8_30_2022_SPRING

@RequestMapping("/send1")
public String send1(HttpServletRequest request) {
//msg 라는 파라미터 명으로 전송되는 문자열 추출하기
String msg=request.getParameter("mag");
//콘솔창에 출력하기
System.out.println("msg:"+msg);
return "result";

Spring MVC Project

Model View Controller

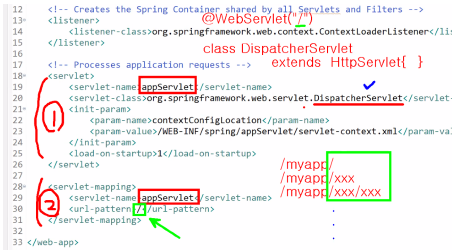
package 이름의 마지막 문자열이 context 경로가 됨

/myapp/ 에 대한 root 요청은 home.jsp가 함

```
//태그 라이브러리도 설정 되어 있음.
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ page session="false" %>
<html>
<head>
  <title>Home</title>
</head>
<body>
<h1>
  Hello world!
</h1>

<P>  The time on the server is ${serverTime}. </P>
</body>
</html>
```

SPRING MVC

- Maven 프로젝트
  - java로 만드는 어떤 프로젝트든 Maven으로 만들 수 있다.
    - java app, dynamic web project, spring mvc project등
  - pom.xml이 만들어지는 특징이 있다.
    - 프로젝트의 설정, 주로 의존 라이브러리의 목록이 들어있다.
    - 필요한 의존 라이브러리가 있다면 pom.xml에 dependency 설정을 추가하는 것 만으로 자도 다운로드 되고 사용할 준비가 된다.
  - Maven 프로젝트를 만들기 위해서는 Maven을 다운로드 해 사용해야 한다.
    - https://maven.apache.org/ 에서 다운받아 설치
  - 필요한걸 검색해서 pom.xml →dependencies 안에 붙여넣으면 사용할 수 있다.
    - Libraries →Maven Dependencies로 추가된다.
  - web.xml 문세어 스프링 DispatcherServlet 설정이 있는데 해당 설정에 의해서
    스프링 프레임워크가 동작하는것이다.



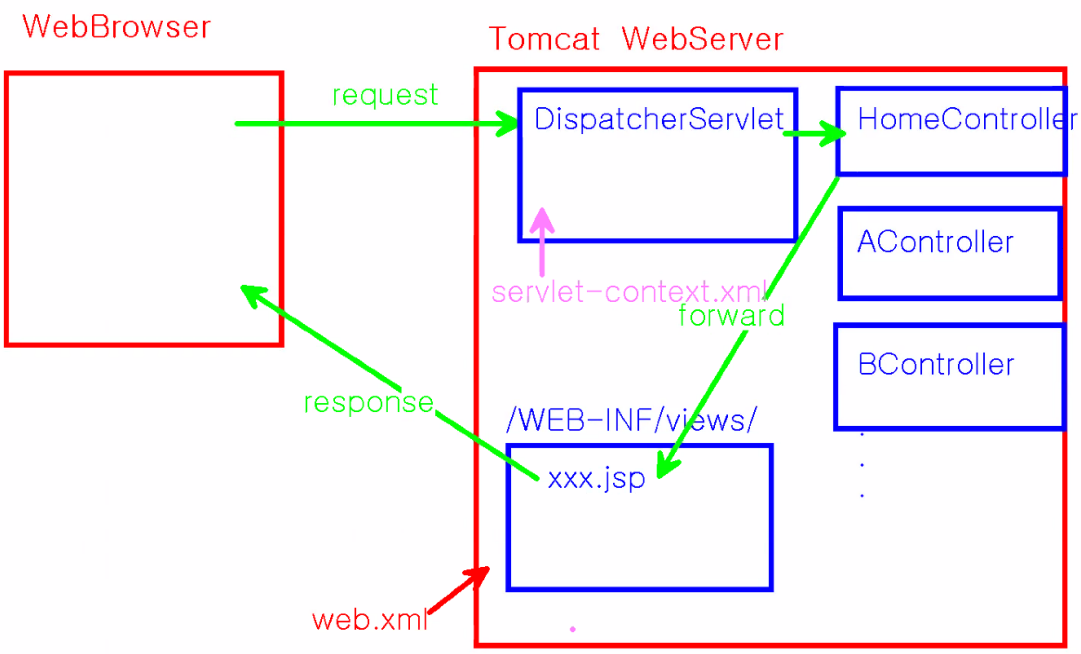
"/" == @WebServlet
모든 요청을 dispatcherServlet으로 받는다.
이 설정이 없으면 spring 프로젝트가 아니게 됨

