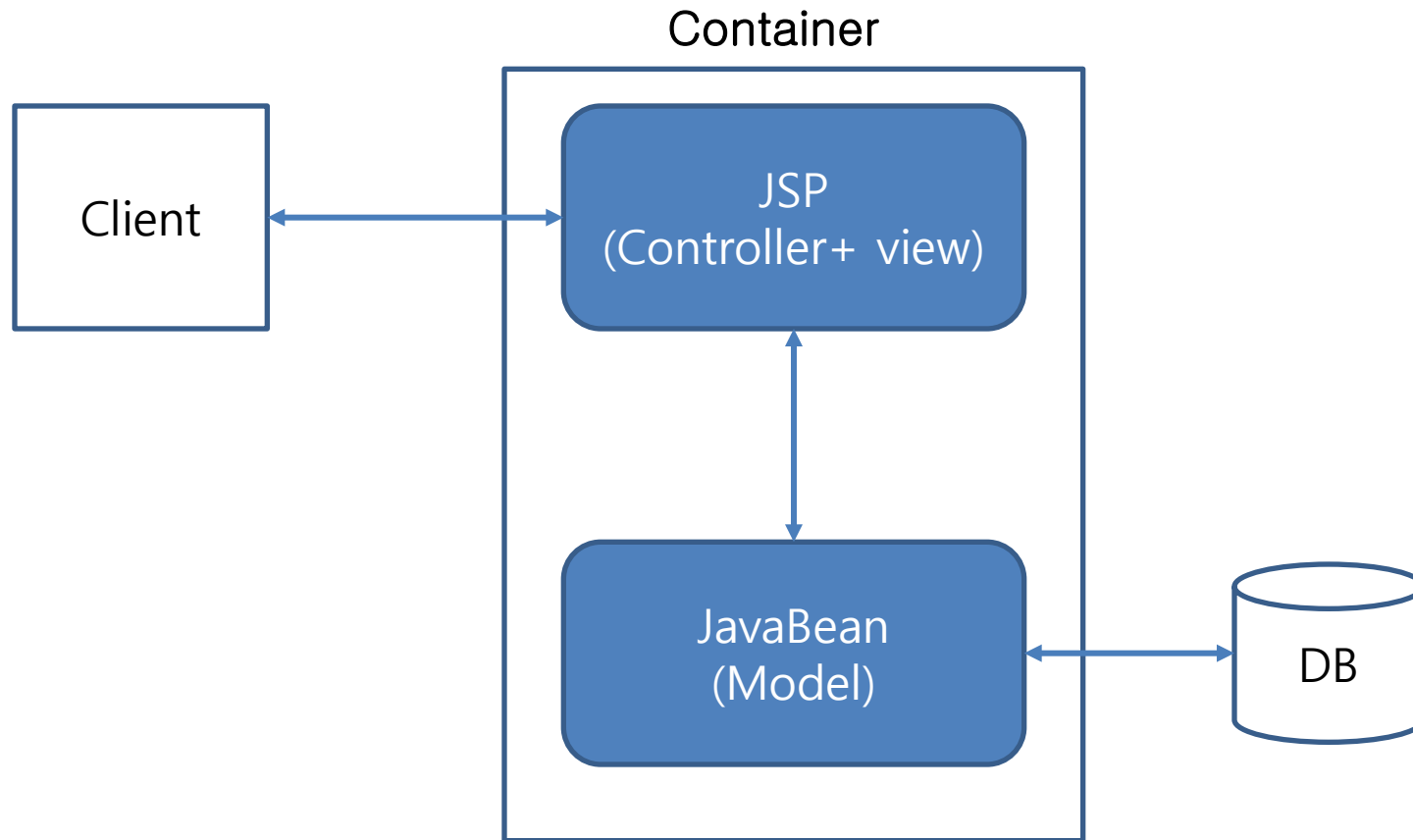


Guestbook

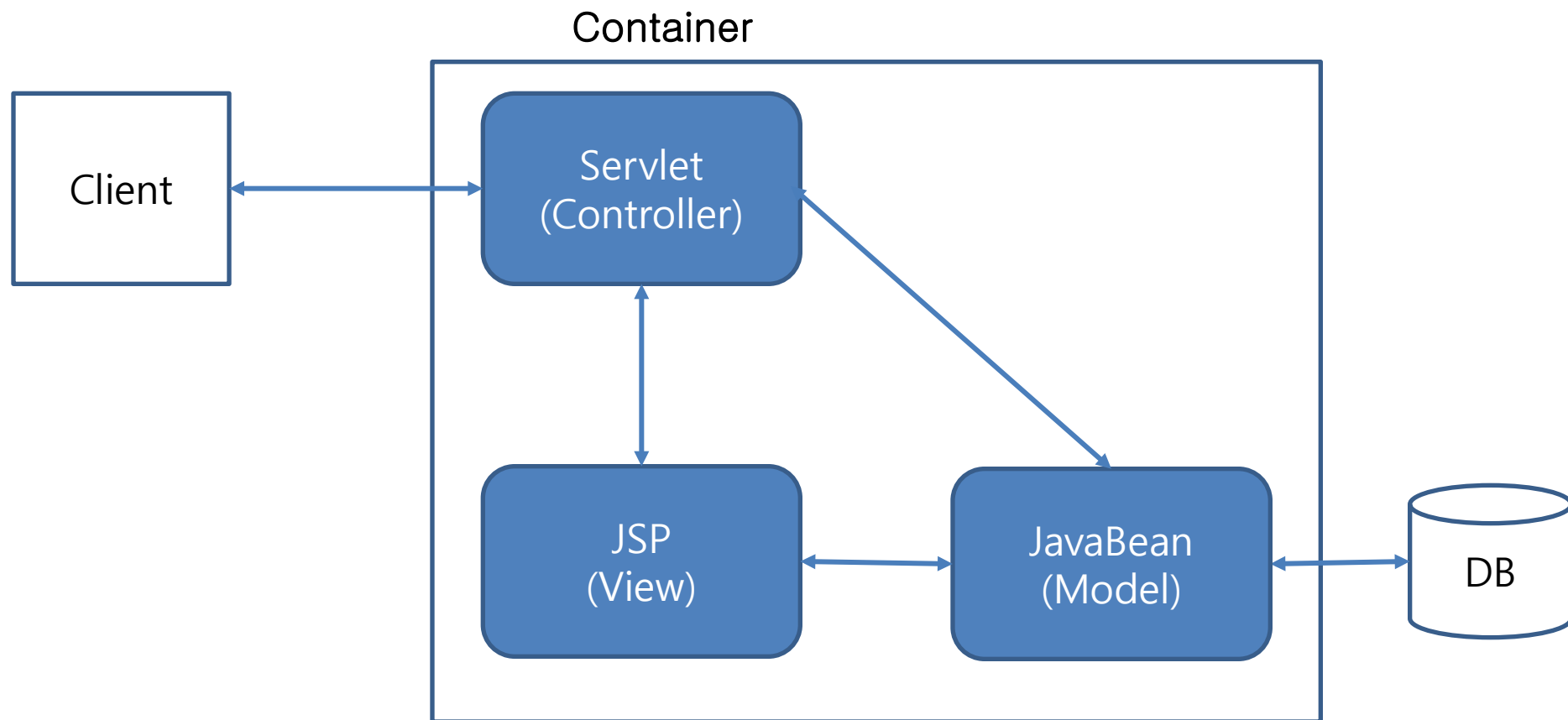
Spring작성하기

1. Model1 게시판
2. Model2 게시판
3. MVC 프레임워크 게시판
4. Spring 설치

- 모델 1 게시판 구조



- Model1의 한계 → Model2의 등장(MVC)
- Controller의 역할이 중요!!!



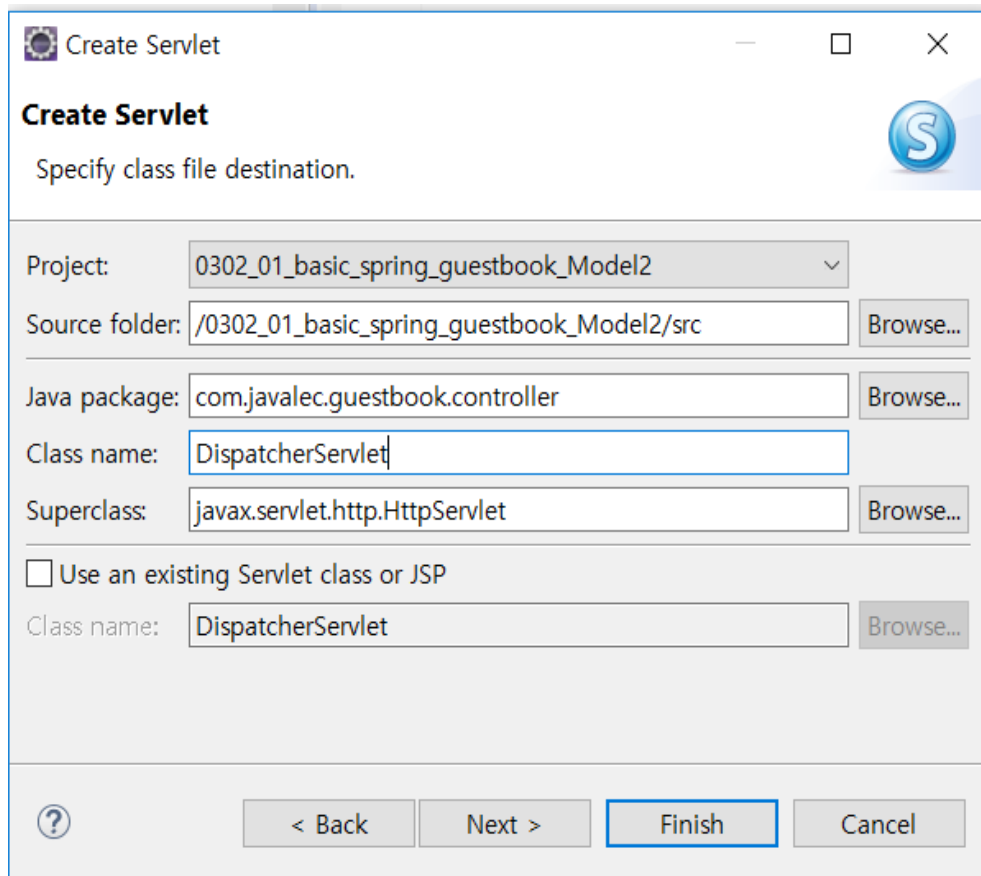
Guestbook

step 1

1.

