08. 스크립트가 없는 JSP

학습 목표

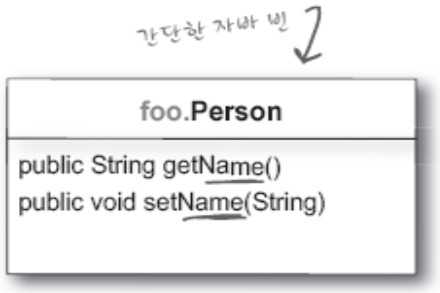
- 자바 빈 표준 액션 종류 및 사용법에 대해 알아본다
- EL 사용법에 대해 알아본다

1. 자바 빈 표준 액션

2. 자바 빈 표준 액션 종류

3. EL

JSP >> String 이 아닌 속성



서블릿 코드

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
                                        throws IOException, ServletException {

    foo.Person p = new foo.Person();
    p.setName("Evan");
    request.setAttribute("person", p);

    RequestDispatcher view = request.getRequestDispatcher("result.jsp");
    view.forward(request, response);
}
```

JSP 코드

```
<html><body>

<% foo.Person p = (foo.Person) request.getAttribute("person"); %>
Person is: <%= p.getName() %>

</body></html>
```

getName()의 리턴값을 출력하겠죠.

또는 표현식을 사용하면

```
<html><body>
Person is:
<%= ((foo.Person) request.getAttribute("person")).getName() %>

</body></html>
```

JSP >> 자바 빈 표준 액션

■ JSP에는 자바 빈과 관련한 표준 액션을 이용할 수 있다.

- 표준 액션을 이용하면 JSP에서 스크립팅 코드 없이 자바 빈을 사용할 수 있다.
- 자바 빈이 속성 값이 되어야 한다.

표준 액션을 사용하지 않고서(스크립팅으로)

지금까지 해왔던 방식이죠.

```
<html><body>

<% foo.Person p = (foo.Person) request.getAttribute("person"); %>
Person is: <%= p.getName() %>

</body></html>
```

표준 액션을 사용하면(스크립팅 사용 안 함)

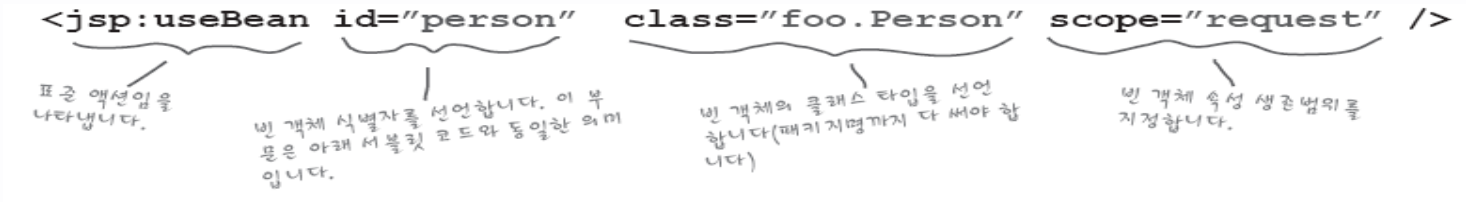
어! 자바 코드가 어디 갔죠. 스크립팅이 없습니다. 단지 두 개의 표준 액션 태그만으로 모든 걸 처리했습니다.

