200806 Spring

- cf
  - 스프링으로 무엇을 하는가?
    - 할 수 있는가?
    - 웹 전용 프레임워크가 아님
  - 전자정부프레임워크
    - 스프링 기반
  - 구조에 대한 이해
    - ex
      - 톰캣에서 직접 돌려보기
        - 톰캣 구조
    - 스프링 전에 jsp 구조를 이해하면 알기 쉬움
  - 이클립스에서 플러그인을 받을 수도 있고 따로 STS라는 IDE를 받아도 됨
  - 프레임워크
    - 짜여진 틀
    - 대규모 프로젝트에 적합?
    - 어노테이션이 많이 쓰이곤함
  - 의뢰물을 줄 때
    - 보통 java파일은 안 주고 .class 파일을 주나봄
  - cmd에서 프로그램 실행하면 오류 원인 알 수 있다고?
  - 개발환경과, 실행환경의 차이 감안
    - 호스팅 보내는 환경과 맞춰줘야함
- 1. 시작
  - Spring Legacy Project
    - 패키지 필수
      - 패키지명은 기본 세자리 이상
        a.a.a
        - 보통 도메인 거꾸로
          - 관례적
        - 유니크함을 위해서?
    - 템플릿
      - Spring MVC Project

        - mvc 형태로 만들어짐
    - 외부 라이브러리를 많이 씀
        - 자동화
    - 스프링
        - 메이븐을 기본적으로 내장? 하고 있는가봄
            - pom.xml에 추가하면 자동으로 받아짐
    - webapp의 바로 밑에 있으면 접근 가능함
        - views는 컨트롤러를 통해 접근 가능
            - 그냥은 접속 못함
        - -context.xml은 설정파일
- +
    - WAS
    - static에 대한 이해
        - 이게 언제 만들어지는 거지?
- ?
    - FTP 계정?
    - 확장자 bat
        - 배치 파일
            - 정보·통신 자주 반복하여 실행할 명령어를 순차적으로 기록하여 놓은 파일. 이 파일을 실행하면 기록된 순서에 따라 명령어가 모두 실행된다.
            - batch file
            - 파일 확장자 .bat, .cmd, .btm
            - 인터넷 미디어 타입
              application/bat
              application/x-bat
              application/x-msdos-program
              text/plain
            - 포맷 종류
                - 스크립트
        - batch

- 1.
  명사 (일괄적으로 처리되는) 집단[무리]

  2.
  명사 한 회분(한 번에 만들어 내는 음식기계 등의 양)

  3.
  동사 (일괄 처리를 위해) 함께 묶다
- 배치 처리 batch處理
  정보·통신 처리할 데이터를 일정 기간 또는 일정량을 기준으로 묶어서 한꺼번에 처리하는 방식.

▼ 2. 이클립스의 dynamic web project의 구조

- 1. webContent 안의 jsp실행
- ▼ 2. WebContent/Web-INF
  - ▼ /lib
    - 외부 라이브러리 저장
  - ▼ web.xml
    - 없으면 서버걸 가져다씀. 있으면 자기걸 쓰고
    - 배치 지시자(deployment descripotr)로서 일종의 환경 설정 파일
      웹 애플리케이션에 대한 여러가지 설정을 할 때 사용

▼ 3. 톰캣 어플리케이션 구조

- ▼ webApps-manager
  - 자바에서의 하나 프로젝트 단위?
- ▼ 톰캣사용시 디렉터리의 파일목록 보여주기(개발 편의상)

- 아파치나 nginx 웹서버의 경우 해당 경로에 index 페이지 파일 설정이 안되어있으면, 그 directory 의 파일 목록을 보여주는 기능이 있다.

  tomcat 에도 있을까 하고 살펴봤더니 옵션 한곳만 바꿔주면 된다.

  conf/web.xml

  ```
  <servlet>
      <servlet-name>default</servlet-name>
      <servlet-class>
       org.apache.catalina.servlets.DefaultServlet
      </servlet-class>
      <init-param>
         <param-name>debug</param-name>
         <param-value>0</param-value>
      </init-param>
      <init-param>
         <param-name>listings</param-name>
         <param-value>false</param-value>
      </init-param>
      <load-on-startup>1</load-on-startup>
  </servlet>
  ```

  listings 의 false 를 true 로 바꿔주고,
  tomcat을 재시작해주면 파일 목록을 볼 수 있다 :)

(아마 운영 서버에서는 이렇게 쓰진 않겠지...)