- Model2의 기본형태인 서블릿에서의 처리를 구현한다.

1) DispatcherServlet 생성



Create Servlet

Specify class file destination.

Project: 0302_01_basic_spring_guestbook_Model2

Source folder: /0302_01_basic_spring_guestbook_Model2/src

Java package: com.javalec.guestbook.controller

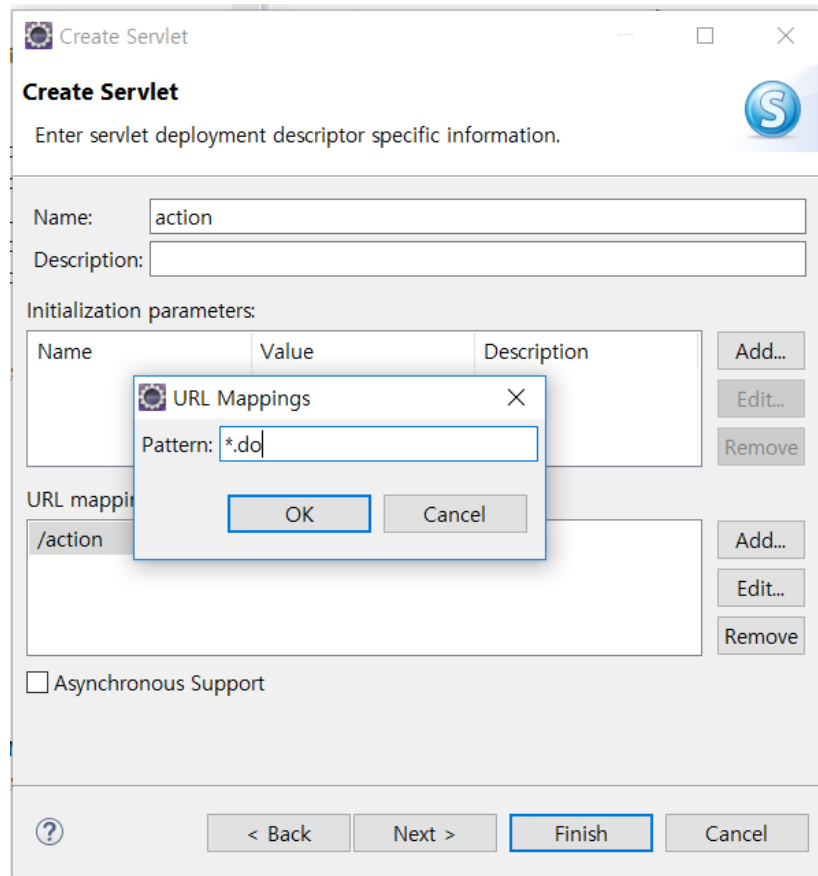
Class name: DispatcherServlet

Superclass: javax.servlet.http.HttpServlet

☐ Use an existing Servlet class or JSP

Class name: DispatcherServlet

< Back Next > Finish Cancel



Create Servlet

Enter servlet deployment descriptor specific information.

Name: action

Description:

Initialization parameters:

Name	Value	Description
------	-------	-------------

URL mappings

Pattern: *.do

OK Cancel

URL mapping

Name	Value	Description
/action		

☐ Asynchronous Support

< Back Next > Finish Cancel

2) DispatcherServlet 구현

```
private void process(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    // TODO Auto-generated method stub
    // 1. 클라이언트의 요청 path 정보를 추출한다.
    String uri = request.getRequestURI();
    String path = uri.substring(uri.lastIndexOf("/"));
    System.out.println(path);

    // 2. 클라이언트의 요청 path에 따라 적절히 분기처리 한다.
    if (path.equals("/getguestbooklist.do")) {
        System.out.println("게스트북 리스트 조회 처리");

    }else if(path.equals("/insert.do")) {

        System.out.println("입력 처리");
    }else if(path.equals("/delete.do")) {

        System.out.println("삭제 처리");
    }else if(path.equals("/deleteform.do")) {

        System.out.println("삭제폼 이동처리");
    }
}
```

3) DispatcherServlet의 코드를 완성한다.

```
// 2. 클라이언트의 요청 path에 따라 적절히 분기처리 한다.
if (path.equals("/getguestbooklist.do")) {
    System.out.println("리스트 조회");

    //1-2. 키워드 조회
    String keyword = WebUtil.checkNotNullParam( request.getParameter( "kwd" ), "" );

    System.out.println("keyword : " + keyword);
    //2. dao 객체 생성
    GuestbookDao dao = new GuestbookDao();
    //3. 리스트 조회
    List<GuestbookVo> list = dao.getList();

    System.out.println("list : " + list.size());
    //4. 세팅
    request.setAttribute( "list", list );

    RequestDispatcher rd =
        request.getRequestDispatcher( "index.jsp" );
    rd.forward( request, response );
}
```

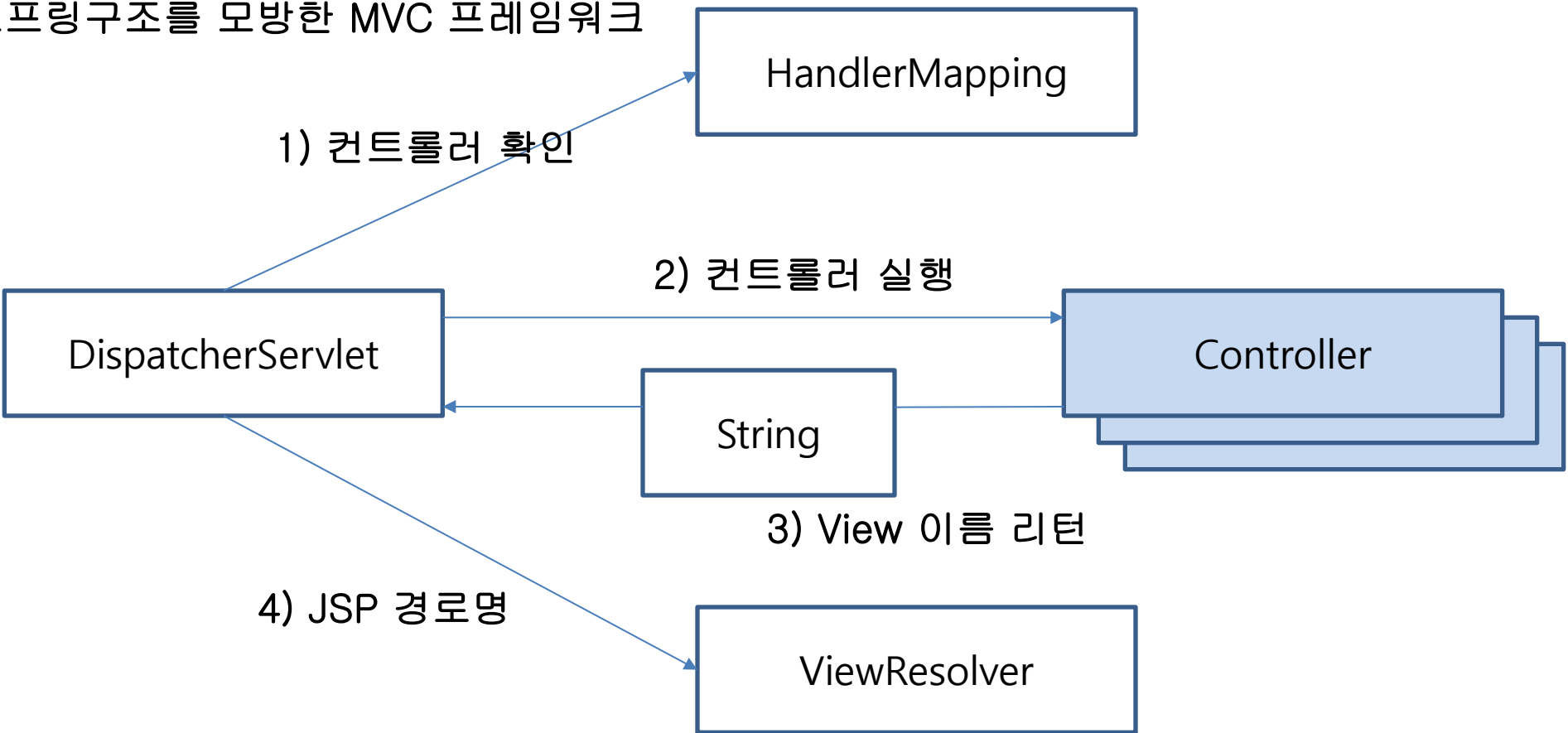
2. MVC 구현 (Servlet + Bean)

- 1. MVCW 구조
- 2. MVC 구현

- Model2 게시판 구조의 단점
 - 하나의 Controller에 다수의 기이 집중적으로 구현됨.
 - 수많은 분기로 인한 개발/유지 보수가 어려워짐.
- ➔ 하나의 서블릿(DispatcherServlet)에서 요청을 받아 해당 작업에 맞는 Controller를 제공해주는 것이 좋음.

