|  |
| --- |
| Programming with ARTECTURE WEB 2.0 |
|  |
| Rapid application Development for the java platform |

목차 정보

[1. MVC 프레임워크 3](#_Toc348430462)

[1.1. MVC (Model-View-Controller) 디자인 패턴 4](#_Toc348430463)

[1.1.1. Model1 과 Model2 패턴 4](#_Toc348430464)

[1.1.2. MVC 디자인 모델 5](#_Toc348430465)

[1.2. Struts1 6](#_Toc348430466)

[1.2.1. 스트럿츠 아키텍처 6](#_Toc348430467)

[1.2.2. Action 구현 8](#_Toc348430468)

[1.2.3. 예외처리 9](#_Toc348430469)

[1.2.4. 코드 기반 예외처리 13](#_Toc348430470)

[1.2.5. 여러 포맷으로 결과 보여주기 13](#_Toc348430471)

[1.2.6. JSP 15](#_Toc348430472)

[1.2.7. 템플릿 기반 XML 16](#_Toc348430473)

[1.2.8. 파일업로드 16](#_Toc348430474)

[1.3. Struts2 17](#_Toc348430475)

[1.3.1. Struts2 소개 17](#_Toc348430476)

[1.3.2. 아키텍처 19](#_Toc348430477)

[1.3.3. 환경설정 20](#_Toc348430478)

[2. 뷰 기술 21](#_Toc348430479)

[2.1. Sitemesh 22](#_Toc348430480)

[2.1.1. Sitemesh 설치 22](#_Toc348430481)

[2.2. Velocity & Freemarker 25](#_Toc348430482)

[2.2.1. 설치 25](#_Toc348430483)

[2.3. DWR 27](#_Toc348430484)

[2.3.1. DWR 소개 27](#_Toc348430485)

[2.3.2. DWR 설치 27](#_Toc348430486)

[3. 웹 프로그램 개발하기 30](#_Toc348430487)

|  |  |
| --- | --- |
|  |  |
| MVC 프레임워크 | |
|  |  |

## MVC (Model-View-Controller) 디자인 패턴

### Model1 과 Model2 패턴

Model 1 방식의 웹 어플리케이션이란, 한 개의 JSP에서 모든 비지니스 로직을 처리하고, 그 결과를 바로 출력하는 웹 프로그래밍 모델이다. 이 방식은 아주 단순한 웹 어플리케이션에서는 빠르고 단순하게 프로그램을 짤 수 있어 어플리케이션이 커지게 되면 그 복잡도가 크게 증가하여 디버깅이 어렵고, 한 번 수정할 것이 생기면 수정할 위치를 찾기도 어려울 뿐만 아니라, 단순히 비지니스 로직이 바뀌는 것임에도 화면 출력 부분의 수정이 필요하게 된다(디자인 변경을 위해 프로그램 로직을 바꿔야 하는 경우도 있다). 또한, 한 개 파일의 프로그램과 디자인이 섞여 있기 때문에, 디자이너와 프로그래머의 업무 경계가 불명확해지는 문제가 있다.



그림 ‑ Model1 디자인

이러한 문제를 해결하고자 등장한 것이 Model 2[[1]](#footnote-1) 이다. Model 2란 비즈니스 로직과 (결과) 화면을 분리 하는 것이다.



그림 ‑ Model 2 디자인

모든 요청을 서블릿이 받아 비지니스 로직을 처리하고, 결과 출력을 위해 JSP로 포워딩하도록 하면 어플리케이션 복잡도가 (Model1 과 비교하여) 많이 감소하게 된다. 그러나 이경우 서블릿의 수가 지나치게 늘어나 web.xml 이 복잡해지는 문제가 있다. 현재는 Model 2 디자인을 Model-View-Controller[[2]](#footnote-2)(이하 MVC) 패턴으로 설계하는 것이 일반적인 방식이 되었다.