- 톰캣-webapps-WEB-INF
  - lib
  - classes
    - 컴파일된 자바 클래스 파일만 있음
  - web.xml

- ex
  - package com.bit.ex01;

```
import java.util.Random;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

//스프링에서 컨트롤러 역할을 위한 어노테이션. 붙여야만 컨트롤러 역할을 할 수 있음
@Controller
public class RandomController {

	@RequestMapping(value= "/random") //액션명이라 볼 수 있음
	public String random(Model model) //model이 매개체 역할을 함.
	{
		Random r = new Random();
		int lucky = r.nextInt(45)+1;

		model.addAttribute("Lucky",lucky); //lucky는 int지만 오토박싱됨

		return "random";  //  views/random.jsp 호출
	}

}
```

- <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
  <%@ page session="false" %>
  <%@ page contentType="text/html; charset=UTF-8" %>
  <html>
  <head>
  	<title>Home</title>
  </head>
  <body>
  <h1>
  Random!
  </h1>

  RandomController에서 random메소드의 model에 Lucky 키 값으로 저장

  <P> 오늘의 행운의 숫자는 ${Lucky}</P>
  <c:forEach var="emo" begin="1" end="${Lucky}">
  😁
  </c:forEach>
  </body>
  </html>

200807 Spring

- 1. 스프링 시작 복습
  - cf
    - web.xml을 보면
      - <servlet-mapping>
        <servlet-name>appServlet</servlet-name>
        <url-pattern>/</url-pattern>
        </servlet-mapping>
        - 기본으로 /가 세팅되어 있음
      - 그래서
        - @RequestMapping(value="random")
        - 1. home으로!
        - 이런식으로 시작지점에서 슬래시가 안 들어가는 걸로 보임
  - web.xml에 인코딩값(리퀘스트에 대한?)을 설정해줄 수 있나봄
    - <filter>
      <filter-name>encodingFilter</filter-name>
      <filter-class>org.springframework.web.filter.CharacterEncodingFilter
      </filter-class>
      <init-param>
      <param-name>encoding</param-name>
      <param-value>UTF-8</param-value>
      </init-param>
      <init-param>
      <param-name>forceEncoding</param-name>
      <param-value>true</param-value>
      </init-param>
      </filter>
      <filter-mapping>
      <filter-name>encodingFilter</filter-name>
      <url-pattern>/*</url-pattern>
      </filter-mapping>
    - 이렇게 하면
      - // req.setCharacterEncoding("utf-8"); Post방식에서 이걸 해주지 않아도 한글이 깨지지 않음
  - 경로에 대해 생각해보자

- package com.bit.myspring;

  import java.io.UnsupportedEncodingException;

  import javax.servlet.http.HttpServletRequest;

  import org.springframework.stereotype.Controller;
  import org.springframework.ui.Model;
  import org.springframework.web.bind.annotation.RequestMapping;
  import org.springframework.web.bind.annotation.RequestMethod;

  @Controller
  public class FormController {

  	@RequestMapping(value= "list", method= RequestMethod.GET) // /list로 해도, list를 해도 되고 있음🤔 //webapp 바로 밑에있는 form.jsp에서 action="list"로 왔었음.
  	public String list(HttpServletRequest req, Model model) throws UnsupportedEncodingException
  	{
  		String name= req.getParameter("name");

  		model.addAttribute("name", name);
  		model.addAttribute("method", "GET");

  		return "/list"; // webapp/list.jsp //webapp 바로 밑에 있는 파일임.
  	}

  	@RequestMapping(value= "list", method= RequestMethod.POST)
  	public String list2(HttpServletRequest req, Model model) throws UnsupportedEncodingException
  	{
  //		req.setCharacterEncoding("utf-8"); Post방식에서 이걸 해주지 않아도 한글이 깨지지 않음
  		String name= req.getParameter("name");

  		model.addAttribute("name", name);
  		model.addAttribute("method", "POST");

  		return "list"; // webapp/views/list.jsp
  //		return "aa/list"; // webapp/views/aa/list.jsp
  	}