```
<html><body>
<jsp:useBean id="person"  class="foo.Person" scope="request" />
Person created by servlet: <jsp:getProperty name="person"  property="name" />
</body></html>
```

JSP >> 자바 빈 표준 액션

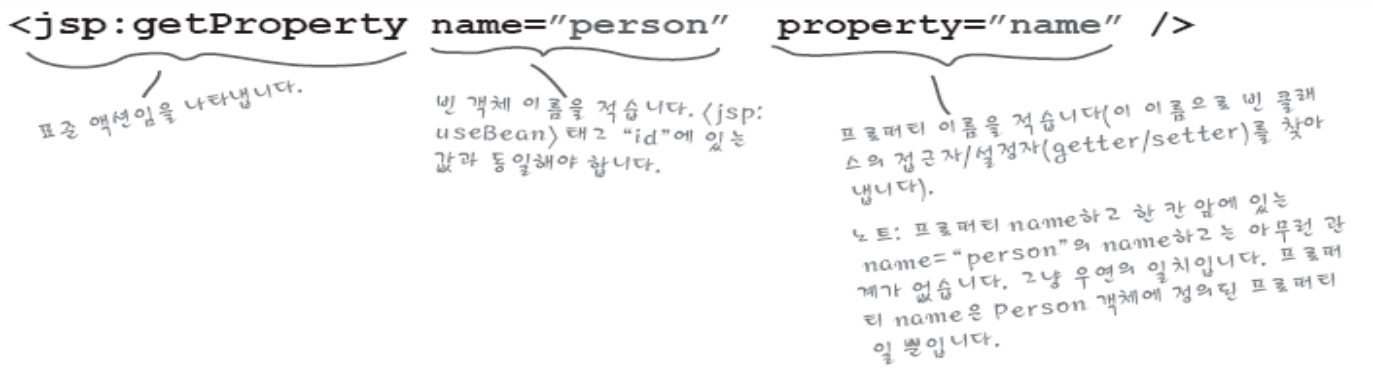
<jsp:useBean>

- 빈을 선언하고 초기화 하는 태그



<jsp:getProperty>

- 속성 빈 프로퍼티를 읽어오는 태그



JSP >> <jsp:useBean>

■ 객체 생성

- 찾는 객체가 없는 경우에는 객체를 생성하기도 한다.

아래 태그가

```
<jsp:useBean id="person"  class="foo.Person" scope="request" />
```

_jspService() 메소드 내에 다음과 같은 코드로 바뀝니다.

```
foo.Person person = null;
```

id에 있는 값을 가지고 변수를 선언합니다. 이제부터 이 변수를 JSP 나머지 부분에서도 참조할 수 있습니다. 물론 다른 빈 태그에서도 참조할 수 있죠.

```
synchronized (request) {
  person = (foo.Person)_jspx_page_context.getAttribute("person", PageContext.REQUEST_SCOPE);
  if (person == null){
     person = new foo.Person();
     _jspx_page_context.setAttribute("person", person, PageContext.REQUEST_SCOPE);
  }
}
```

태그에 정의된 생존범위(scope)에서 속성을 찾아봅니다. 결과를 person에 설정하겠죠.

해당 생존범위에 이런 이름을 가진 속성이 없다면...

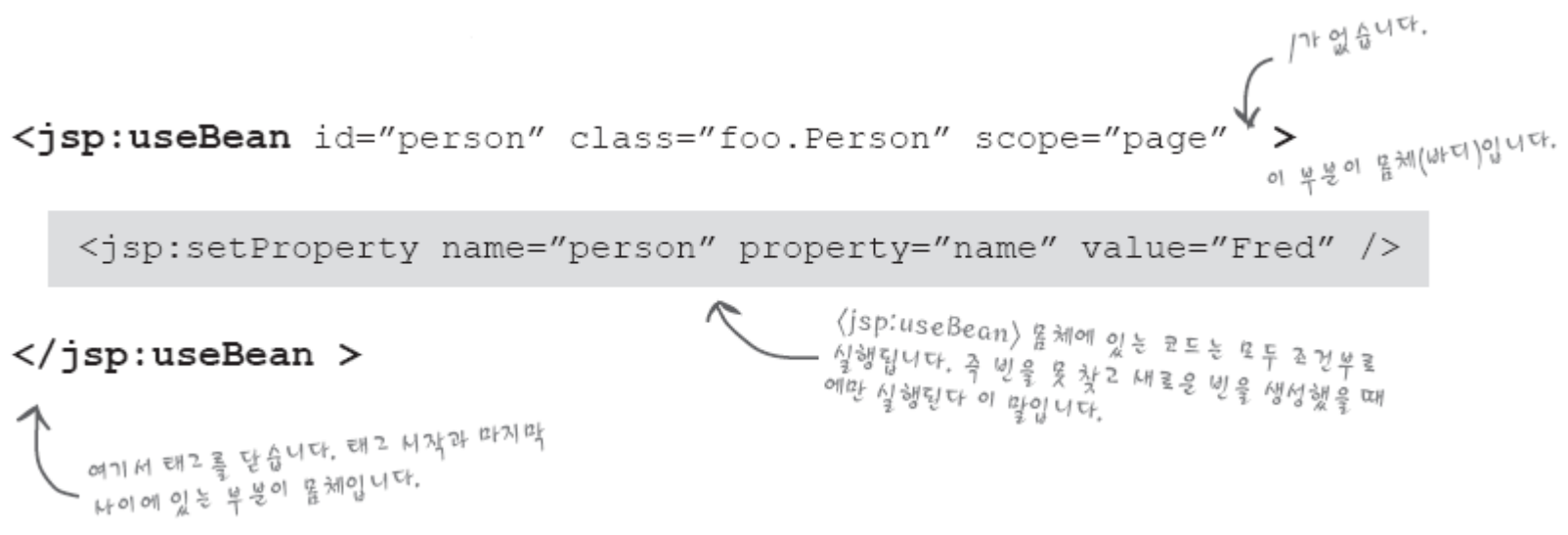
하나를 만들어서 person에 할당합니다.

마지막으로 방금 만든 객체를 태그에서 정의한 생존범위에 설정합니다.

JSP >> <jsp:useBean>

■ 객체 생성시 기본 값 설정

- **<jsp:useBean>** 태그 안에 **<jsp:setProperty>** 를 사용하면 조건부로 자바 빈의 프로퍼티 값을 설정한다.
- 이미 빈이 있는 경우에는 실행되지 않는다.



JSP >> <jsp:useBean>

■ 빈 참조 시 다형성 기법 사용 (1)

- **<jsp:useBean>** 에서 객체를 찾지 못한 경우 새로 생성할 클래스를 결정함.

JSP useBean 태그