.XXX를 사용해 이용할 수 있다.

- DispatcherServlet은 미리 만들어 놓은 서블릿인데 만일 해당 서블릿의 동작을 바꾸려면
  - 미리 만들어진 Apache Tomcat 서버의 설정은 web.xml로 할 수 있듯. DispatcherServletdml 동작도 xml문서로 설정할 수 있다.
    servlet-context.xml 문서에 설정을 하면 된다.

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

  <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->

  <!-- Enables the Spring MVC @Controller programming model -->
  <annotation-driven />

  <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the ${webappRoot}/resources 
  <resources mapping="/resources/**" location="/resources/" />

  <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
  <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />접두어
    <beans:property name="suffix" value=".jsp" />접미어
  </beans:bean>

  <context:component-scan base-package="com.dubu.myapp" />

</beans:beans>
```

- servlet-context.xml에는 스프링 프레임 워크가 동작하는데 있어서 필요한 설정을 하는 곳이다.
- 동작하는 방식



Tomcat의 설정은 web.xml 로 하고
dispatcher의 설정은 servlet-context.xml로 함

- MVC
  M → Model
  V → View
  C → Controller
  를 의미한다
- 스프링 프레임 워크에서
  Model은 data이고
  View 는 /WEB-INF/views/ 폴더 안에 있는 jsp페이지 이고
  Controller 는 HomeController, AController..등등 @Controller 어노테이션으로 만든 객체를 의미한다.

Spring Framework

- 왜 사용하는가?
  - 개발 단계에서는 까다롭고, 코딩량은 늘어나지만
    한번 개발해 놓으면 유지보수가 용의하다
- 왜 유지보수가 용의한가?
  - 객체들간의 의존관계가 느슨해져서 용의하다.
- 어떻게하면 객체들간의 의존관계를 느슨하게 할 수 있을까?
  - 필요한 핵심의존 객체의 생성과 관리를 Framework에 맡긴다.
  - interface type을 적극 활용한다.
    - ex> dao가 필요하다면 직접 new 하지 말고 framework에서 받아온다.

Step01_Basic

▼ 기초 설정

/pom.xml(복붙하셈)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.livelikesloth</groupId>
  <artifactId>step01</artifactId>
  <name>Step01_Basic</name>
  <packaging>war</packaging>
  <version>1.0.0-BUILD-SNAPSHOT</version>
  <properties>
    <java-version>1.8</java-version>
    <org.springframework-version>4.0.0.RELEASE</org.springframework-version>
    <org.aspectj-version>1.6.10</org.aspectj-version>
    <org.slf4j-version>1.6.6</org.slf4j-version>
  </properties>
  <dependencies>
    <!-- Spring -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${org.springframework-version}</version>
      <exclusions>
        <!-- Exclude Commons Logging in favor of SLF4j -->
        <exclusion>
          <groupId>commons-logging</groupId>
          <artifactId>commons-logging</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${org.springframework-version}</version>
    </dependency>

    <!-- AspectJ -->
    <dependency>
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjrt</artifactId>
      <version>${org.aspectj-version}</version>
    </dependency>

    <!-- Logging -->
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>${org.slf4j-version}</version>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>jcl-over-slf4j</artifactId>
      <version>${org.slf4j-version}</version>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-log4j12</artifactId>
      <version>${org.slf4j-version}</version>
      <scope>runtime</scope>
    </dependency>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.15</version>
      <exclusions>
        <exclusion>
          <groupId>javax.mail</groupId>
          <artifactId>mail</artifactId>
        </exclusion>
        <exclusion>
          <groupId>javax.jms</groupId>
          <artifactId>jms</artifactId>
        </exclusion>
        <exclusion>
          <groupId>com.sun.jdmk</groupId>
          <artifactId>jmxtools</artifactId>
        </exclusion>
        <exclusion>
          <groupId>com.sun.jmx</groupId>
          <artifactId>jmxri</artifactId>
        </exclusion>
      </exclusions>
      <scope>runtime</scope>
    </dependency>
```

```
<!-- @Inject -->
<dependency>
  <groupId>javax.inject</groupId>
  <artifactId>javax.inject</artifactId>
  <version>1</version>
</dependency>

<!-- Servlet -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>javax.servlet.jsp</groupId>