}

- WEB-INF/spring/appServlet/servlet-context.xml
  - <?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 directory -->
    <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/" />
    <beans:property name="suffix" value=".jsp" />
    </beans:bean>

    <context:component-scan base-package="com.bit.myspring" />

    </beans:beans>
  - 여기에서 보면 기본 경로를 잡아주고 있는것으로 보임

- 2. 로그인

```
package com.bit.myspring;

import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class MemberController {

	@RequestMapping(value="member/loginForm")
	public String loginForm()
	{
		return "member/loginForm";  //  member/loginForm.jsp
	}



	@RequestMapping(value="member/confirmId")
	public String confirmId(HttpServletRequest req, Model model) //값을 넘겨야 해서
Model?
	{
//		평상시 하던 방법
		String id = req.getParameter("id");
		String pwd = req.getParameter("pwd");

		model.addAttribute("id", id);
		model.addAttribute("pwd", pwd);

		return "member/confirmId"; //
	}

	@RequestMapping(value="member/confirmId1")
	public String confirmId2(@RequestParam("id") String id,
					@RequestParam("pwd") String pwd,
							Model
model)
	{
		model.addAttribute("id", id);
		model.addAttribute("pwd", pwd);

		return "member/confirmId"; // views/member/confirmId.jsp
	}

}
```

- @RequestParam("파라미터명") 타입 변수명
- 파라미터명으로 받은 걸 해당 변수에 대입함
  - @RequestParam("pwd") int pwd,
    - 바로 기본 데이터타입형 변수에도 넣어짐
    - 파싱 해주지 않아도

- 3. 회원가입

- ////////////////////////////joinForm////////////////////////////////

```
	@RequestMapping(value="member/joinForm")
	public String joinForm()
	{
		return "member/joinForm";
	}


	@RequestMapping(value="member/join")
	public String join(@RequestParam("id") String id,
					@RequestParam("pwd") String pwd,
					@RequestParam("name") String name,
					@RequestParam("email") String email,
								Model model)
	{
		MemberDto dto = new MemberDto();
		dto.setId(id);
		dto.setPwd(pwd);
		dto.setName(name);
		dto.setEmail(email);

		model.addAttribute("member", dto);

		return "member/join";
	}


	@RequestMapping(value="member/join1")
	public String join1(@ModelAttribute("member") MemberDto dto, Model model)
	{
//		자동적으로 setter호출
		model.addAttribute("member", dto);

		return "member/join";
	}



}
```

  - @ModelAttribute

- ?

-
```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>
```
  - 따로 jstl 을 의존성 추가 안 해도 사용 가능할까?
  - web.xml이 기본적으로 갖고 있던데
    - 오 그런것 같음
      - repository/javax/servlet/jstl/1.2/jstl-1.2-sources.jar
      - maven Dependencies를 보면 jstl-1.2.jar을 갖고 있음
- 컨트롤러에서 바로 webapp 밑에 있는 파일에서
- setString이랑 setNString이랑 뭐가 다르지?

- cf
  - JMX
    - JMX(Java Management eXtensions)
    - 감시 관리를 위한 도구를 제공하는 자바 API
      - 응용 프로그램(소프트웨어)/객체/장치 (프린터 등) 및 서비스 지향 네트워크
        - 응용 프로그램(소프트웨어)/객체/장치, 서비스 지향 네트워크는 MBean(Managed Bean)이라는 객체로 표현
    - JConsole, VisualVM
  - post는 바디에 달고가서 content type이랑 얽히게 되나?
    - HTTP POST 메서드는 서버로 데이터를 전송합니다. 요청 본문의 유형은 Content-Type 헤더로 나타냅니다.

