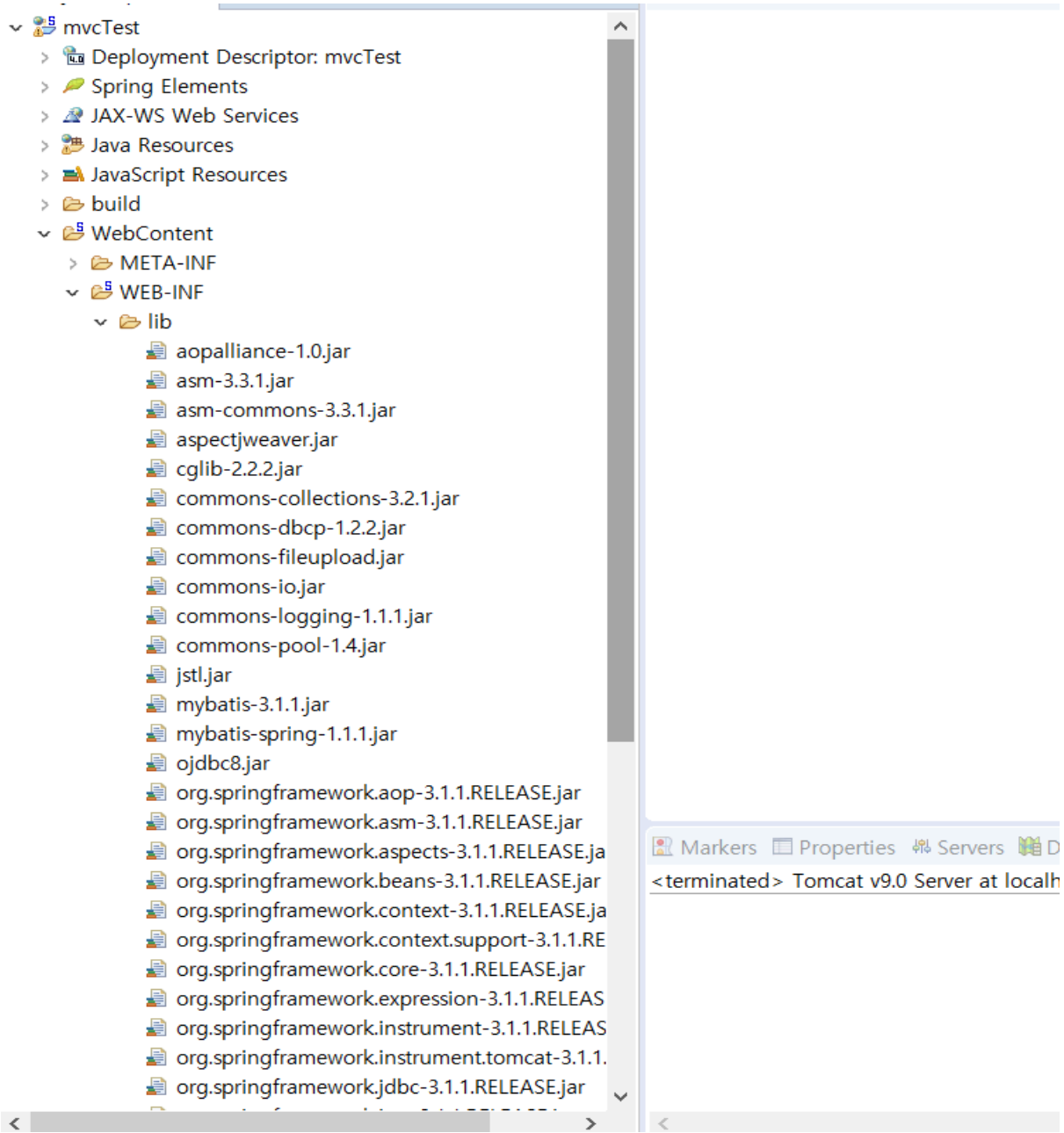
spring MVC (프레임워크 의존 **)**

현재의 스프링(스프링부트) 환경 에서는 많은 부분이 자동완성이 되고 번거로운 환경설정을 하지 않아도 됩니다. 우리는 스프링 부트 학습하기 전에 옛날 방식의 스프링 MVC 패턴을 경험 해 보도록 합니다.

1) 프로젝트 생성 및 라이브러리 준비



옛날 스프링은 일일이 필요한 jar를 해당싸이트 방문하여 다운받아 설정해야 합니다.

앞으로 학습하게 될 스프링 부트 환경에서는
일일이 해당싸이트 방문하지 않고
필요한 명세를 선택만 해주면 알아서 다운로드 해 줍니다.

2) web.xml 설정

web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://xmlns.jcp.or
  <display-name>mvcTest</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>

  <servlet>
    <servlet-name>kosta</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>kosta</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

jsp mvc 패턴 학습할 때에는 프론트컨트롤러 역할을 하는 DispatcherServlet을 직접 만들었는데 스프링에서는 스프링이 DispatcherServlet을 제공 해 주고 이것을 사용하겠다고 web.xml에 설정 해 줍니다.