### MVC 디자인 모델

MVC 디자인 모델은 UI 가 포함된 프레임워크에는 거의 필수적으로 사용되는 패턴으로 원래는 일반 GUI 애플리케이션을 개발할 때 UI 와 비즈니스 로직을 효과적으로 분리하기 위하여 고안되었으나 웹 애플리케이션에서 역시 그 효과가 입증되어 널리 쓰이고 있다. MVC 디자인 구조는 과거 MFC 프로그램에서 등장했던 Document-View 구조를 한층 더 발전시킨 것으로 프로그램의 구성요소를 모델, 뷰, 컨트롤러로 나누어서 각각 다른 역할을 처리하도록 하는 것이다.



그림 ‑ Model, View, Controller 의 역할



그림 ‑ 서버-클라이언트 환경에서 Model, View, Controller

## Struts1

### 스트럿츠 아키텍처

스트럿트는 아파치 그룹에 지원 받은 오픈소스 프로젝트로 Craig McClanahan에 의해 2000년 5월 개발이 시작되었고 얼마 후 오픈 소스 커뮤니티 Apache 로 이전되어 많은 사람들에 의해 공동 개발된 MVC Model2 디자인에 기반한 웹 어플리케이션 프레임워크이다.

스트럿트는 Controller 계층을 ActionServlet 형태로 구현하고 JSP 태그 라이브러리를 이용하여 View 계층을 구현할 것을 권장하였다. 또한 Action 클래스를 통하여 Model 계층에 대한 래퍼를 제공하고 있다. 그림 1‑5은 MVC Model2 와 스트럿트 환경에서 어떻게 사용자의 요청을 처리하는 가를 보여주고 있다.



그림 ‑ MVC Model2 와 스트럿츠 비교

1. View1(JSP, HTML, WML)으로부터 사용자 요청이 발생한다.
2. ActionServlet이 사용자 요청을 받는다. ActionServlet은 Struts에서 Controller의 기능을 수행한다. 사용자에 의해 요청된 URI와 Configuration XML을 기초로 사용자 요구를 만족시키기 위해 필요한 비즈니스 로직(Model)을 호출하는 Action을 호출한다.
3. Action 클래스는 사용자 요구에 맞는 Model 컴포넌트에 대한 비즈니스 로직을 수행시킨다.
4. Action Class가 비즈니스 로직 수행을 끝내면 ActionServlet이 다시 제어권을 넘겨받는다. Action Class의 수행을 통해 발생한 결과 값을 가리키는 key 값을 ActionServlet이 넘겨받게 되고 ActionServlet은 이 값을 기초로 사용자에게 결과값을 전송한다.
5. ActionServlet이 Action 클래스로부터 넘겨받은 key값을 기초로 View를 생성하여 사용자에게 보여줌으로써 Struts 프로세스 사이클이 끝이 난다.

### Action 구현

ARCHITECTURE WEB 2.0 for JAVA 는 스트럿츠 Action 클래스를 확장하는 방법이 제공된다.

* FrameworkActionSupport
* FrameworkDispatchActionSupport



그림 ‑ 스트럿츠의 Action 와 DispatchAction 에 대응하는 확장 클래스

이 클래스들은 컨테이너에 존재하는 클래스 타입을 인자로 해당하는 객체를 getComponent 함수와 객체를 컨텍스트 존재하는 객체들을 사용하여 자동으로 인자로 전달된 객체의 프로퍼티를 설정하는 autowireComponent 함수를 제공한다.

architecture.examples.struts.action.QueryAction.java

**package** architecture.examples.struts.action;

**import** java.util.List;

**import** java.util.Map;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.commons.lang.StringUtils;

**import** org.apache.struts.action.ActionForm;

**import** org.apache.struts.action.ActionForward;

**import** org.apache.struts.action.ActionMapping;

**import** architecture.ee.services.SqlQueryClient;

**import** architecture.ee.util.OutputFormat;

**import** architecture.ee.web.struts.action.FrameworkDispatchActionSupport;

**import** architecture.ee.web.util.ParamUtils;

**import** architecture.ee.web.util.ServletUtils;

**import** architecture.ee.web.view.json.JsonView;

DispatchAction 을 구현하기위하여 FrameworkDispatchActionSupport 클래스를 상속

**public** **class** QueryAction **extends** FrameworkDispatchActionSupport {

**private** String [] stringToArray(String parametersString){

**return** StringUtils.*splitPreserveAllTokens*(parametersString, ",");

}

**public** ActionForward list(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) **throws** Exception {

OutputFormat output = getOutputFormat(request, response);