      PUT과 POST의 차이는 멱등성으로, PUT은 멱등성을 가집니다. PUT은 한 번을 보내도, 여러 번을 연속으로 보내도 같은 효과를 보입니다. 즉, 부수 효과(side effect)가 없습니다.
      - 멱등성
        - 멱등성(Idempotence)이란?
          멱등성이은 동일한 동사를 두번 사용해도 리소스에는 아무런 변화가 없음을 의미한다.
          - HTTP 메소드를 예를 들자면
            - GET, PUT, DELETE는 같은 경로로 여러번 사용해도 결과가 같다
            - 하지만 POST같은 경우는 새로운 데이터가 생성되는 것이기 때문에 멱등이아니다.
        - 연산을 여러 번 적용하더라도 결과가 달라지지 않는 성질을 의미
  - 데이터베이스

- 대규모를 감안할 것
- 쿼리문을 작성할 때도 생각하기
  - ex. * 보다는 컬럼명을 명시하는게 성능이 좋음
- 인터페이스
  - 인터페이스를 생성함으로써 규격화하기
    - 통일
    - 강제화
  - 일종의 다중 상속

- 4. @PathVariable
  -
```
// member/student/aaa/91241033
@RequestMapping(value= "member/student/{studentId}/{num}")
public String student(@PathVariable String studentId,
                      @PathVariable int num,
                                          Model model)
{
    model.addAttribute("studentId", studentId);
    model.addAttribute("num", num);

    return "member/student";  //  view/member/student
}
```

  - 가변적으로 값을 받을 수 있음
  - 해당 {} 의 이름과 @PathVariable의 변수명이 같아야함
    - 아니면
      - 오류뜸
      - 400(잘못된 요청): 서버가 요청의 구문을 인식하지 못했다.
  - Spring 에서 @PathVariable 사용하여 값을 넘겨받을때 값에 . 가 포함되어 있으면 .포함하여 그뒤가 잘려서 들어온다.

    @RequestMapping(value = "/user/email/{email}", method=RequestMethod.GET)

    위와같은 형식일때 아래와 같이 바꿔주면 제대로 들어온다.

    @RequestMapping(value = "user/email/{email:.+}", method = RequestMethod.GET)
    public ModelAndView getUserByEmail(@PathVariable("email") String email) {

    출처: https://winmargo.tistory.com/202 [보리 & 마고]
  - 특수문자가 포함되어 있을때

- .+

- 5. ModelAndView
  - @RequestMapping("member/getPostForm") //value가 맨 처음 매개변수라서 value를 적어주지 않아도 되긴 하나봄.

```
public String getPostForm()
{
	return "member/getPostForm";
}

@RequestMapping(value= "member/goGet", method= RequestMethod.GET)
public String goStudent(HttpServletRequest req, Model model)
{
	String id= req.getParameter("id");
	model.addAttribute("id", id);

	return "member/goGet";
}

@RequestMapping(value="member/goPost", method= RequestMethod.POST)
public ModelAndView goStudent2(HttpServletRequest req)
{
//		ModelAndView 객체 리턴할 것
	ModelAndView mv = new ModelAndView();
	String id= req.getParameter("id");

	mv.addObject("id", id);
	mv.setViewName("member/goGet");

	return mv;
}
```

- 6. 데이터베이스 연결
  - server.xml

- <GlobalNamingResources>
  <!-- Editable user database that can also be used by
  UserDatabaseRealm to authenticate users
  -->
  <Resource auth="Container" description="User database that can be updated and saved" factory="org.apache.catalina.users.MemoryUserDatabaseFactory" name="UserDatabase" pathname="conf/tomcat-users.xml" type="org.apache.catalina.UserDatabase"/>

  <Resource auth="Container"
  driverClassName="oracle.jdbc.OracleDriver"
  maxActive="50" maxWait="1000"
  name="jdbc/Oracle11g" password="0000"
  type="javax.sql.DataSource"
  url="jdbc:oracle:thin:@localhost:1521:xe"
  username="scott"/>

  </GlobalNamingResources>
- <Context docBase="springNote" path="/springnote" reloadable="true" source="org.eclipse.jst.jee.server:springNote">
  <ResourceLink global="jdbc/Oracle11g" name="jdbc/Oracle11g" type="javax.sql.DataSource"/>
  </Context>

▼ pom.xml

- <!-- 스프링에서 JDBC 를 사용하기 위한 라이브러리 입니다. -->
  <dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-jdbc</artifactId>
  <version>${org.springframework-version}</version>
  </dependency>

  <!-- 컨넥션 풀을 위한 라이브러리 -->
  <dependency>
  <groupId>commons-dbcp</groupId>
  <artifactId>commons-dbcp</artifactId>
  <version>1.4</version>
  </dependency>

  <!-- 오라클 JDBC 드라이버 -->
  <dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc6</artifactId>
  <version>12.1.0.2</version>
  </dependency>