이때 DispatcherServlet를 설정한 서블릿 이름을 kosta로 설정하게 되면 /WEB-INF 폴더아래에 kosta-servlet.xml를 만들어 주어야 합니다.
스프링은 이 파일을 beanConfiguration 파일로 인식합니다.
여기에 어플리케이션에 필요한 모든 객체를 생성 해 주어야 합니다.

앞으로 학습하게 될 스프링 부트환경에서는
이러한 부분이 생략되고 어노테이션으로 설정하게 됩니다.

3) bean configuration 파일 생성

kosta-servlet.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springf

</beans>

```

web.xml에 설정한 서블릿 이름이
kosta 이므로 WEB-INF 폴더에 kosta-servlet.xml를 객체제공자로서의 역할을 수행하는 beanConfiguration 파일로 만들어야 합니다.

4) Controller의 작성

HelloController.java

```java
package com.kosta.controller;

import javax.servlet.http.HttpServletRequest;


public class HelloController implements Controller {

    @Override
    public ModelAndView handleRequest(HttpServletRequest arg0, HttpServletResponse arg1) throws E
        System.out.println("컨트롤러 동작함.");
        ModelAndView mav = new ModelAndView();
        mav.addObject("msg", "hello 스프링");
        mav.setViewName("/WEB-INF/views/hello.jsp");
        return mav;
    }

}
```

jsp MVC 패턴에서 학습했던 Action의 역할을 스프링 MVC에서는 Controller라고 합니다.
컨트롤러가 되기 위해서는 스프링이 제공하는 Controller 인터페이스를 구현해야 하고

실제로 사용자가 요청했을때에 동작하기 위한 메소드는 handleRequest메소드를
오버라이딩하여 그 안에 써줍니다.

또, 이 메소드는 사용자의 요청에 대한 처리된 결과 Model와 어디로 가야 할 지 View를
하나의 세트로 표현하는 ModelAndView를 반환하도록 해야 합니다.

msg이라는 어트리뷰터 이름으로 데이터를 상태유지 하고
/WEB-INF/views 폴더의 hello.jsp로 포워드 하도록 view페이지를 설정한다.
우리가 학습했던 jsp MVC 패턴에서는 우리가 직접 해당 jsp로 포워드 하도록
코딩하였는데 여기서는 DistpatcherServlet이 알아서 포워딩 해 준다.

이와 같이 컨트롤러를 만들기 위하여 Controller인터페이스를 구현해야 하고 그
인터페이스의 추상메소드인 handleRequest를 오버라이딩 해야만 컨트롤러를 만들 수
있는 이러한 방식을 “프레임워크 의존적”이라고 표현합니다.

이어서 학습하게될 현재의 스프링버전에서는 “프레임워크 의존적”이지 않고
어떤 인터페이스를 구현할 필요도 없고 메소드이름도 자유롭게 원래 자바 클래스
만들듯이 작성하여 컨트롤러를 만들수 있습니다. 이러한 방식은 **POJO(Plain Old Java Object)**, 방식 이라고 합니다.

5) bean configuration의 구체화

kosta-servlet.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.o

    <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
        <property name="mappings">
            <props>
                <prop key="/kosta.do">helloController</prop>
            </props>
        </property>
    </bean>

    <bean id="helloController" class="com.kosta.controller.HelloController"/>

</beans>
```

bean Configuration 파일인 kosta-servlet.xml 파일에
어플리케이션에서 필요한 모든 객체를 생성 해 주어야 합니다.