String statement = ParamUtils.*getParameter*(request, "statement" );

String parametersString = ParamUtils.*getParameter*(request, "parameters", **null** );

SqlQueryClient client = getComponent("sqlQueryClient", SqlQueryClient.**class**);

List<Map<String, Object>> list ;

getComponent 함수를 사용하여

컨테이너에 등록된 서비스에 접근할 수 있다.

**if**( StringUtils.*isEmpty*( parametersString) )

list = client.list(statement);

**else**

list = client.list(statement, stringToArray(parametersString));

Map model = getModelMap(request, response);

model.put("items", list);

**if**(output == OutputFormat.*JSON* ){

JsonView view = **new** JsonView();

view.setModelKey("items");

view.render(model, request, response);

**return** **null**;

}**else**{

saveModelMap(request, model);

}

**return** (mapping.findForward(output.name().toLowerCase()));

}

}

### 예외처리

스트럿트 프레임워크는 예외 처리를 위하여 ActionMessage클래스를 사용할 것을 권장하고 있다. ActionMessage클래스는 어플리케이션에서 발생된 예외를 잡아 뷰에 에러 메시지들을 캡슐화하여 전송한다. 뷰에 전달된 ActionMessage 는 스트럿츠에서 제공되는 유틸리티 (태그 라이브러리)를 사용하여 에러 메시지로 랜더링된다.



그림 ‑ 스트럿츠의ActionMessage 기반의 예외처리

struts-config.xml 파일에 Action 에서 예외(java.lang.Throwable)가 발생되면 디폴트로 던져진 예외가 처리될 수 있도록 global-exceptions 설정을 추가한다. 또한 global-forward 설정을 추가하여 Action 에서 오류가 발생할 때 error 에 해당하는 페이지로 이동하도록 한다.

WEB-INF/struts-config/struts-config.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE struts-config PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"

"http://struts.apache.org/dtds/struts-config\_1\_3.dtd">

<struts-config>

<form-beans></form-beans>

<global-exceptions>

<exception key=*"global"* type=*"java.lang.Throwable"*

handler=*"architecture.ee.web.struts.action.FrameworkExceptionHandler"*

path=*"/includes/jsp/error.jsp"* />

</global-exceptions>

<global-forwards>

<forward name=*"error"* path=*"/includes/jsp/error.jsp"*></forward>

</global-forwards>

<action-mappings></action-mappings>

<message-resources parameter=*"MessageResources"* />

</struts-config>

액션 구현 클래스에서 예상되는 오류에 따라 예외를 처리하는 방법은 ActionMessage 클래스를 사용하는 것이다 (Struts 1.2이전버전에는 ActionError 클래스를 사용하였다). 이 접근 방법은 개발자가 직접 적절한 오류 메시지 형식으로 오류 메시지를 뷰에 전달할 수 있는 이점이 있다. 또한 개발자가 쉽게 I18N 기능을 구현하기 위한 좋은 방법이기도 하다. 또는 struts-config.xml 파일에서 Action 설정에서 직접 특정 오류 페이지를 지정하여 처리하는 방법도 있다.