- <repositories>
  <repository>
  <id>oracle</id>
  <url>http://maven.jahia.org/maven2</url>
  </repository>
  </repositories>
  - 위치 상관이 있으려나. 프로퍼티즈랑 디펜던시즈 사이에 놓음
  - 오라클

- 7. 심플 노트
  - 개선의 여지가 있어보임
    - 그래도 일단 붙여둠
  - list.jsp

- <%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>

  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
	<body>

		<table>
			<tr>
				<th>아이디</th>
				<th>작성자</th>
				<th>이름</th>
			</tr>

	<c:forEach var="item" items="${arr}">
<%--			<c:url var="delPage" value="delete"> --%>
<%--			<c:param name="id" value="${item.id}" /> --%>
<%--			</c:url> --%>
<%--				<td><a href="${delPage}" class="confirmDel"
onclick="return checkDel();">${item.id}</td> --%>
			<tr>
				<td><a href="delete?id=${item.id}" class="confirmDel"
onclick="return checkDel();">${item.id}</td>
				<td>${item.content}</td>
				<td>${item.writer}</td>
			</tr>
	</c:forEach>

		</table>

	<script>

	function checkDel()
	{
		confirm("삭제하시겠습니까?");
	}

	</script>

```
        </body>
    </html>
```

- controller

- package com.bit.springnote.controller;

```
import javax.servlet.http.HttpServletRequest;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

import com.bit.springnote.command.NoteCommand;
import com.bit.springnote.command.NoteDelete;
import com.bit.springnote.command.NoteList;
import com.bit.springnote.command.NoteWrite;

@Controller
public class NoteController {

	NoteCommand command;

	@RequestMapping(value="noteForm")
	public String noteForm()
	{
		return "noteForm"; //  views/noteForm.jsp
	}

	@RequestMapping(value="write", method=RequestMethod.POST)
	public String write(HttpServletRequest req, Model model)
	{
		model.addAttribute("request", req); //모델에 리퀘스트 객체를 담아주고 있
음

		command = new NoteWrite();
		command.execute(model);

		return "redirect:list"; //
	}

	@RequestMapping(value="delete")
	public String delete(@RequestParam("id") int id, Model model)
	{

		model.addAttribute("id", id);

		command = new NoteDelete();
		command.execute(model);
```

```
		return "redirect:list";
	}

	@RequestMapping(value="list")
	public String list(Model model)
	{
		command =new NoteList();
		command.execute(model);

		return "list";
	}


}
```

- 인터페이스 만들어서 규격화
  - package com.bit.springnote.command;

    import java.util.Map;

    import org.springframework.ui.Model;

    import com.bit.springnote.dao.NoteDao;

    public class NoteDelete implements NoteCommand {

    	@Override
    	public void execute(Model model)
    	{
    		Map<String, Object> map= model.asMap();

    		int id = (Integer) map.get("id");

    		NoteDao dao= new NoteDao();
    		dao.delete(id);

    	}

    }

- package com.bit.springnote.command;

  import java.util.ArrayList;

  import org.springframework.ui.Model;

  import com.bit.springnote.dao.NoteDao;
  import com.bit.springnote.dto.NoteDto;

  public class NoteList implements NoteCommand {

      @Override
      public void execute(Model model)
      {
          NoteDao dao = new NoteDao();
          ArrayList<NoteDto> arr = dao.list();

          model.addAttribute("arr", arr);

      }

  }

  - package com.bit.springnote.command;

    import java.util.Map;

    import org.springframework.ui.Model;

    import com.bit.springnote.dao.NoteDao;

    public class NoteDelete implements NoteCommand {

        @Override
        public void execute(Model model)
        {
            Map<String, Object> map= model.asMap();

            int id = (Integer) map.get("id");

            NoteDao dao= new NoteDao();
            dao.delete(id);