1. MVC 구조

스프링구조를 모방한 MVC 프레임워크



- DispatcherServlet
 - 클라이언트의 요청을 처리하기 위한 서블릿
- HandlerMapping
 - 클라이언트 요청에 따른 Controller 매핑
- Controller
 - 실질적인 클라이언트의 요청 처리
- ViewResolver
 - Controller가 리턴한 View 이름을 통해 jsp 경로 생성 처리

Web.xml

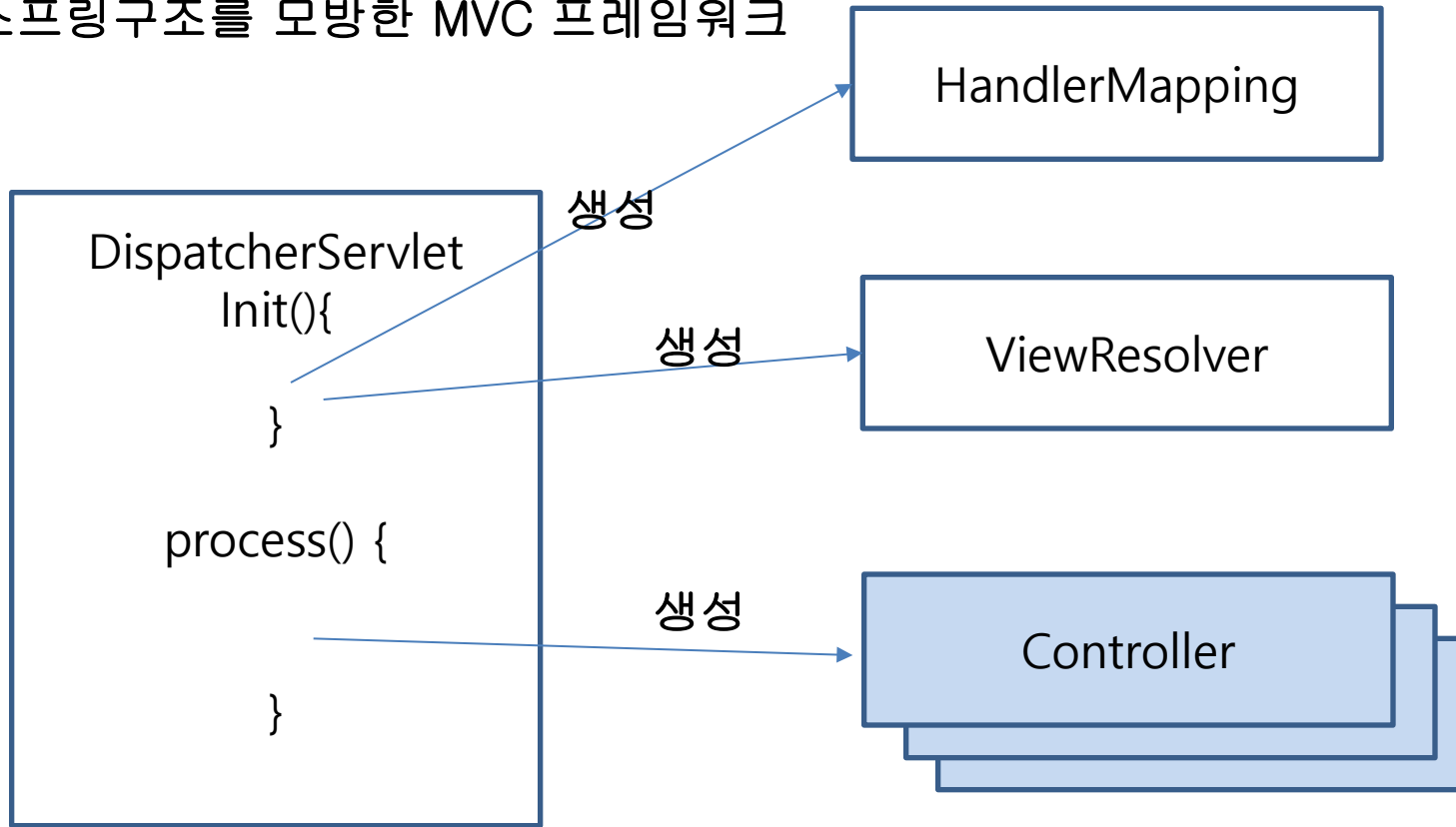
```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>
    com.javalec.guestbook.controller.DispatcherServlet
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
<welcome-file-list>
  <welcome-file>hello.jsp</welcome-file>
</welcome-file-list>
```

1) DispatcherServlet

```
public class DispatcherServlet extends HttpServlet {  
    private static final long serialVersionUID = 1L;  
  
    private HandlerMapping handleMapping ;  
    private ViewResolver viewResolver ;  
  
    /* handlerMapping ViewResolver 객체를 init() 메소드에서 생성...  
    *  
    */  
    public void init() throws ServletException{  
        handleMapping = new HandlerMapping() ;  
        viewResolver = new ViewResolver() ;  
        viewResolver.setPrefix("./");  
        viewResolver.setSuffix(".jsp");  
    }  
  
    :
```

1. MVC 구조

스프링구조를 모방한 MVC 프레임워크



1) DispatcherServlet

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// TODO Auto-generated method stub
process(request, response);
}

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
// TODO Auto-generated method stub
request.setCharacterEncoding("UTF-8");
process(request, response);
}
```

1) DispatcherServlet

```
private void process(HttpServletRequest request,
HttpServletRequest response) throws ServletException,
IOException {
// 1. 클라이언트의 요청 path 정보를 추출한다.
String uri = request.getRequestURI();
String path = uri.substring(uri.lastIndexOf("/"));
System.out.println(path);
//2. HandlerMapping을 통해 Path에 해당하는 Controller 를
검색함!!
Controller ctrl = handleMapping.getController(path) ;
//3. 검색된 controller 수행
String viewName = ctrl.handleRequest(request, response) ;
```


1) DispatcherServlet

```
//4. ViewResolver를 통해 viewname 에 해당하는 화면을 검색
String view = null ;
if(!viewName.contains(".do")) {
//액션을 취하기 위한 것이면....do 가 없으면.... jsp 붙여서 보내고...
view = viewResolver.getView(viewName);
//5. 검색된 화면으로 이동
WebUtil.forward(request, response, view);
else {
//do 가 있으면 처리 요청으로 보낸다..
view = viewName ;
response.sendRedirect(view);
}
```

- 2) HandlerMapping

```
public class HandlerMapping {  
    private Map<String, Controller> mappings ;  
  
    public HandlerMapping() {  
        mappings = new HashMap<String, Controller>();  
        mappings.put("/insert.do", new InsertGuestBookController());  
        mappings.put("/getguestbooklist.do", new GetGuestBookListController());  
        mappings.put("/deleteform.do", new DeleteFormController());  
        mappings.put("/delete.do", new DeleteController());  
    }  
  
    public Controller getController(String path) {  
        return mappings.get(path) ;  
    }  
}
```

- 3) Controller

```
public interface Controller {  
    String handleRequest(HttpServletRequest request, HttpServletResponse response) ;  
}
```

3-1) deleteController.java

```
public String handleRequest(HttpServletRequest request, HttpServletResponse response) {  
    // TODO Auto-generated method stub  
    String no = request.getParameter( "no" );  
    String password = request.getParameter( "password" );  
    GuestbookVo vo = new GuestbookVo();  
    vo.setNo(Long.parseLong( no ));  
    vo.setPassword(password);  
    GuestbookDao dao = new GuestbookDao();  
    dao.delete(vo);  
    return "getguestbooklist.do" ;}
```

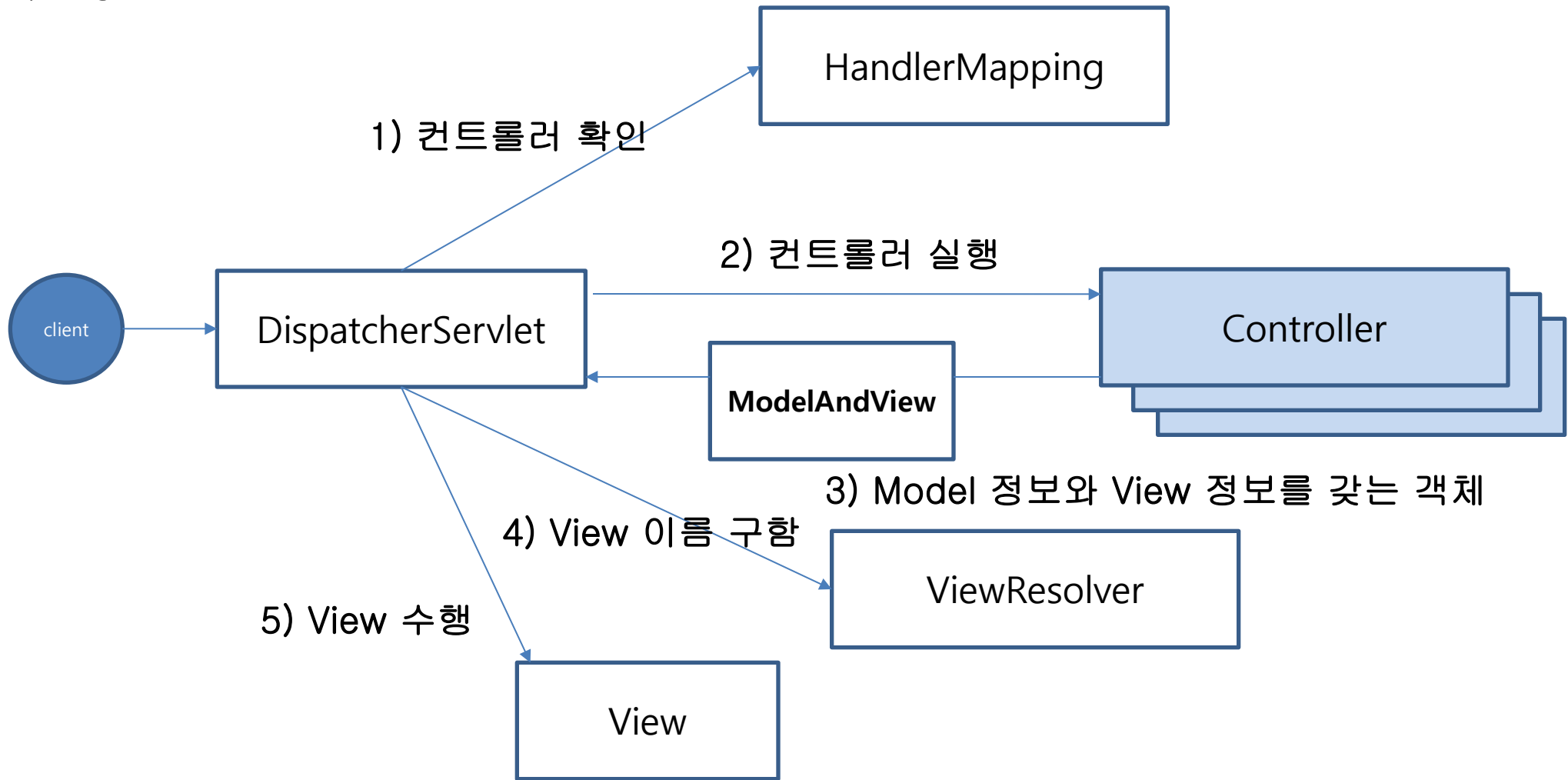
- 4) ViewResolver