  <artifactId>jsp-api</artifactId>
  <version>2.2</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>jstl</artifactId>
  <version>1.2</version>
</dependency>

<!-- Test -->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.7</version>
  <scope>test</scope>
</dependency>
<!-- 추가 의존 라이브러리 -->
<!-- MyBatis 라이브러리 -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.2.8</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis-spring</artifactId>
  <version>1.2.0</version>
</dependency>
<!-- Spring JDBC 라이브러리 -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>4.0.0.RELEASE</version>
</dependency>
<!-- 파일업로드 처리를 위한 라이브러리 (SmartEditor 에서도 필요함) -->
<dependency>
  <groupId>commons-io</groupId>
  <artifactId>commons-io</artifactId>
  <version>2.4</version>
</dependency>
<dependency>
  <groupId>commons-fileupload</groupId>
  <artifactId>commons-fileupload</artifactId>
  <version>1.3.1</version>
</dependency>
<!-- json, xml 응답을 편하게 할수 있도록 도와 주는 라이브러리 -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.6.0</version>
</dependency>
<!-- Aop 용 라이브러리 -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>4.0.0.RELEASE</version>
</dependency>
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.8.0</version>
</dependency>
<!-- Spring Security 관련 라이브러리 -->
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>4.0.0.RELEASE</version>
</dependency>
<!-- 트렌젝션 처리를 위한 라이브러리 -->
```

```
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-tx</artifactId>
      <version>4.0.0.RELEASE</version>
    </dependency>
    <!-- 오라클 라이브러리 -->
    <dependency>
      <groupId>com.oracle.database.jdbc</groupId>
      <artifactId>ojdbc8</artifactId>
      <version>21.1.0.0</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-eclipse-plugin</artifactId>
        <version>2.9</version>
        <configuration>
          <additionalProjectnatures>
            <projectnature>org.springframework.ide.eclipse.core.springnature</projectnature>
          </additionalProjectnatures>
          <additionalBuildcommands>
            <buildcommand>org.springframework.ide.eclipse.core.springbuilder</buildcommand>
          </additionalBuildcommands>
          <downloadSources>true</downloadSources>
          <downloadJavadocs>true</downloadJavadocs>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>2.5.1</version>
        <configuration>
          <source>1.8</source>
          <target>1.8</target>
          <compilerArgument>-Xlint:all</compilerArgument>
          <showWarnings>true</showWarnings>
          <showDeprecation>true</showDeprecation>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>1.2.1</version>
        <configuration>
          <mainClass>org.test.int1.Main</mainClass>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

- 프로젝트 우클릭 Maven → Update project

▼ DispatcherServlet의 urlpattern 변경

/WEB-INF/web.xml(복붙하라)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee https://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">

  <!-- 최상위 경로 요청이 왔을때 forward 이동될 경로 -->
  <welcome-file-list>
    <welcome-file>home.do</welcome-file>
  </welcome-file-list>
  <!-- The definition of the Root Spring Container shared by all Servlets
    and Filters -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-context.xml</param-value>
  </context-param>

  <!-- Creates the Spring Container shared by all Servlets and Filters -->
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <!-- Processes application requests -->
  <servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
```

```
    </servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/appServlet/servlet-context.xml
      </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <!-- .do로 끝나는 요청만 DispatcherServlet 을 거치도록 설정한다. -->
  <servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>

</web-app>
```