        }

    }

200810 Spring

- 1. mybatis- xml에 쿼리문 작성하기
  - pom.xml

```
<repositories>
    <repository>
        <id>oracle</id>
        <url>http://maven.jahia.org/maven2</url>
    </repository>
</repositories>

<dependencies>

<!-- mybatis -->
    <dependency>
        <groupId>org.mybatis</groupId>
        <artifactId>mybatis</artifactId>
        <version>3.5.1</version>
    </dependency>

        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>2.0.1</version>
        </dependency>

        <!-- 스프링에서 JDBC 를 사용하기 위한 라이브러리 입니다. -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-jdbc</artifactId>
            <version>${org.springframework-version}</version>
        </dependency>

        <!-- 커넥션 풀을 위한 라이브러리 -->
        <dependency>
            <groupId>commons-dbcp</groupId>
            <artifactId>commons-dbcp</artifactId>
            <version>1.4</version>
        </dependency>

        <!-- 오라클 JDBC 드라이버 -->
        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc6</artifactId>
            <version>12.1.0.2</version>
        </dependency>

        <!-- lombok -->
        <dependency>
            <groupId>org.projectlombok</groupId>
```

```
        <artifactId>lombok</artifactId>
        <version>1.18.8</version>
        <scope>provided</scope>
    </dependency>
```

- webapp - WEB-INF - spring
  - appServlet
    - servlet-context.xml
      - <!-- DB연동 리소스 -->

```
<beans:bean name="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
    <beans:property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"/>
    <beans:property name="url"
value="jdbc:oracle:thin:@localhost:1521:xe"/>
    <beans:property name="username" value="scott"/>
    <beans:property name="password" value="0000"/>
</beans:bean>

<!-- 매퍼 위치 설정 -->
<beans:bean id="sqlSessionFactory"
class="org.mybatis.spring.SqlSessionFactoryBean">
    <beans:property name="dataSource" ref="dataSource"/>
    <beans:property name="mapperLocations"
value="classpath:com/bit/springNote2/dao/mapper/*.xml"/>
</beans:bean>
```

      - jsp에서의 useBeans처럼
        객체를 미리 만드는 것
        - 이곳을 통해 이미 생성되게 됨
          - 컨트롤러에서 가져다 쓸 것
        - <beans:bean id="sqlSession"
          class="org.mybatis.spring.SqlSessionTemplate">
          <beans:constructor-arg index="0" ref="sqlSessionFactory"/>
          </beans:bean>
          - 위에서 만든 sqlSessionFactory를 참조하고 있음
        - 마이바티스 활용해서 하는 것인가봄
        - value="classpath:com/bit/springNote2/dao/mapper/*.xml"/>
          - 위치를 뜻함
            - 자바 패키지형태로 디렉터리를 만들것임
              - com.bit.springNote2.dao.mapper
                - IDao.xml

- <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">

  <!-- dtd는 xml의 뼈대? -->
  <mapper namespace="com.bit.springNote2.dao.IDao">

  <select id="listDao"
  resultType="com.bit.springNote2.dto.NoteDto">
  select * from tblNote order by id desc
  </select>

  <insert id="writeDao">
  insert into tblNote(id, writer, content)
  values(tblNoteSeq.nextVal,#{param1},#{param2})
  </insert>

  <delete id="deleteDao">
  delete from tblNote where id = #{param1}
  </delete>

  </mapper>

  - 여기에서 인터페이스를 구현함
  - cf
    - dtd
      - Document Type Definition
      - XML 파일은 프로그램이 자체적으로 파서를 내장하고 있지 않은 한 생전 처음 보는 태그들이 마구 떠주는데,
      - 이럴 때 처음 부분에서 어떤 DTD를 가리키고 있는지 찾아서 실제로 어떤 녀석인지를 파악하는 데 쓴다
    - xml
      - XML(Extensible Markup Language)
      - W3C에서 개발
      - 다른 특수한 목적을 갖는 마크업 언어를 만드는 데 사용하도록 권장하는 다목적 마크업 언어이다.
    - 스프링에서의 xml파일

- web.xml
  - 설정을 위한 설정파일입니다.
    즉, 최초로 WAS가 최초로 구동될 때, 각종 설정을 정의해줍니다.
- servlet-contex.xml
  - servlet에서 보듯이 요청과 관련된 객체를 정의합니다.
- root-contex.xml
  - 뷰와 관련되지 않은?
  - 따라서 Service, Repository(DAO), DB등 비즈니스 로직과 관련된 설정을 해줍니다.