```
public class ViewResolver {  
    public String prefix;  
    public String suffix;  
  
    public void setPrefix(String prefix) {  
        this.prefix = prefix;  
    }  
  
    public void setSuffix(String suffix) {  
        this.suffix = suffix;  
    }  
  
    public String getView(String viewName) {  
        return prefix + viewName + suffix;  
    }  
}
```

3. MVC 구현 (Spring)

- 1. MVCW 구조
- 2. MVC 구현(viewResolver 없는 경우)
- 3. MVC 구현(WEB-INF 디렉토리 접근)
- 4. Annotation 기반의 구현

Spring MVC



- web.xml 설정
 - 클라이언트의 요청을 처리하기 위한 DispatcherServlet 을 등록한다.
- (스프링 프레임워크에서 제공)

```
<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>

<servlet-mapping>
<servlet-name>appServlet</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
```

- DispatcherServlet에서 클라이언트 요청에 필요한 객체 (HandlerMapping, Controller, ViewResolver 객체)들은 어떻게 생성하나 ??
 - ➔ 스프링 컨테이너에 의해 구동 됨
 - ➔ 어떻게 ??
 - ➔ 스프링 설정파일(servlet-context.xml)을 로딩하여 생성
즉... servlet-context.xml에 HandlerMapping, Controller,
ViewResolver 객체가 등록됨!!
 - ➔ 내부적으로 DispatcherServlet의 init() 메소드에서 servlet-
context.xml 파일을 로딩하여 해당 객체들을 생성함!!

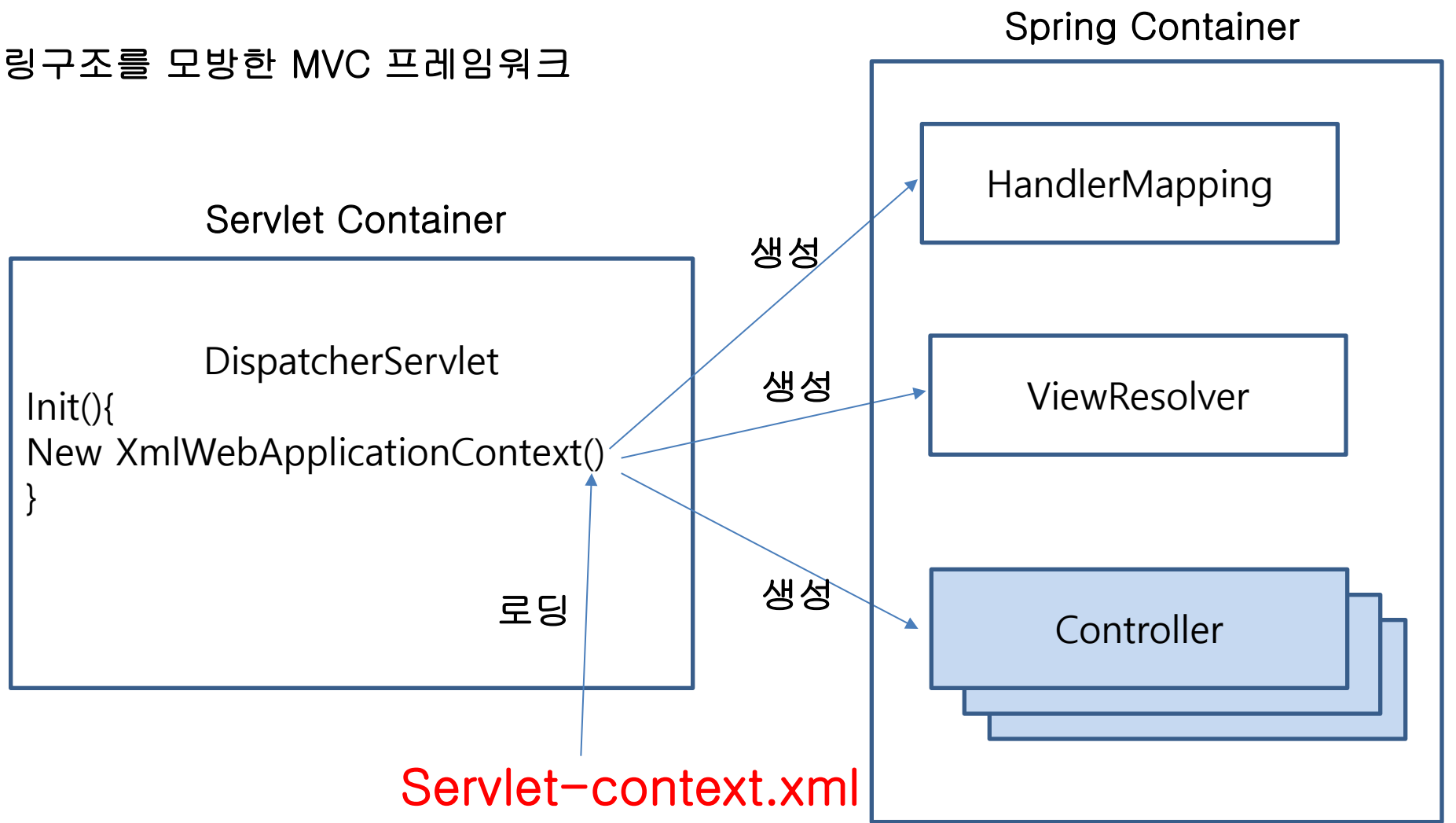
- web.xml 설정
 - 클라이언트의 요청을 처리하기 위한 DispatcherServlet 을 등록한다.

```
<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>

<servlet-mapping>
<servlet-name>appServlet</servlet-name>
<url-pattern>*.do</url-pattern>
</servlet-mapping>
```

1. MVC 구조

스프링구조를 모방한 MVC 프레임워크



1) Controller

1-1) GetGuestBookListController

```
public class GetGuestBookListController implements Controller {  
    @Override  
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response)  
    {  
        // TODO Auto-generated method stub  
        //1. dao 생성  
        GuestbookDao dao = new GuestbookDao();  
        //2. 페이징을 위한 기본 데이터 계산  
        //3. 리스트 가져오기  
        List<GuestbookVo> list = dao.getList();  
        ModelAndView mav = new ModelAndView() ;  
        mav.addObject("list", list) ;  
        mav.setViewName("index.jsp");  
        return mav;  
    }  
}
```

1) Controller

<bean> 을 통한 controller 등록 시 다음을 반드시 준수해야 한다.

- a. org.springframework.web.servlet.mvc.Controller 인터페이스 구현
- b. Controller 인터페이스의 `handleRequest(..)` 메소드 재정의!!

➔ 스프링의 `DispatcherServlet` 클래스에서 내부적으로 `handleRequest(..)` 메소드를 호출하여 처리함!!

1) Controller

```
package org.springframework.web.servlet.mvc;  
  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import org.springframework.web.servlet.ModelAndView;  
  
public interface Controller {  
    ModelAndView handleRequest(HttpServletRequest  
request, HttpServletResponse response) throws Exception;  
  
}
```

1) Controller

1-2) InsertGuestBookController

```
public class InsertGuestBookController implements Controller {  
  
    @Override  
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) {  
        // TODO Auto-generated method stub  
        String name = request.getParameter( "name" );  
        String password = request.getParameter( "pass" );  
        String content = request.getParameter( "content" );  
  
        GuestbookVo vo = new GuestbookVo();  
        vo.setName(name);  
        vo.setContent(content);  
        vo.setPassword(password);  
  
        GuestbookDao dao = new GuestbookDao();  
        dao.insert(vo);  
  
        // 3. 화면 네비게이션  
        ModelAndView mav = new ModelAndView() ;  
        mav.setViewName("getguestbooklist.do");  
        return mav ;  
    }  
}
```

* 참고.

ModelAndView 객체

- 조회 등의 처리 결과(Model)과 보여줄 뷰에 대한 정보(View)를 저장하고 있는 객체.
 - DispatcherServlet에서는 해당 ModelAndView객체의 Model에서 꺼내서 HttpServletRequest에 저장하고, View에 저장된 JSP 파일 이름의 주소로 Forwarding 처리.
- ➔ JSP 파일에서는 HttpServletRequest에서 정보를 추출할 수 있음.!!

• 2) HandlerMapping 등록

```
<beans:bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <beans:property name="mappings">
    <beans:props>
      <beans:prop key="/insert.do">insert</beans:prop>
      <beans:prop key="/getguestbooklist.do">getguestbooklist</beans:prop>
      <beans:prop key="/deleteform.do">deleteform</beans:prop>
      <beans:prop key="/delete.do">delete</beans:prop>
    </beans:props>
  </beans:property>
</beans:bean>
```

<!-- Controller 등록 -->

```
<beans:bean id="insert" class="com.javalec.guestbook.controller.InsertGuestBookController"/>
<beans:bean id="getguestbooklist" class="com.javalec.guestbook.controller.GetGuestBookListController"/>
<beans:bean id="deleteform" class="com.javalec.guestbook.controller.DeleteFormController"/>
<beans:bean id="delete" class="com.javalec.guestbook.controller.DeleteController"/>
```


- ViewResolver
 - 클라이언트로 부터의 직접적인 JSP 호출을 방지 목적.
 - 특정 디렉토리(WEB-INF)아래의 JSP 파일 매핑 호출 처리 수행.
- *servlet-context.xml에 추가!!

```
<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>
```

- 1) InsertGuestBookController

```
public class InsertGuestBookController implements Controller {

    @Override
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) {
        // TODO Auto-generated method stub
        String name = request.getParameter( "name" );
        String password = request.getParameter( "pass" );
        String content = request.getParameter( "content" );

        GuestbookVo vo = new GuestbookVo();
        vo.setName(name);
        vo.setContent(content);
        vo.setPassword(password);

        GuestbookDao dao = new GuestbookDao();
        dao.insert(vo);

        // 3. 화면 네비게이션

        ModelAndView mav = new ModelAndView() ;
        mav.setViewName("getguestbooklist.do");
        return mav ;
    }
}
```

- 1) InsertGuestBookController
- → 404 오류 발생... ??

해결 방법은 ??

HTTP Status 404 – Not Found

Type Status Report

Message /guestbook_vr/WEB-INF/views/getguestbooklist.do.jsp

Description The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.

Apache Tomcat/8.5.32

→ `mav.setViewName("redirect:getguestbooklist.do");`

: `redirect`를 붙여줌 → `ViewResolver`가 있어도 이를 무시하고 처리한다...

기준 : `*.do`, `WEB-INF` 밖의 `jsp` : `redirect` 붙여줌.

`WEB-INF` 안의 `JSP` : `redirect` 없이 `JSP` 이름만으로 사용.

1) Controller

a. <context:component-scan /> 등록.

: <bean >엘리먼트 대신 객체를 생성하기 위함!