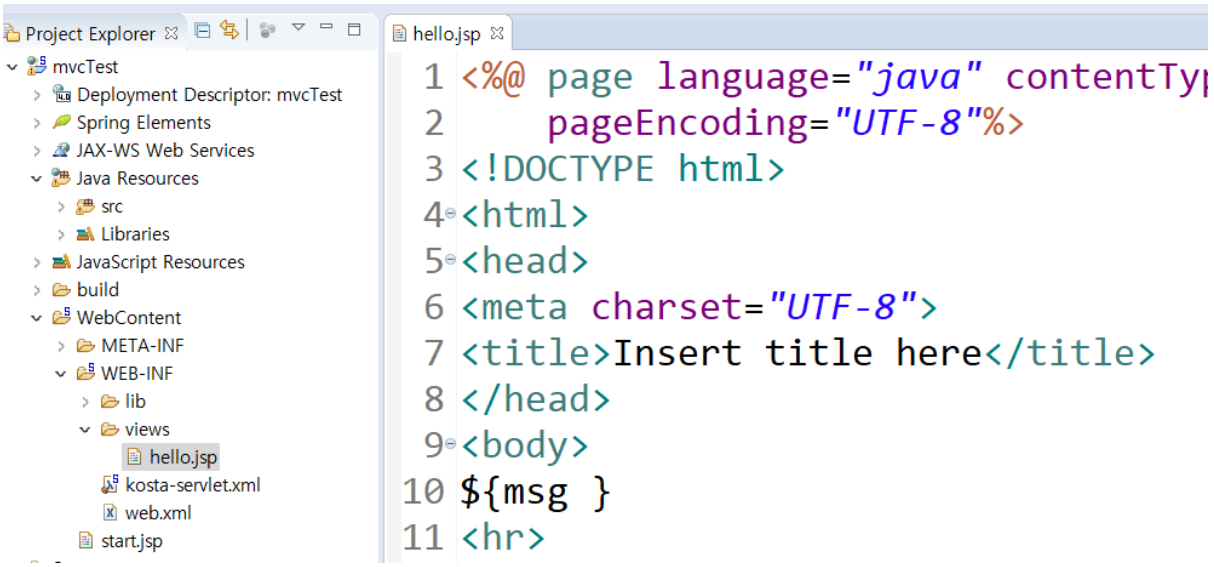
우리가 작성했던 HelloController객체를 생성 해 주고

사용자가 어떠한 요청일 때에 그 컨트롤러가 동작할 것인지
HandelerMapping으로 설정 해 줍니다.

어플리케이션 규모가 복잡해 지면
xml이 비대해지게 되어 가독성이 어렵게 됩니다.

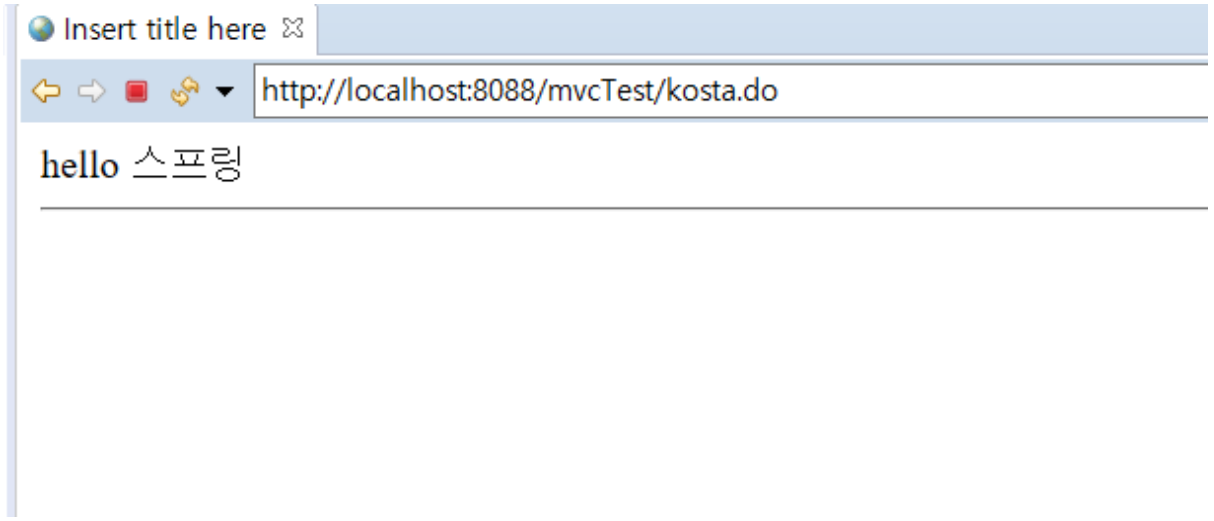
이에 현재 스프링 버전에서는
이 부분이 모두 생략되고
어노테이션 기반으로 자동화 됩니다.

6) view의 작성



컨트롤러에서 설정한 view의 위치인 /WEB-INF 폴더아래에 views 폴더를 만들고 hello.jsp를 작성하고 상태유지한 msg를 표현언어로 출력한다.

7) 실행결과 확인



8) ViewResolver의 사용

```
public class HelloController implements Controller {

    @Override
    public ModelAndView handleRequest(HttpServletRequest arg0, HttpServletRespc
        System.out.println("컨트롤러 동작함.");
        ModelAndView mav = new ModelAndView();
        mav.addObject("msg", "hello 스프링");
        mav.setViewName("/WEB-INF/views/hello.jsp");
        return mav;
    }

}
```

prefix

suffix

우리는 view를 한곳에 모아 둘 것이며 또 view의 확장자가 jsp라고 할 때에
매번 동일한 내용을 일일이 작성하는 것은 번거로운 일이다.
따라서 보통은 뷰 이름을 설정할 때에 view의 위치인 prefix와 view확장자를 생략하고
view이름만 설정하게 된다.

생략하게 된 모습은 다음과 같다.

```
public class HelloController implements Controller {

    @Override
    public ModelAndView handleRequest(HttpServletRequest arg0, H
        System.out.println("컨트롤러 동작함.");
        ModelAndView mav = new ModelAndView();
        mav.addObject("msg", "hello 스프링");
        mav.setViewName("hello");
        return mav;
    }
}
```

view의 위치와 확장자를 생략하고 view설정

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
</bean>
```

뷰를 찾을 위치

뷰의 확장자

view의 위치와 view의 확장자를 생략했으므로
beanConfiguration 파일에서 뷰를 찾을 위치와 뷰의 확장자를 설정해야 한다.
이러한 역할을 수행하는 것을 ViewResolver라고 한다.