▼ HomeController.java 단순화

```
package com.livelikesloth.step01;

import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {

  // /home.do 요청이 왔을 때 동작하는 컨트롤러 메소드
  @RequestMapping("/home.do")
  public String home(HttpServletRequest request) {

    // /WEB-INF/views/home.jsp 페이지로 forward 이동해서 응답
    return "home";
  }
}
```

확인하기

```
package com.livelikesloth.step01;



import java.util.ArrayList;
import java.util.List;

import javax.servlet.http.HttpServletRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {

  // /home.do 요청이 왔을 때 동작하는 컨트롤러 메소드
  @RequestMapping("/home.do")
  public String home(HttpServletRequest request) {

    //DB 에서 읽어온 공지사항이라고 가정
    List<String> noticeList = new ArrayList<>();
    noticeList.add("Spring Framework 시작입니다.");
    noticeList.add("화이팅");
    noticeList.add("두부");
    noticeList.add("유키");

    //view page 에 전달할 Model(data)을 request 영역에 담기.
    request.setAttribute("noticeList", noticeList);

    // /WEB-INF/views/home.jsp 페이지로 forward 이동해서 응답
    return "home";
  }
}
```

Step02_RequestParam

▼ home.jsp 에 form 설정

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>/views/home.jsp</title>
</head>
<body>
  <div class="container">
  <h1>인덱스 입니다.</h1>

  <form action="${pageContext.request.contextPath}/send1.do" method="post">
    <input type="text" name="msg" placeholder="할말입력..." />
    <button type="submit">전송</button>
  </form>
  <form action="${pageContext.request.contextPath}/send2.do" method="post">
    <input type="text" name="msg" placeholder="할말입력..." />
    <button type="submit">전송</button>
  </form>


  <h2>공지사항</h2>
    <c:forEach var="tmp" items="${requestScope.noticeList}">
      <li>${tmp}</li>
    </c:forEach>
  </div>
</body>
</html>
```

▼ SendController만들기

요청 파라미터 추출하는 방법1

- 메소드의 인자로 HttpServletRequest 객체를 전달 받은 다음 해당객체의 getParameter() 메소드를 이용해서 추출한다.(서블릿 or jsp 하던 방식 그대로)

```
    @RequestMapping("/send1")
    public String send1(HttpServletRequest request) {
      //msg 라는 파라미터 명으로 전송되는 문자열 추출하기
      String msg=request.getParameter("mag");
      //콘솔창에 출력하기
      System.out.println("msg:"+msg);
      return "result";
```

요청 파라미터 추출하는 방법 2

- 파라미터 명과 동일한 이름의 매개 변수를 선언하면 자동으로 추출되어서 전달된다.
  <input name="msg"> -> String msg
  <input name="num"> -> int num or String num
  xxx.do?msg=xxx -> String msg
  xxx.do?num-xxx -> int num or String num

```
@RequestMapping("/send2")
public String send2(String msg) {
  //콘솔창에 출력하기
  System.out.println("msg : "+msg);
  return "result";
}
```

result.jsp 에서 받아준다.

post 방식 get 방식 요청을 처리하는 방법

```
//post 방식 /add.do 요청을 처리할 메소드
@RequestMapping(value = "/add", method = RequestMethod.POST)
public String add(String content) {
```

```
  System.out.println("post방식 요청 : "+content);
  return "result";
}
//get 방식 /add.do 요청을 처리할 메소드
@RequestMapping(value = "/add", method = RequestMethod.GET)
public String add2(String content) {
  System.out.println("get방식 요청 : "+content);
  return "result";
```