```
<context:component-scan base-package="com.javalec.board.controller" />
```

b. @Controller 어노테이션 사용

```
@Controller  
public class GuestBookController {  
    :
```

- 2) RequestMapping 적용
 - 클라이언트의 주소 매핑처리 수행.
 - HandlerMapping 사용의 어노테이션 기반 대체 방안.

```
<beans:bean  
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  
<beans:property name="mappings">  
<beans:props>  
<beans:prop key="/insert.do">insert</beans:prop>  
</beans:props>  
</beans:property>  
</beans:bean>  
<!-- Controller 등록 -->  
<beans:bean id="insert"  
class="com.javalec.guestbook.controller.InsertGuestBookController"/>
```

@RequestMapping(/deleteform.do

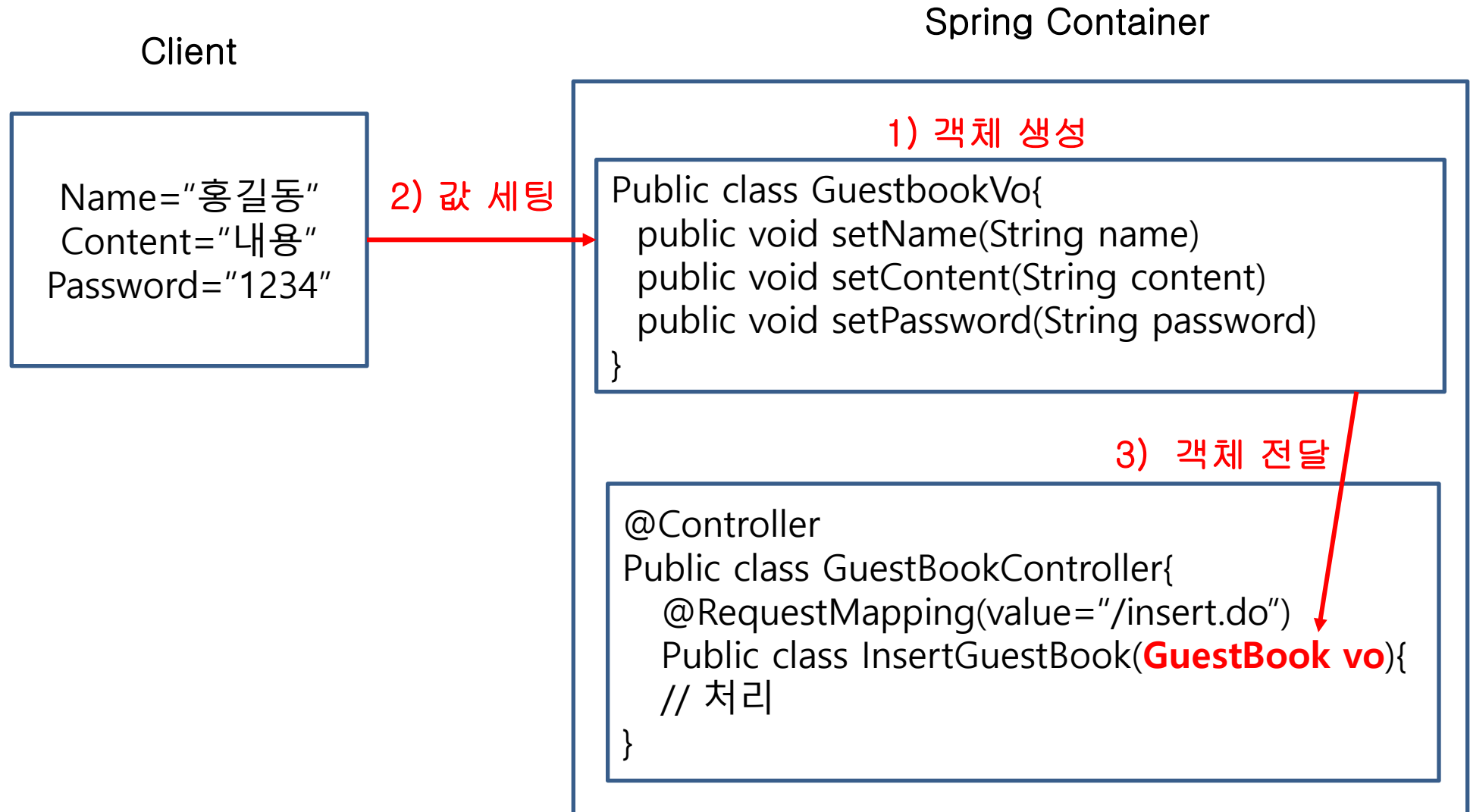
- 클라이언트의 요청 처리 방식
 - 기존 방식

```
public class InsertGuestBookController implements Controller {  
  
    @Override  
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) {  
        // TODO Auto-generated method stub  
        String name = request.getParameter( "name" );  
        String password = request.getParameter( "pass" );  
        String content = request.getParameter( "content" );  
  
        GuestbookVo vo = new GuestbookVo();  
        vo.setName(name);  
        vo.setContent(content);  
        vo.setPassword(password);  
        GuestbookDao dao = new GuestbookDao();  
        dao.insert(vo);  
    }  
}
```

고객의 요청을 HttpServletRequest에 담아 서블릿의 서비스 메소드에 전달한다..

- 클라이언트 요청 처리 방식의 변경
 - Controller에서 클라이언트 요청에 의해 호출되는 함수에 인자로 선언 시 컨테이너가 해당 객체를 만들어 세팅작업을 수행한다.

```
// 컨트롤러의 요청 시 수행될 메소드 선언!!  
@RequestMapping(value="/insert.do")  
public String insertGuestBook(GuestbookVo vo, GuestbookDao dao) {  
    System.out.println("글 등록 처리");  
    dao.insert(vo);  
    return "getguestbooklist.do" ;  
}
```



3) RequestMapping의 메소드 방식에 따른 처리

```
@RequestMapping(value="/deleteform.do", method=RequestMethod.GET)
public String deleteForm() {
    System.out.println("글 삭제 폼 이동 처리");
    return "deleteform.jsp" ;
}

@RequestMapping(value="/delete.do", method=RequestMethod.POST)
public String deleteGuestBook(GuestbookVo vo, GuestbookDao dao) {
    System.out.println("글 삭제 처리");

    dao.delete(vo);
    return "getguestbooklist.do" ;
}
```

4) 화면에 디폴트 값을 보여주기..

GuestBookController.java

```
@RequestMapping(value="/deleteform.do", method=RequestMethod.GET)
//public String deleteForm() {
public String deleteForm(GuestbookVo vo) { // 화면에 디폴트 세팅을 위한 생성
    System.out.println("글 삭제 폼 이동 처리");
    // 비밀번호 디폴트값 세팅 작업 !!!
    vo.setPassword("1234");
    return "deleteform.jsp" ;
}
```

deleteform.jsp

```
<td>비밀번호</td>
<td><input type="password" name="password" value="${guestbookVo.password }"></td>
<td><input type="submit" value="확인"></td>
<td><a href="getguestbooklist.do">메인으로 돌아가기</a></td>
```

5) @ModelAttribute 사용.(파라미터 설정에 적용 시)

- JSP 화면에서 comman객체(VO객체)를 사용 시 이름 설정 방법

GuestBookController.java

```
public String deleteForm(@ModelAttribute("book") GuestbookVo vo) { // 화면에 디폴트 세팅을 위한 생성
    System.out.println("글 삭제 폼 이동 처리");
    // 비밀번호 디폴트값 세팅 작업 !!!
    vo.setPassword("1234");
    return "deleteform.jsp" ;
}
```

deleteform.jsp

```
<td>비밀번호</td>
<!-- @ModelAttribute 사용시 -->
<td><input type="password" name="password" value="${book.password }"></td>
```

5) @ModelAttribute 사용.(메소드 위에서 선언 시)

- View 단에서 사용할 데이터를 설정하기 위한 용도로 사용.
 - @ModelAttribute 로 선언된 메소드는 @RequestMapping으로 선언된 메소드보다 먼저 자동 실행.
 - @RequestMapping에 의해 수행된 결과인 Model에 자동 결과 삽입되어 View단으로 전달...
- 단... 모든 요청 처리 전에 무조건 먼저 수행됨!!

6) Login 처리 방법은 ??

– Controller 클래스에서 어떻게 로그인을 구현할 수 있을까??

➔ HttpServletRequest를 사용하지 않는데 어떻게 HttpSession

객체를 이용하여 로그인 처리를 할 수 있을까 ??

정답 : 로그인 처리가 필요한 메소드에서 HttpSession 객체를

파라미터로 전달한다.➔ 스프링 컨테이너가 해당 세션 생성해 줌..

7) Controller의 리턴타입은 ??

a) ModelAndView 객체 리턴

: 처리 결과와 리턴할 JSP 정보를 ModelAndView 객체에 저장 후

리턴

```
public class GetBoardListController implements Controller {  
  
    @Override  
    public ModelAndView handleRequest(HttpServletRequest request, HttpServletResponse response) {  
        System.out.println("글 목록 검색 처리");  
        // 1. 사용자 입력 정보 추출(검색 기능은 나중에 구현)  
        // 2. DB 연동 처리  
        BoardVO vo = new BoardVO();  
        BoardDAO boardDAO = new BoardDAO();  
        List<BoardVO> boardList = boardDAO.getBoardList();  
        // 3. 검색 결과를 세션에 저장하고 목록 화면을 리턴한다.  
        ModelAndView mav = new ModelAndView() ;  
        mav.addObject("boardList", boardList) ;  
        mav.setViewName("getBoardList");  
        return mav;  
    }  
}
```

b) String 객체 리턴

: 리턴할 JSP 정보를 String 타입으로 리턴, 처리한 결과는 Model 객체를 이용하여 처리.

```
@RequestMapping("/getguestbooklist.do")  
public String getBoardList(GuestbookDao dao, Model model) {  
    System.out.println("글 목록 검색 처리");  
    model.addAttribute("list", dao.getList());  
    return "index.jsp";  
}
```

➔ 대부분 ModelAndView 타입을 리턴하기 보다는 String 타입을 리턴처리하는 것을 선호함!!

8) @RequestParam 적용

-클라이언트의 요청 중 Command객체(VO)에 없는 파라미터 정보를 추출하고 싶은 경우 적용.