architecture.examples.struts.action.QueryAction.java

**package** architecture.examples.struts.action;

**import** java.util.List;

**import** java.util.Map;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.commons.lang.StringUtils;

**import** org.apache.struts.action.ActionForm;

**import** org.apache.struts.action.ActionForward;

**import** org.apache.struts.action.ActionMapping;

**import** org.apache.struts.action.ActionMessage;

**import** org.apache.struts.action.ActionMessages;

**import** architecture.ee.services.SqlQueryClient;

**import** architecture.ee.util.OutputFormat;

**import** architecture.ee.web.struts.action.FrameworkDispatchActionSupport;

**import** architecture.ee.web.util.ParamUtils;

**import** architecture.ee.web.util.ServletUtils;

**import** architecture.ee.web.view.json.JsonView;

**public** **class** QueryAction **extends** FrameworkDispatchActionSupport {

**private** String [] stringToArray(String parametersString){

**return** StringUtils.*splitPreserveAllTokens*(parametersString, ",");

}

**public** ActionForward listWithError(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response) **throws** Exception {

OutputFormat output = getOutputFormat(request, response);

**try** {

String statement = ParamUtils.*getParameter*(request, "statement" );

String parametersString = ParamUtils.*getParameter*(request, "parameters", **null** );

SqlQueryClient client = getComponent("sqlQueryClient", SqlQueryClient.**class**);

List<Map<String, Object>> list ;

**if**( StringUtils.*isEmpty*( parametersString) )

list = client.list(statement);

**else**

list = client.list(statement, stringToArray(parametersString));

Map model = getModelMap(request, response);

model.put("items", list);

**if**(output == OutputFormat.*JSON* ){

JsonView view = **new** JsonView();

view.setModelKey("items");

view.render(model, request, response);

**return** **null**;

Action 에서 발생이 예상되는

오류를 catch 하여 ActionMessages 를 생성/저장하고

Error페이지로 forward 한다.

}**else**{

saveModelMap(request, model);

}

} **catch** (Exception e) {

ActionMessages errors = **new** ActionMessages();

errors.add( Globals.*ERROR\_KEY*,

**new** ActionMessage(e.getMessage()));

saveErrors(request, errors);

**return** mapping.findForward("error");

}

**return** (mapping.findForward(output.name().toLowerCase()));

}

}

WEB-INF/struts-config/sample-struts-config.xml

<?xml version=*"1.0"* encoding=*"ISO-8859-1"* ?>

<!DOCTYPE struts-config PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"

"http://struts.apache.org/dtds/struts-config\_1\_3.dtd">

<struts-config>

<action-mappings>

<action path=*"/query"* type=*"architecture.examples.struts.action.QueryAction"* parameter=*"method"* scope=*"request"*>

<forward name=*"html"* path=*"/includes/jsp/list.jsp"* />

<exception key=*"database.error.duplicate"* path=*"/includes/jsp/error.jsp "*

type=*"javax.sql.SQLException"* />

</action>

Action 에서 발생이 예상되는

오류와 처리할 페이지를 설정한다.

</action-mappings>

</struts-config>

### 코드 기반 예외처리

코드 값을 사용하여 오류를 처리하면 보다 효과적으로 오류를 처리할 수 있는데 이를 위하여 서비스 또는 Action 에서 Coedable 인터페이스를 구현하는 예외 객체를 사용하여야 한다. 오류화면에서는 코드 값에 해당하는 메시지를 사용자에게 전달할 수 도 있게 된다.



그림 ‑ 서비스 또는 Action 에서 Codeable 예외를 던지면 error.jsp 에 의하여 처리된다.

### 여러 포맷으로 결과 보여주기

요청에 따라 Action 의 처리결과를 HTML 뿐 아니라 XML, JSON 과 같은 형식으로 보여줄 수 있다. 이를 위하여 Action 을 호출할 때 output 파라메터 값(html, xml, json)을 사용한다. Action 에서는 다음과 같은 방식으로 요청된 Action 에 결과 데이터 유형을 결정할 수 있다. 결과를 보여줄 view 로 foward 역시 동일한 이름을 사용하기 때문에 json 유형이 아닌 경우에는 struts-config 에 html 과 xml 에 해당하는 view 페이지를 지정하여야 한다.



