-JSP(Java Server Pages) : Java + html  
jsp 파일을 실행하면 servlet engine을 통해 Java 실행하고 그 결과를 웹브라우저에 띄워줌  
(jsp -> servlet -> java -> html -> wb)

-jar(Java application resource) : Java 파일을 묶어 놓은 것. 실행파일 또는 일종의 라이브러리로 제공되는 경우 있음.

-JSP 기본 태그  
<%@ %> : java 설정 영역  
<% %> : Java 코드 영역. 요즘에는 이 방식보다 core 태그 라이브러리 이용함.

-el 태그 : ${i} 형태. 값을 가져오거나 변경할 때 사용함.

-코어태그를 사용하기 위해서는 문서 상단에 다음과 같은 지시자 넣음.

|  |
| --- |
| <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %> |

-코어태그 주요 사용방법  
1) 반복문

|  |
| --- |
| **<c:forEach var=”i” begin=”1” end=”10” step=”1” varStatus=”s”**>  내용 **</c:forEach>** // 1부터 10까지 i의 값을 1씩 증가하겠다는 의미. status.index : 목록형 데이터인 경우 목록의 인덱스번호 0번부터 몇 번째인지의  인덱스 값을 가져옴 status.count : 몇 개째인지 가져옴    목록형 데이터 출력 예시. 첫 번째 라인의 경우 s.index의 값 0, s.count의 값 1 **<c:forEach** var="data" items="${list}" varStatus="s">  <tr>  <td>${data.no},${s.index},${s.count}</td>  <td>${data.title}</td>  </tr> **</c:forEach>** |

2) 조건문 if(단발성 조건일 때 사용)

|  |
| --- |
| **<c:if** **test="${i % 2 ne 0}"**>  내용 **</c:if>** // i의 값이 짝수가 아니면 내용을 실행하겠다라는 의미. test : 조건 설정eq : 같다(자바에서 ==과 동일) ne : 다르다(자바에서 !=과 동일) |

3) 조건문 choose ~ when otherwis(if ~else if ~ else와 동일)

|  |
| --- |
| **<c:choose>**  **<c:when** test="${i % 2 eq 0}">  <b>${i}</b><br/>  **</c:when>**  **<c:otherwise>**  ${i}<br/>  </c:otherwise> **</c:choose>** // i 값이 짝수이면 i의 값을 볼드로, 그게 아니라면 그냥 i 값 출력 |

-form 태그 : 값을 전송하기 위한 태그

|  |
| --- |
| **<form** **action="Test02s.jsp"** id="actionForm" **method="post"**>  <input type="text" name="msg" />  <input type="button" value="send" id="sendBtn" /> **</form>**  action : 값을 보낼 경로 method : 데이터 전송방식  form으로 값을 전송할 때 name은 key의 역할을, value는 value역할을 함. 키와 값으로 매핑되어 전송됨.(주소/Test02.jsp?키=값&키=값….. 형태)  받는쪽 입장에서 해당 value를 취득하려면 ${param.네임}을 쓰면 됨. |

-데이터 전송 방식 : get방식 / post 방식  
1) get 방식 : 데이터가 주소창에 보이기 때문에 보안에 취약함. 대외적으로 알려야하는 정보가 있다거나 보안상 중요한 부분이 아닐 경우 Get 방식 사용

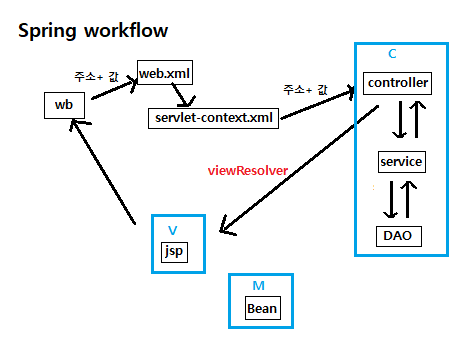
|  |
| --- |
| https://www.google.com/search?source=hp&ei=rHnLXpPCJIz6wQP6g66oBg&q=get&oq=get&gs\_lcp=CgZwc3ktYWIQAzICCAAyAggAMgIIADICCAAyAggAMgIIADICCAAyAggAMgUIABCDATICCABQhRZY2xhg1BloAHAAeACAAZcCiAGMA5IBBTAuMS4xmAEAoAEBqgEHZ3dzLXdpeg&sclient=psy-ab&ved=0ahUKEwiTyOfKxM7pAhUMfXAKHfqBC2UQ4dUDCAc&uact=5 |