```
<form action="searchkeywordlist.do" method="post">
<table border=1 width=500>
  <tr>
    <td>검색</td><td><input type="text" name="searchKeyword" value='${ searchKeyword}'></td>
  </tr>
  <tr>
    <td colspan=2 align=right><input type="submit" VALUE=" 확인 "></td>
  </tr>
</table>
</form>
```


4. Spring MVC 구조

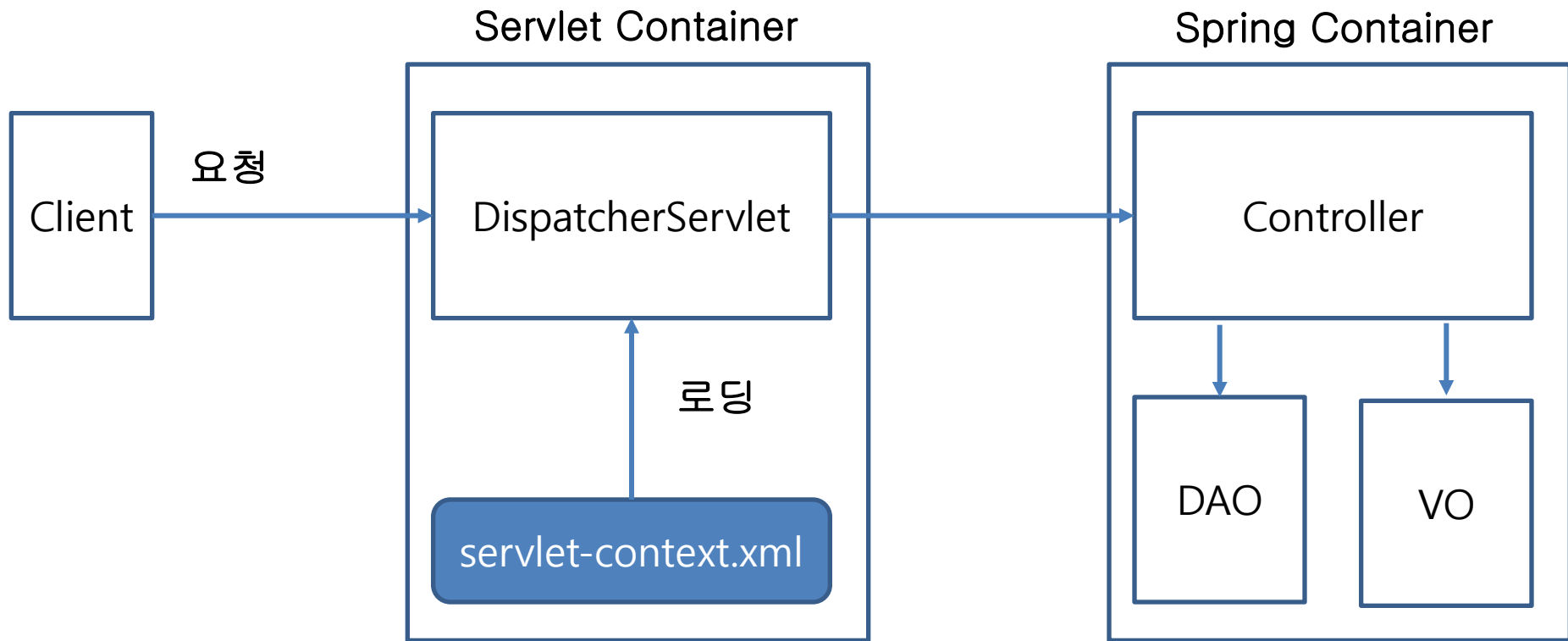
```
@RequestMapping("/searchkeywordlist.do")
public String getBoardList(GuestbookDao dao, Model model,
    @RequestParam(value="searchKeyword", defaultValue="", required=false) String keyword) {
    System.out.println("글 키워드 검색 목록 검색 처리 키워드 : " + keyword);
    model.addAttribute("list", dao.getKeywordList(keyword));
    model.addAttribute("searchKeyword", keyword);
    return "index.jsp";
}
```

- @RequestParam을 통해 클라이언트의 요청 파라미터를 받아 들인다!!
 - defaultValue : 디폴트 값 설정
 - required : 파라미터 생략 여부.
- ➔ @RequestParam을 쓰기 싫으면?? VO에 해당 필드를 추가하라!!!

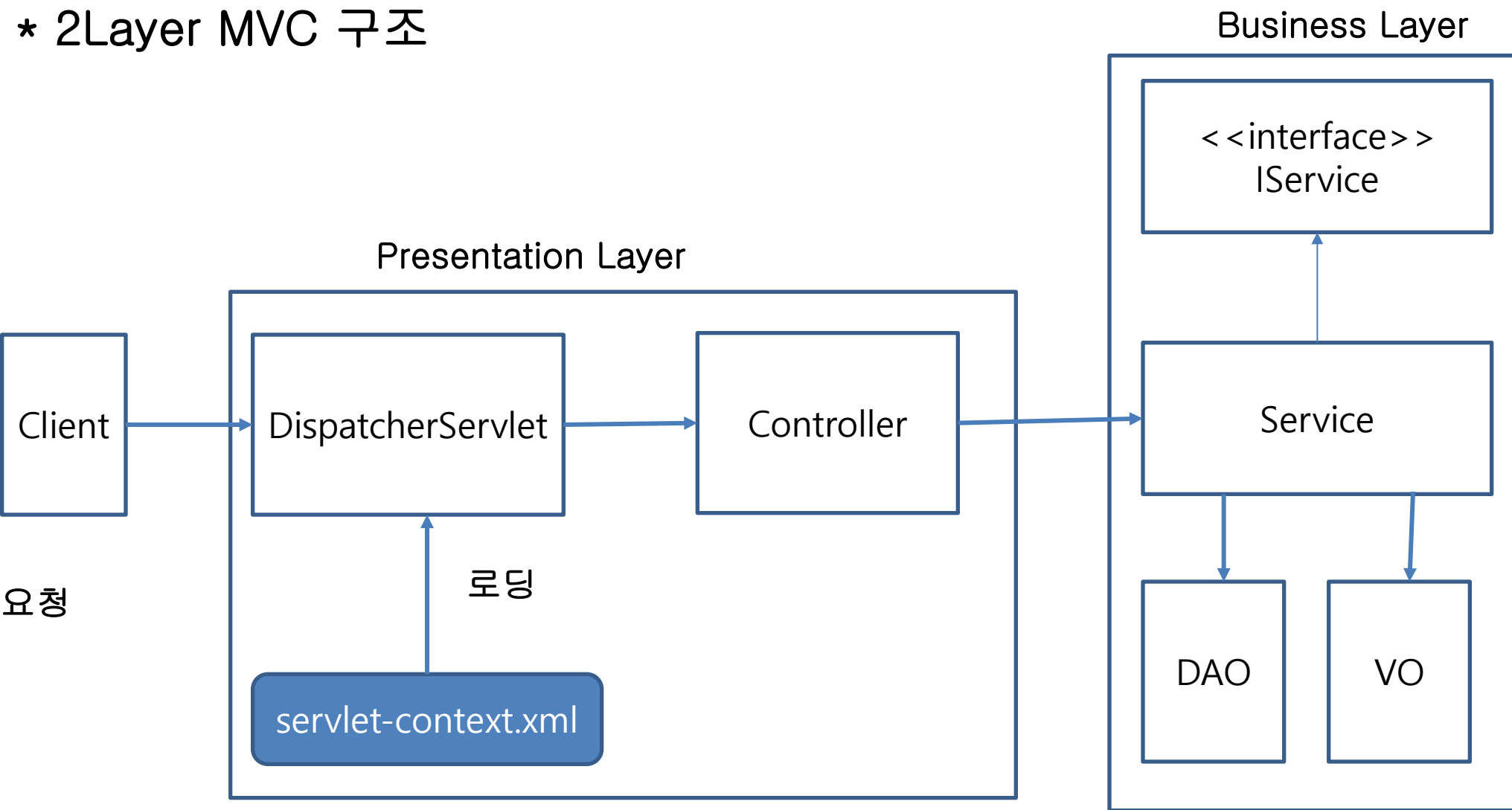
3. MVC 구현 (2Layer 구조)

- 1. MVCW 구조
- 2. MVC 구현(viewResolver 없는 경우)
- 3. MVC 구현(WEB-INF 디렉토리 접근)
- 4. Annotation 기반의 구현

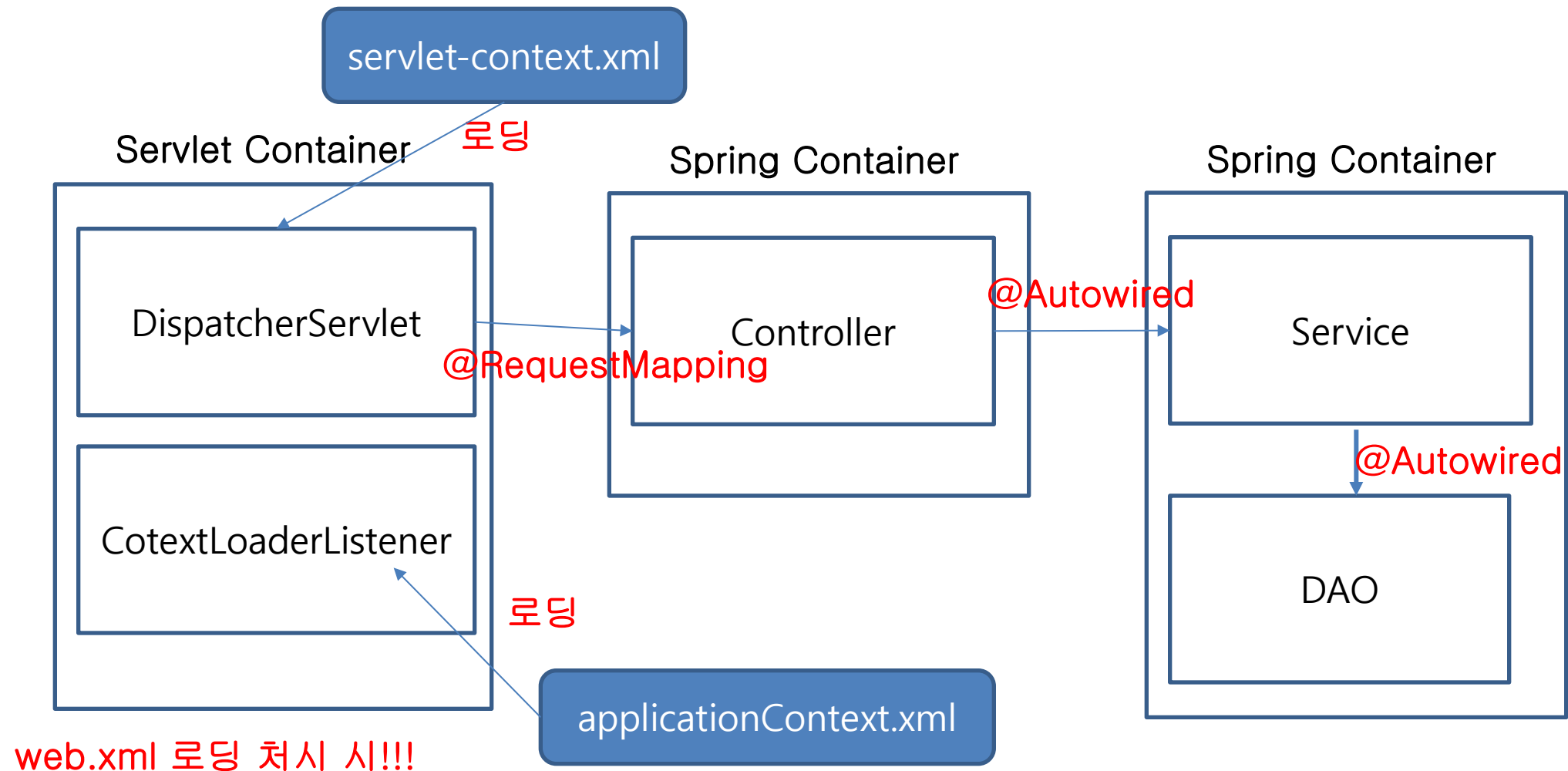
* 현재의 MVC 구조



* 2Layer MVC 구조



* 현재의 MVC 구조



1) applicationContext.xml

- web.xml의 ContextLoaderListener 객체에 의해 호출
- 루트 컨테이너 역할 수행.
- 자식 컨테이너(servlet-context.xml)에서 사용할 비즈니스 객체 생성
 - * 비즈니스 객체 (@Service, @Repository) 메모리 로딩.

2) Servlet-context.xml

- web.xml에 등록된 DispatcherServlet 객체에 의해 호출
- controller 객체(@Controller)를 메모리 로딩

1) ContextLoaderListener 등록

- web.xml 파일에서 applicationContext.xml파일(루트 컨테이너 정보)를 읽어 컨테이너를 생성한다.

아래의 두 위치 중 하나의 위치에 생성하여 web.xml에 등록한다.

a) src/main/resource/applicationContext.xml

b) /WEB-INF/applicationContext.xml

* web.xml(applicationContext.xml 등록)