그림 ‑ output 파라메터 값에 따른 Action 처리

architecture.examples.struts.action.QueryAction.java

**public** ActionForward list(ActionMapping mapping, ActionForm form, HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

OutputFormat output = getOutputFormat(request, response);

// 중략

**if**(output == OutputFormat.*JSON* ){

JsonView view = **new** JsonView();

view.setModelKey("items");

view.render(model, request, response);

**return** **null**;

}**else**{

saveModelMap(request, model);

}

**return** (mapping.findForward(output.name().toLowerCase()));

}

WEB-INF/struts-config/sample-struts-config.xml

<?xml version=*"1.0"* encoding=*"ISO-8859-1"* ?>

<!DOCTYPE struts-config PUBLIC

"-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"

"http://struts.apache.org/dtds/struts-config\_1\_3.dtd">

<struts-config>

<action-mappings>

<action path=*"/query"* type=*"architecture.examples.struts.action.QueryAction"*

parameter=*"method"* scope=*"request"*>

<forward name=*"html"* path=*"/includes/jsp/list.jsp"* />

<forward name=*"xml"* path=*"/templates/freemarker/items-xml.ftl"* />

</action>

</action-mappings>

</struts-config>

### JSP

PUSH 방식의 Struts 에서 Action 은 뷰에 해당하는 JSP에게 ModelMap 객체를 넘겨주고 JSP 는 ModelMap 을 꺼내어 사용하게 된다.

architecture.examples.struts.action.QueryAction.java

**public** ActionForward list(ActionMapping mapping, ActionForm form, HttpServletRequest request,

HttpServletResponse response) **throws** Exception {

OutputFormat output = getOutputFormat(request, response);

String statement = ParamUtils.*getParameter*(request, "statement" );

String parametersString = ParamUtils.*getParameter*(request, "parameters", **null** );

SqlQueryClient client = getComponent("sqlQueryClient", SqlQueryClient.**class**);

List<Map<String, Object>> list ;

**if**( StringUtils.*isEmpty*( parametersString) )

list = client.list(statement);

**else**

list = client.list(statement, stringToArray(parametersString));

Map model = getModelMap(request, response);

결과 데이터를 Map 에 넣는다.

model.put("items", list);

**if**(output == OutputFormat.*JSON* ){

JsonView view = **new** JsonView();

view.setModelKey("items");

view.render(model, request, response);

**return** **null**;

request 의 attribute 에 저장한다.

}**else**{

saveModelMap(request, model);

}

**return** (mapping.findForward(output.name().toLowerCase()));

}

/includes/jsp/list.jsp

<%@ page import=*"architecture.ee.services.\*,architecture.ee.web.util.\*"*%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>쿼리 결과</title>

</head>

<body>

<%= request.getAttribute( WebApplicatioinConstants.MODEL\_ATTRIBUTE) %>

</body>

</html>

### 템플릿 기반 XML

### 파일업로드

## Struts2

### Struts2 소개

스트럿츠2는 2001년 공개된 이후 자바 영역에서 웹 어플리케이션 개발을 위한 표준 프레임워크로 널리 사용되었던 Struts1 과는 사실상 다른 프레임워크로 공개 소프트웨어 WebWork 2.2를 계승하는 프로젝트로 Struts1 보다는 WebWork 2.2 와 유사하다.

초기 대부분의 웹 어플리케이션들은 Page Controller 패턴을 사용하였다. Page Controller 패턴은 요청이 많을 경우 부하가 걸린다는 것이고 하나의 페이지마다 하나씩의 Page Controller 가 존재한다는 문제점을 가지고 있었다. Struts 1 과 Struts 2 는 Page Controller 패턴의 문제점을 개선한 형태의 Front Controller 디자인 패턴을 이용한 MVC 모델이라는 공통점을 가지고 있다.