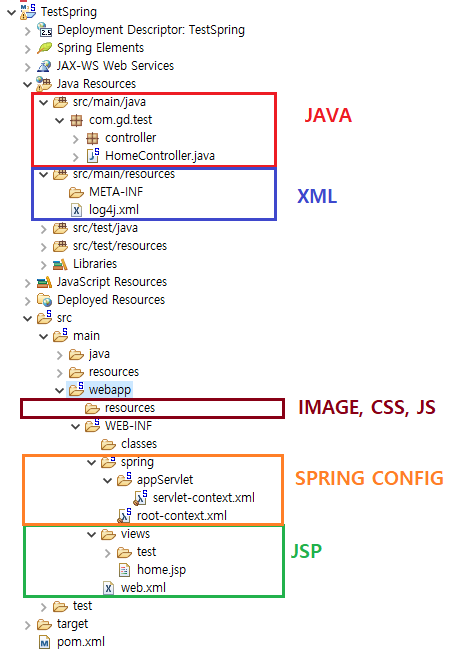
2) post 방식 : 주소에 헤더를 통하여 데이터 전송. 패스워드와 같은 정보는 post방식으로 전송

-MVC 디자인 패턴  
M(Model) : 데이터 담기  
V(View) : 보여주기  
C(Controller) : 조작하기

-JSP Model  
1) JSP Model 1 -> M, V + C : 데이터만 따로 담고, 뷰에서 컨트롤러까지 작업  
2) JSP Model 2 -> M, V, C  
 servlet 구현 필요.  
 주소 -> Servlet -> JSP -> servlet엔진 -> 실행 -> 반환 -> WB 과정을 거침

-framework -> 작업을 위한 틀  
: 구현상 편의나 보안 등 다양한 이점을 확보하기 위하여 개발의 형태를 제시

-스프링 워크플로우  


-스프링 기본 구조  


-xml : 데이터를 보관하고 관리하기 위해 사용(태그 구조)

-servlet-context.xml

|  |
| --- |
| <annotation-driven /> : Annotation을 기능으로 사용하겠다는 의미 <context:component-scan base-package="com.gd.test" />  : component-scan : 클래스 정보취득 범위 정의 <resources mapping="/resources/\*\*" location="/resources/" /> : 이미지 등 파일의 경로 매핑 지정  <beans:bean class="org.springframework.web.servlet.view.**InternalResourceViewResolver**">  <beans:property name="prefix" value="/WEB-INF/views/" />  <beans:property name="suffix" value=".jsp" />  </beans:bean>  **InternalResourceViewResolver** : View의 정보가 넘어올 경우 낚아챔  View정보의 앞과 뒤에 해당 내용을 붙여줌. 위의 경우  ex) test01 정보를 낚아챌 경우 /WEB-INF/views/test01.jsp |

-Controller

|  |
| --- |
| **@Controller** // Annotation을 붙여야 스프링에서 컨트롤러로 인식함  public class TestController {  @RequestMapping(value="/test01") // test01로 요청이 들어오면 아래 메소드 실행  // ModelAndView : 데이터를 담는 역할과 View의 정보를 담고 있음  public **ModelAndView** test01(ModelAndView mav) {  // addObject(키, 값) : 키와 값을 View에 전달  mav.**addObject**("msg", "Hi~!");    List<HashMap<String,String>> list = new ArrayList<HashMap<String,String>>();    for(int i = 10; i > 0; i--) {  HashMap<String,String> data = new HashMap<String,String>();    data.put("no", Integer.toString(i));  data.put("title", "Test" + i);  list.add(data);  }    mav.addObject("list", list);    // setViewName(경로) : View의 위치 정보  mav.**setViewName**("test/test01");    return mav;  }  } |