- namespace는 연결할 클래스 및 인터페이스를 지정하는 듯함
  - cf. IDao라는 이름으로 dao 패키지에 인터페이스가 있음
    - package com.bit.springNote2.dao;

      import java.util.ArrayList;

      import
      org.springframework.stereotype.Repository;

      import com.bit.springNote2.dto.NoteDto;

      public interface IDao {

      public ArrayList<NoteDto> listDao();
      public void writeDao(String writer, String
      Content);
      public void deleteDao(String id);

      }

- root-context.xml
- 컨트롤러

- package com.bit.springNote2.controller;

```
import java.io.UnsupportedEncodingException;

import javax.servlet.http.HttpServletRequest;

import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import com.bit.springNote2.dao.IDao;


@Controller
public class NoteController {

	@Autowired //sevlet-context.xml 에서 생성된 객체를 사용 : '주입' 받음
	private SqlSession sqlSession;


	@RequestMapping(value="noteForm")
	public String noteForm()
	{
		return "noteForm"; //  views/noteForm.jsp
	}


	@RequestMapping(value="write", method=RequestMethod.POST)
	public String write(HttpServletRequest req, Model model) throws
UnsupportedEncodingException
	{
		IDao dao= sqlSession.getMapper(IDao.class);
		dao.writeDao(req.getParameter("writer"),req.getParameter("content"));

		return "redirect:list";
	}

	@RequestMapping(value="delete")
	public String delete(HttpServletRequest req, Model model)
	{
		IDao dao= sqlSession.getMapper(IDao.class);
		dao.deleteDao(req.getParameter("id"));
```

```
        return "redirect:list";
    }


    @RequestMapping(value="list")
    public String list(Model model)
    {
        IDao dao= sqlSession.getMapper(IDao.class);
        model.addAttribute("arr", dao.listDao());

        return "list"; //list.jsp호출
    }



}
```

- web.xml에 인코딩 설정
  - ```
    <filter>
        <filter-name>encodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter
        </filter-class>
        <init-param>
            <param-name>encoding</param-name>
            <param-value>UTF-8</param-value>
        </init-param>
        <init-param>
            <param-name>forceEncoding</param-name>
            <param-value>true</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>encodingFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
    ```

- DI
  - 의존?
    - 객체간의 의존
      - 한 클래스가 다른 클래스의 메서드를 실행할 때
        - '의존한다'
    - 변경에 의해 영향을 받는 관계
  - 의전 대상을 구하는 방법
    ```

- 1. 의존하는 객체 직접 생성하기
- 2. DI
  - 스프링은 DI 방식 이용
- 3. 서비스 로케이터

- Dependency Injection
  - 의존 주입
    - 의존 객체를 전달받는 방식
  - 장점
    - 객체 변경의 유연함
      - 변경할 코드가 하나에 집중

- cf
  - jstl
    - prefix
      - 관습적으로 c를 씀(core의 c)
  - 캐시(cache)
    - 데이터 값을 복사해 놓는 임시 장소
    - 자주 쓰는 것을 메모리를 사용하는 캐시에 보관하면 해당 수행의 속도를 향상시킬 수 있음
  - JVM이 인식하는건 결국 class?

- ?
  - mybatis?
  - 심각: 서블릿 [appServlet]을(를) 위한 Servlet.init() 호출이 예외를 발생시켰습니다.
    java.lang.ArrayIndexOutOfBoundsException: 38366
    at org.springframework.asm.ClassReader.<init>(Unknown Source)
    at org.springframework.asm.ClassReader.<init>(Unknown Source)
    at org.springframework.asm.ClassReader.<init>(Unknown Source)
    - pom.xml에서
      - <properties>
        <java-version>1.8</java-version>
        <org.springframework-version>5.0.7.RELEASE</org.springframework-version>
        - 자바, 스프링 버전 맞춰주니 됨
    - 여러가지 원인에서 일어날 수 있는 것 같은데
      - 기본적으로는
        - 설정, 어노테이션 관련이 축인 것 같았음

- MyBaits
  - 자바 퍼시스턴스 프레임워크의 일종
    - ibatis 3.0의 포크
    - 퍼시스턴스 프레임워크
      - 데이터의 저장, 조회, 변경, 삭제를 다루는 클래스 및 설정 파일들의 집합
  - XML 서술자, 애노테이션을 사용
    - 저장 프로시저나 SQL문으로 객체들 연결
  - sql문을 xml 파일을 통해 매핑함
  - 마이바티스는 JDBC로 처리하는 상당부분의 코드와 파라미터 설정및 결과 매핑을 대신 해준다
    - 마이바티스는 데이터베이스 레코드에 원시타입과 Map 인터페이스 그리고 자바 POJO 를 설정해서 매핑하기 위해 XML과 애노테이션을 사용할 수 있다.
- 2. jdbctemplate 사용
  - 1. web.xml
    - 한글처리
  - 2. pom.xml
    - 라이브러리 등 의존 추가
  - 3. servlet-context.xml
    - <beans:bean name="dataSource"
      class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <beans:property name="driverClassName"
      value="oracle.jdbc.driver.OracleDriver"/>
      <beans:property name="url"
      value="jdbc:oracle:thin:@localhost:1521:xe"/>
      <beans:property name="username" value="scott"/>
      <beans:property name="password" value="0000"/>
      </beans:bean>

      <beans:bean name="template"
      class="org.springframework.jdbc.core.JdbcTemplate">
      <beans:property name="dataSource" ref="dataSource">
      </beans:property>
      </beans:bean>

      <beans:bean name="dao" class="com.bit.springNote3.dao.NoteDao">
      <beans:property name="template" ref="template"></beans:property>
      </beans:bean>
  - 4.dto준비

- package com.bit.springNote3.dto;

  import lombok.AllArgsConstructor;
  import lombok.Getter;
  import lombok.Setter;

  @Getter @Setter @AllArgsConstructor
  public class NoteDto {

  	private int id;
  	private String writer;
  	private String content;

  	public NoteDto() {};

  }

- 5. dao작성