그림 ‑ Front Controller MVC Model

Struts 1 과 Struts 2 의 MVC 모델의 차이는 Push Model & Pull Model 방식으로 설명할 수 있다. Struts1 에 적용된 Push Model 은 초기 MVC 모델에 주로 채택되었던 방법으로 사용자의 요청에 의해 처리된 결과를 View 에 밀어 넣는 (Push) 방식이고, Struts 2 에 적용된 Pull 방식은 View 에서 다수의 Controller 로 부터 필요에 따라 처리 결과를 댕겨오는 (Pull) 방식이다. Pull Model 은 필요한 컴포넌트를 호출하여 처리하는 방식이 가능하기 때문에 이를 Component MVC 라고도 부르며, 최근에는 많은 웹 어플리케이션 프레임워크들이 이 방식을 사용하고 있다.



그림 ‑ Push Model



그림 ‑ Pull Model

### 아키텍처

그림 1‑13는 스트럿츠2의 아키텍처를 보여준다.



그림 ‑ Struts 2 아키텍처

**FilterDispatcher**

표준 필터 FilterDispatcher 는 ActionMapper 정보를 이용하여 Acton 을 호출할 것인가를 결정한다. 만일 호출하여야 한다면 ActionProxy 에게 제어를 위임한다.

**ActionProxy**

ActionProxy 는 Struts.xml 파일을 사용하여 초기화되는 ConfigurationManager 는 참조하여 명령 패턴을 구현하는 ActionInvocation 을 생성한다. ActionInvocation 은 (설정된 경우) Inteceptor들을 호출 하고 Action 을 호출한다. ActionInvocation 는 Action 호출 결과에 해당하는 Result 를 찾는다. 랜더링될 JSP 또는 Freemarker 같은 템플릿을 포함하는 Result 를 실행한다. 이제 반대 순서로 Inteceptor 들을 호출한다.

### 환경설정

스트럿츠2 기술을 사용하기 위하여 다음과 같이 WEB-INF/web.xml 파일을 수정한다. 다운로드는 <http://struts.apache.org/2.x/index.html> 에서 한다. 편의를 위하여 ARCHITECTURE WEB 2.0 의 struts2 예제는 struts2 관련 라이브러리들을 포함하고 있다.

WEB-INF/web.xml

<filter>

<filter-name>struts2-prepare</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareFilter</filter-class>

</filter>

<filter>

<filter-name>struts2-execute</filter-name>

<filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2-prepare</filter-name>

<url-pattern>\*.do</url-pattern>

</filter-mapping>

<filter-mapping>

<filter-name>struts2-execute</filter-name>

<url-pattern>\*.do</url-pattern>

</filter-mapping>

|  |  |
| --- | --- |
|  |  |
| 뷰 기술 | |
|  |  |

## Sitemesh

ARCHITECTURE 2.0 플랫폼은 Gang of Four 의 Decorator Pattern 을 적용하여 웹페이지의 레이아웃을 관리할 수 있는 Sitemesh 프레임워크를 지원한다. [Sitemesh](http://wiki.sitemesh.org/display/sitemesh/Home) 을 사용하면 오직 콘텐츠에 해당하는 영역만을 작업하여 웹 응용프로그램을 구현하는 것이 가능하기 때문에 UI 영역에서의 코드 중복을 최소화 할 수 있다.



그림 ‑

### Sitemesh 설치

Sitemesh 사용을 위하여 먼저 <http://wiki.sitemesh.org/display/sitemesh/Download> 에 2.4.x 버전의 배포본을 다운로드 한다. (ARCHITECTURE 2.0 이 제공하는 예제는 이미 포함하고 있다) 다운받은 Sitemesh 라이브러리를 WEB-INF/lib 경로에 복사한다. 다음으로 web.xml에 필터 설정을 추가한다.

WEB-INF/web.xml

<filter>

<filter-name>sitemesh</filter-name>

<filter-class>com.opensymphony.sitemesh.webapp.SiteMeshFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>sitemesh</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

filter-mapping 설정을 아래와 같이 할 수 도 있다.