```
<context-param>  
  <param-name>contextConfigLocation</param-name>  
  <param-value>classpath:applicationContext.xml</param-  
value>  
</context-param>  
<listener>  
  <listener-  
class>org.springframework.web.context.ContextLoaderListener  
</listener-class>  
</listener>
```


* web.xml(Servlet-context.xml 등록)

```
<servlet>
  <servlet-name>action</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>
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
<welcome-file-list>
```

2) GuestBookController.java

```
@Controller("guestbook")
public class GuestBookController {

    @Autowired
    private IGuestBookService guestbookService ;

    @RequestMapping(value="/insert.do")
    public String insertGuestBook(GuestbookVo vo) {
        guestbookService.insertGuestBook(vo);
        return "getguestbooklist.do" ;
    }

    @RequestMapping("/deleteform.do")
    public String deleteForm() {
        return "deleteform.jsp" ;
    }
}
```

3) GuestBookService

```
@Service("guestbookService")
public class GuestBookService implements IGuestBookService {

    @Autowired
    @Qualifier("dao")
    private IGuestbookDao guestBookDao ;

    @Override
    public void insertGuestBook(GuestbookVo vo) {
        guestBookDao.insert(vo);
    }

    @Override
    public void deleteGuestBook(GuestbookVo vo) {
        guestBookDao.delete(vo);
    }
}
```

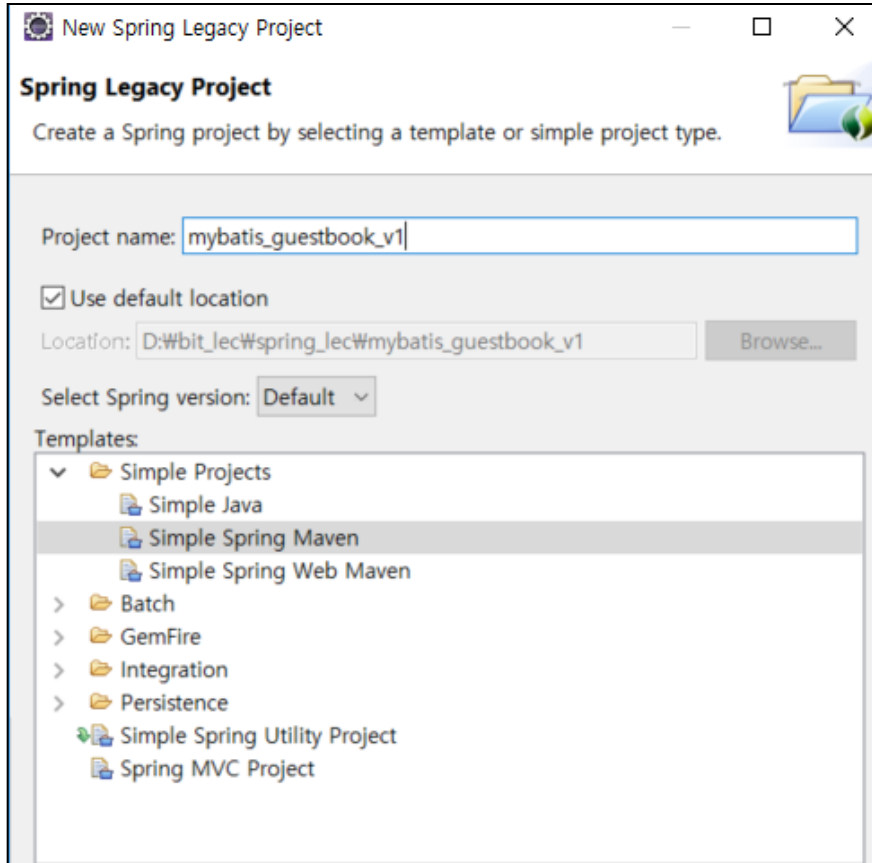
4. Mybatis

- 1. Mybatis 개요
- 2. 설정

- 개요

- XML 기반으로 DB와의 연동을 처리하기 위한 프레임워크.
 - DB 연동과 관련된 복잡한 자바코드코드는 Mybatis 프레임워크에서 내부적으로 처리, 개발자는 XML(쿼리) 만 관리하면 됨!!
-
- DB 관련 쿼리는 XML 로 관리
 - 자바 코드가 단순해 짐!!

* Mybatis 프로젝트 생성



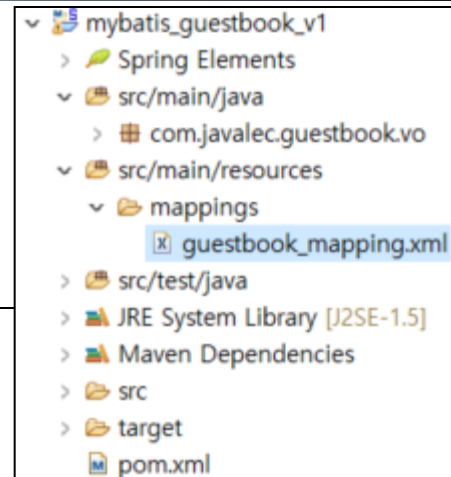
pom.xml

```
<!-- Mybatis -->
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.3.1</version>
</dependency>

<!-- Ibatis -->
<dependency>
  <groupId>org.apache.ibatis</groupId>
  <artifactId>ibatis-core</artifactId>
  <version>3.0</version>
</dependency>
```

1) SQL Mapper XML 파일 작성

– DB 연동을 위한 SQL 문을 XML에 저장한 파일



```
<?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">
```

```
<mapper namespace="GuestBookDao">
```

```
<insert id="insert">
```

```
    insert into guestbook values (guestbook_seq.nextval, #{name}, #{content}, #{password}, sysdate)
```

```
</insert>
```

```
<delete id="delete">
```

```
    DELETE GUESTBOOK WHERE no = #{no}
```

```
</delete>
```

```
<select id="getList" resultType="guestbook">
```

```
    SELECT no, name, content, password, to_char(reg_date, 'yyyy-mm-dd hh:mi:ss' ) regdate from guestbook order by regdate desc
```

```
</select>
```

```
    <select id="getKeywordList" resultType="guestbook">
```

```
    SELECT no, name, content, password, to_char(reg_date, 'yyyy-mm-dd hh:mi:ss' ) regdate
    from guestbook where content like #{keyword} order by regdate desc
```

```
</select>
```

```
</mapper>
```

Sql-map-config.xml의 <typeAliases>로 지정

2) Mybatis 환경설정 파일 생성

- Mybatis의 환경 설정 관련 파일 생성(sql-map-config.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <!-- Properties 파일 설정 -->
  <properties resource="db.properties" />
  <!-- Alias 설정 -->
  <typeAliases>
    <typeAlias alias="guestbook" type="com.javalec.guestbook.vo.GuestbookVo" />
  </typeAliases>
  <!-- DataSource 설정 -->
  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC" />
      <dataSource type="POOLED">
        <property name="driver" value="${jdbc.driverClassName}" />
        <property name="url" value="${jdbc.url}" />
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}" />
      </dataSource>
    </environment>
  </environments>
  <!-- Sql Mapper 설정 -->
  <mappers>
    <mapper resource="mappings/guestbook-mapping.xml" />
  </mappers>
</configuration>
```


* sql-map-config.xml의 세부 설정

1) <properties>

: 외부 프로퍼티 파일의 참조를 위해 선언. \${프로퍼티 이름} 사용

2) <typeAliases>

: 특정 클래스의 별칭 지정

여러 개의 <typeAlias> 지정 가능.

sql Mapper 파일에서 사용하여 설정을 간단하게 처리.

- * sql-map-config.xml의 세부 설정

- 3) <environments>

- : DataSource 설정 등 다양한 환경 설정 가능.