```
package com.bit.springNote3.dao;


import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.ArrayList;

import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.PreparedStatementCreator;
import org.springframework.jdbc.core.PreparedStatementSetter;

import com.bit.springNote3.dto.NoteDto;

public class NoteDao {

//servlet-context.xml에서 객체 생성해서 주입
	private JdbcTemplate template;

	public void setTemplate(JdbcTemplate template)
	{
		this.template = template;
	}


//write
public void write(final String writer, final String content)
{
	template.update(new PreparedStatementCreator() {

		@Override
		public PreparedStatement createPreparedStatement(Connection con)
throws SQLException {

			String sql="insert into tblnote(id, writer, content)
values(tblNoteSeq.nextval, ?, ?)";

			PreparedStatement ps = con.prepareStatement(sql);

			ps.setString(1, writer);
			ps.setString(2, content);

			return ps;
		}
	});
```

```
}


//list
public ArrayList<NoteDto> list()
{
	String sql="select * from tblNote order by id desc";
	return (ArrayList<NoteDto>)template.query(sql, new
BeanPropertyRowMapper<NoteDto>(NoteDto.class));
}


//delete
public void delete(final int id)
{
	String sql= "delete from tblNote where id=?";
	template.update(sql, new PreparedStatementSetter() {

		@Override
		public void setValues(PreparedStatement ps) throws SQLException {
			ps.setInt(1, id);

		}
	});
}


}
```

- RowMapper : 쿼리 결과를 객체로 변환
  RowMapper는 JDBC의 인터페이스인 ResultSet에서 원하는 객체로 타입을 변환하는 역할을 합니다. 기본적인 전략을 구현한 클래스는 Spring JDBC에서 제공을 합니다.

  BeanPropertyRowMapper
  DB의 컬럼명과 bean 객체의 속성명이 일치하다면 BeanPropertyRowMapper를 이용하여 자동으로 객체변환을 할 수 있습니다. DB 컬럼명이 'snake_case'로 되어 있어도 'camelCase'로 선언된 클래스의 필드로 매핑이 됩니다.
- 	return (ArrayList<NoteDto>)template.query(sql, new BeanPropertyRowMapper<NoteDto>(NoteDto.class));
  - 오류가 났었음
    - 왜?
      - dto에 기본생성자가 없었기 때문에

        - 기본생성자를 이용해 객체를 생성하는 게 아닐가 싶다.
- 주의
    - 낯선 오류를 맞닥뜨렸을 때, 쫄지 말고 주변을 잘 살필 것
        - 예를 들어 기본생성자가 없다는 힌트도, 오류 메세지에 잘 나타나 있었음