WEB-INF/web.xml

<filter-mapping>

<filter-name>sitemesh</filter-name>

<url-pattern>/\*</url-pattern>

<dispatcher>REQUEST</dispatcher>

<dispatcher>FORWARD</dispatcher>

<dispatcher>INCLUDE</dispatcher>

</filter-mapping>



그림 ‑ Sitemesh필터

이제 다운받은 Sitemesh 배포판 “[sitemesh-example.war](http://wiki.sitemesh.org/download/attachments/294938/sitemesh-example.war?version=1&modificationDate=1309409022953)“ 의 WEB-INF/sitemesh.xml 파일을 WEB-INF 경로에 복사한다. (ARCHITECTURE 2.0 이 제공하는 예제는 이미 포함하고 있다)

이제 모든 요청에 SiteMeshFilter 가 적용된다. 이제 WEB-INF/decorators.xml 파일을 생성하여 레이아웃으로 사용하게 될 페이지들을 설정한다.

WEB-INF/decorators.xml

<?xml version=*"1.0"* encoding=*"ISO-8859-1"*?>

<decorators defaultdir=*"/decorators"*>

<excludes>

<pattern>/css/\*</pattern>

<pattern>/images/\*</pattern>

<pattern>/includes/\*</pattern>

</excludes>

<decorator name=*"main"* page=*"main.jsp"*>

<pattern>/\*</pattern>

</decorator>

<decorator name=*"panel"* page=*"panel.jsp"*/>

<decorator name=*"printable"* page=*"printable.jsp"*/>

<decorator name=*"black"* page=*"black.jsp"*/>

</decorators>

위와 같은 설정에서는 index.jsp 를 호출하는 경우 /decorators/main.jsp를 사용하여 index.jsp를 완전하게 한다.

## Velocity & Freemarker

Freemarker 와 Velocity 는 가장 널리 사용되고 있는 템플릿 엔진이다. 두 언어는 사용에 있어 매우 유사하다.

### 설치

ARCHITECTURE WEB 2.0 for JAVA 는 Velocity 와 Freemarker 지원을 위하여 다음과 같은 서블릿 클래스들을 제공하고 있다. 현재는 freemarker 만 지원하고 있다.

* FreeMarkerViewRendererServlet
* VelocityViewRendererServlet

WEB-INF/web.xml (Freemarker)

<servlet>

<servlet-name>freemarker-renderer</servlet-name>

<servlet-class>architecture.ee.web.servlet.FreeMarkerViewRendererServlet</servlet-class>

<load-on-startup>2</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>freemarker-renderer</servlet-name>

<url-pattern>\*.ftl</url-pattern>

</servlet-mapping>

WEB-INF/web.xml (Velocity)

<servlet>

<servlet-name>velocity-renderer</servlet-name>

<servlet-class>architecture.ee.web.servlet.VelocityViewRendererServlet</servlet-class>

<load-on-startup>2</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>velocity-renderer</servlet-name>

<url-pattern>\*.vm</url-pattern>

</servlet-mapping>

이제 startup-config.xml 에 템플릿 엔진 사용 여부 및 관련 속성값들을 지정하여야 한다.

WEB-INF/setup-config.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<startup-config>

<view>

디버깅 여부

<render>

사용 유무

<freemarker>

<debug>true</debug>

템플릿 파일 위치 경로. 디폴트 값은 서블릿 컨텍스트 루트.

<enabled>true</enabled>

<source>

<location>/WEB-INF/template/ftl/</location>

</source>

</freemarker>

<velocity>

<enabled>false</enabled>

</velocity>

</render>

</view>

마지막으로 webApplicaitonContext.xml 파일에 다음과 같은 정의를 추가한다.

WEB-INF/context-config/webApplicationContext.xml

<import resource=*"classpath:context/freemarkerSubsystemContext.xml"*/>

## DWR

### DWR 소개

DWR은 AJAX(Asynchronous JavaScript and XML)기술을 웹 사이트에 적용하기 위한 공개소프트웨어이다. DWR 는 크게 2개의 파트로 구성된다.