- 4) <mappers>

- : 여러 개의 <mapper> 가질 수 있음

- SQL 명령어가 저장된 XML 파일 등록.

1) SqlSession 객체 생성

```
public class SqlSessionFactoryBean {  
  
    private static SqlSessionFactory sessionFactory = null ;  
  
    public static SqlSession getSqlSession(){  
        try {  
            Reader reader = Resources.getResourceAsReader("sql-map-config.xml");  
            sessionFactory = new SqlSessionFactoryBuilder().build(reader) ;  
        }catch(Exception e) {  
            e.printStackTrace();  
        }  
        return sessionFactory.openSession();  
    }  
}
```

2) GuestBookDao 작성

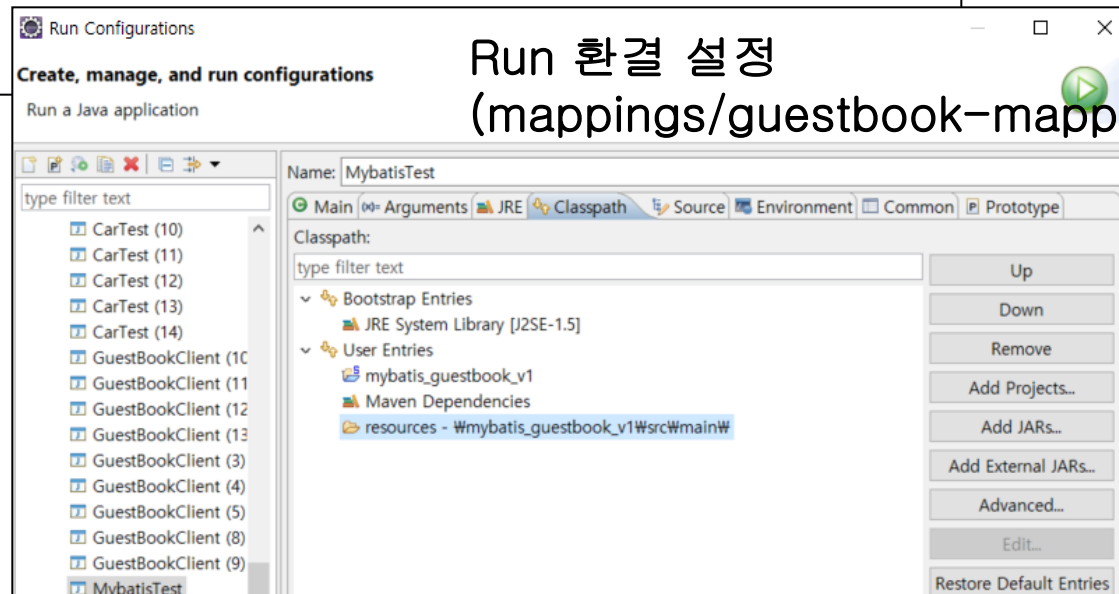
```
public class GuestBookDao {  
    private SqlSession mybatis ;  
    public GuestBookDao() {  
        mybatis = SqlSessionFactoryBean.getSqlSession() ;  
    }  
    public void insert(GuestbookVo vo) {  
        mybatis.insert("GuestBookDao.insert", vo);  
    }  
    public void delete(GuestbookVo vo) {  
        mybatis.delete("GuestBookDao.delete", vo);  
    }  
    public List<GuestbookVo> getKeywordList(Map map) {  
        return mybatis.selectList("GuestBookDao.getKeywordList", map);  
    }  
    public List<GuestbookVo> getList() {  
        return mybatis.selectList("GuestBookDao.getList");  
    }  
}
```

mapper.xml 파일에서 사용

3. 샘플 프로그램 작성

4) Test 프로그램 작성

```
public class MybatisTest {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        GuestBookDao dao = new GuestBookDao();  
        List<GuestbookVo> list = dao.getList();  
        for(GuestbookVo guestbook : list) {  
            System.out.println(guestbook.toString());  
        }  
    }  
}
```



(mappings/guestbook-mapping.xml 파일 클래스 패스 추가)

1) <mapper namespace="guestbook">

- mapper 파일의 root 엘리먼트
- 여러 mapper 파일 중 유일하게 식별하기 위한 네임스페이스 제공
- DAO 클래스에서 해당 “네임스페이스이름.아이디” 로 SQL문 사용

1) <SELECT>

a) id 속성 : 전체 mapper 파일 내(네임스페이스)에서 유일한 구분자.

DAO 클래스에서 SQL을 식별하기 위한 사용.

b) parameterType 속성

: SQL 실행을 위한 입력 데이터 타입 지정

- 기본타입, VO 형태의 클래스 지정(패키지명.클래스명)

- 메인 설정 파일(sql-map-config.xml)에 등록된 <typeAlias>의

별칭 사용,

- 생략 가능.

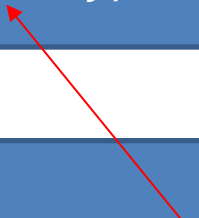
1) <SELECT>

c) resultType 속성

- 검색결과로 리턴되는 ResultSet 클래스가 어떤 자바 객체와 매핑될지 지정

```
<typeAlias alias="guestbook" type="com.javalec.guestbook.vo.GuestbookVo" />
```

```
<select id="getList" resultType="guestbook">  
SELECT no, name, content, password, to_char(reg_date, 'yyyy-mm-dd hh:mi:ss' )  
regdate from guestbook order by regdate desc  
</select>
```



1) <SELECT>


d) resultMap 속성

– 검색결과를 parameterType 속성과 매핑할 수 없는 경우 사용

예) 조인 검색 시 결과 중 일부를 특정 VO 객체로 리턴 시 사용

```
<resultMap type="guestbook" id="guestbookResult">
  <id property="no" column="no"/>
  <result property="name" column="name"/>
  <result property="content" column="content"/>
  <result property="password" column="password"/>
  <result property="regDate" column="regdate"/>
</resultMap>

<select id="getList" resultMap="guestbookResult">
  SELECT no, name, content, password, to_char(reg_date, 'yyyy-mm-dd hh:mi:ss' )
  regdate from guestbook order by regdate desc
</select>
```



2) <insert> 엘리먼트

```
<insert id="insert">  
insert into guestbook values (guestbook_seq.nextval, #{name}, #{content},  
#{password}, sysdate)  
</insert>
```

3) <update> <delete> 엘리먼트

- SQL의 update, delete 작업을 위한 엘리먼트
- parameterType를 통해 SQL에 적용할 파라미터 값을 전달

```
<delete id="delete">  
DELETE GUESTBOOK WHERE no = #{no}  
</delete>
```

4) CDATA Section 사용

- SQL 문에서 XML에서 사용되는 특수 기호가 존재 시 사용하는 문법
- XML 파서가 해석하지 못하게 하는 기능 제공

예)

```
<SELECT id="getList" resultType="guestbook">  
    SELECT * FROM GUESTBOOK WHERE NO <= #{no}  
</SELECT>
```

→ <SELECT id="getList" resultType="guestbook">

```
<![CDATA[SELECT * FROM GUESTBOOK WHERE NO <= #{no}  
]]>
```

```
</SELECT>
```

*SqlSession 객체

- Mapper XML에 등록된 SQL을 실행하기 위한 다양한 API 제공

* SqlSession 객체 생성 법

```
Reader reader = Resources.getResourceAsReader("sql-map-config.xml");  
sessionFactory = new SqlSessionFactoryBuilder().build(reader) ;
```

```
SqlSession sqlSession = sessionFactory.openSession();
```

*SqlSession 객체

– Mapper XML에 등록된 SQL을 실행하기 위한 다양한 API 제공

1) selectOne() 메소드

: 하나의 데이터 검색을 위한 SQL문 실행을 위해 적용.

- **public Object selectOne(String statement)**
- **public Object selectOne(String statement, Object parameter)**

2) selectList() 메소드

: 다수의 검색을 위한 SQL 구문 실행

- **public List selectList(String statement)**
- **public List selectList(String statement, Object parameter)**

3) insert(), update(), delete() 메소드

: 입력, 변경, 삭제를 위한 SQL 구문 실행

- **public int insert(String statement , Object parameter)**
- **public int update(String statement , Object parameter)**
- **public int delete(String statement, Object parameter)**

1) mybatis 라이브러리 설정

- 스프링에 mybatis를 연동하기 위해 다음과 같은 라이브러리 필요.

```
<!-- Mybatis -->
<dependency>
<groupId>org.mybatis</groupId>
<artifactId>mybatis</artifactId>
<version>3.3.1</version>
</dependency>
<!-- Mybatis -->
<dependency>
<groupId>org.mybatis</groupId>
<artifactId>mybatis-spring</artifactId>
<version>1.2.4</version>
</dependency>
```

- org.mybatis.spring.SqlSessionFactoryBean
- org.mybatis.spring.SqlSessionTemplate

2) applicationContext.xml 설정

- 스프링과 Mybatis 연동을 위한 설정 적용.

```
<!-- DataSource 설정 -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
destroy-method="close">
<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"/>
<property name="url" value="jdbc:oracle:thin:@localhost:1521:xe"/>
<property name="username" value="beast"/>
<property name="password" value="1111"/>
</bean>
<!-- Mybatis 연결 설정 -->
<bean id="sqlSession" class="org.mybatis.spring.SqlSessionFactoryBean">
<property name="dataSource" ref="dataSource"/>
<property name="configLocation" value="classpath:sql-map-config.xml"/>
</bean>
<bean class="org.mybatis.spring.SqlSessionTemplate">
<constructor-arg ref="sqlSession"/> </constructor-arg>
</bean>
```

메인 설정 파일

3) Dao 클래스 적용.

@Autowired

```
private SqlSessionTemplate mybatis;
```

```
public void insert(GuestbookVo vo) {  
    mybatis.insert("GuestBookDao.insert", vo);  
    //mybatis.commit();  
}
```

```
public void delete(GuestbookVo vo) {  
    mybatis.delete("GuestBookDao.delete", vo);  
    //mybatis.commit();  
}
```

```
public List<GuestbookVo> getKeywordList(Map map) {  
    return mybatis.selectList("GuestBookDao.getKeywordList", map);  
}
```

```
public List<GuestbookVo> getList() {  
    return mybatis.selectList("GuestBookDao.getList");  
}
```


* ORM 플러그인 설치

- Mybatis 관련 복잡한 XML 설정 관리 플러그인