* 서버측의 자바 서블릿은 서버로 요청 데이터를 브라우저로 전달한다.
* 브라우저에서 구동되는 JavaScript는 동적으로 서버의 자바 코드를 호출하고 그 결과를 리턴한다.

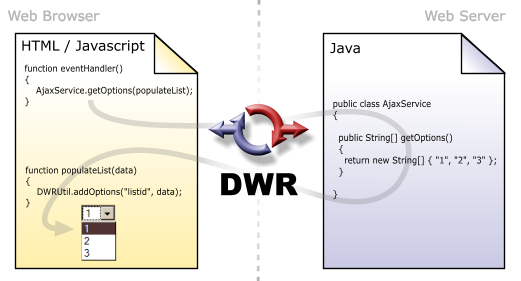


그림 ‑

### DWR 설치

DWR 를 사용할 수 있도록 web.xml에 설정을 추가한다.

WEB-INF/web.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app\_2\_3.dtd">

<web-app id=*"WebApp\_ID"*>

<servlet>

<servlet-name>dwr</servlet-name>

<servlet-class>

org.directwebremoting.spring.DwrSpringServlet

</servlet-class>

<init-param>

<param-name>debug</param-name>

<param-value>true</param-value>

</init-param>

<init-param>

<param-name>crossDomainSessionSecurity</param-name>

<param-value>false</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>dwr</servlet-name>

<url-pattern>/dwr/\*</url-pattern>

</servlet-mapping>

</web-app>

DWR 에서 사용할 서비스 및 데이터 타입을 설정하기 위하여 dwrSubsystemContext.xml 파일을 생성하고 webApplicaitonContext.xml 에 추가한다.

WEB-INF/context-config/dwrSubsystemContext.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:p="http://www.springframework.org/schema/p"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:util="http://www.springframework.org/schema/util"

xmlns:dwr="http://www.directwebremoting.org/schema/spring-dwr"

xsi:schemaLocation=*"*

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util.xsd

http://www.directwebremoting.org/schema/spring-dwr http://www.directwebremoting.org/schema/spring-dwr-2.0.xsd"

default-autowire=*"no"* default-init-method=*"init"*

default-destroy-method=*"destroy"*>

<dwr:configuration>

<dwr:convert class="java.lang.Exception" type="exception">

<dwr:include method=*"message"*/>

</dwr:convert>

<dwr:signatures><![CDATA[

import java.util.Map;

import java.util.List;

]]></dwr:signatures>

</dwr:configuration>

<bean id="dwrDataService" class="tests.service.DwrDataService" p:dataService-ref="dataService">

<dwr:remote javascript=*"dataservice"*>

<dwr:include method=*"queryForList"*/>

</dwr:remote>

</bean>

</beans>

|  |  |
| --- | --- |
|  |  |
| 웹 프로그램 개발하기 | |
|  | 스트럿츠 2 기반의 웹 응용프로그램 구현에 대하여 기술한다. |

1. MVC 디자인을 수용한 웹 디자인 패턴으로 Model 2 또는 MVC Model 2 라 불린다. [↑](#footnote-ref-1)
2. MVC 디자인 패턴은 어플리케이션을 Model, View, Controller으로 나누어 각 계층간의 결합력이 최소화 될 수 있도록 디자인하는 방식으로, 대부분의 웹 프레임워크들은 UI 와 비즈니스 로직을 분리하여 개발자와 디자이너 사이에 작업을 분리하고 컴포넌트 형태로 개발할 수 있도록 MVC 디자인 패턴을 지향하고 있다. 일반적인 웹 어플리케이션들에서, 도메인 모델들은 서버에 저장되며 사용자는 클라이언트에서 동작하는 소프트웨어 (보통 클라이언트 측에서 응답을 랜더링하는 브라우저)와 낮은 수준의 상호작용을 한다. 따라서 전형적인 웹 어플리케이션 개발에서, Model, View 그리고 Controller는 네트워크를 넘어서 서버와 클라이언트에 위치하게 된다. [↑](#footnote-ref-2)