```
<jsp:useBean id="person" class="foo.Person" scope="page" />
```

태그에 있는 class는 참조 및 객체의 타입을 동시에 표현합니다.

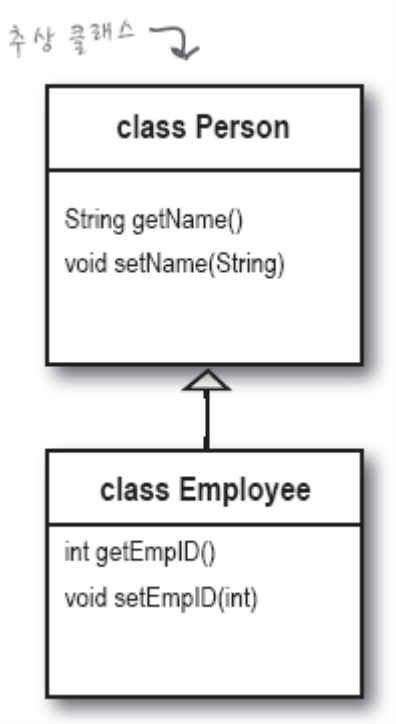
생성된 서블릿 코드

```
foo.Person person = null;
// Person의 속성을 읽어오는 코드
if (person == null){
person = new foo.Person();
...
```

JSP >> <jsp:useBean>

■ 빈 참조 시 다형성 기법 사용 (2)

● 객체를 참조할 참조 변수의 타입 정의 가능



type 속성으로 새로 작성한 JSP

```
<jsp:useBean id="person" type="foo.Person" class="foo.Employee"  scope="page">
```

생성된 서블릿 코드

```
foo.Person person = null;
// Person의 속성을 읽어오는 코드
if (person == null){
person = new foo.Employee();
...
```

이제 참조 타입은 추상 객체고 객체의 타입은 Employee가 됩니다.

JSP >> <jsp:useBean>

■ class 속성 없이 type 속성만 사용하는 경우

- 빈이 먼저 존재해야 한다.
- **type**이 있던 없든 관계없이 **class**가 있을 경우, 이 클래스는 추상 객체여서는 안되며, 반드시 인자가 없는 **public** 생성자가 있어야 한다.

JSP

class는 없고 type만 있습니다.

```
<jsp:useBean id="person" type="foo.Person" scope="page"/>
```

속성 Person이 생존범위 page에 존재하는 경우

문제없이 작동합니다.

속성 Person이 생존범위 page에 존재하지 않을 경우

오류가 발생합니다.

```
java.lang.InstantiationException: bean person not found within scope
```

JSP >> <jsp:useBean>

param 속성

- 요청 파라미터값을 빈 프로퍼티에 설정

TestBean.jsp 코드

```
<jsp:useBean id="person" type="foo.Person" class="foo.Employee">
    <jsp:setProperty name="person" property="name" param="userName" />
</jsp:useBean>
```

```
<html><body>

<form action="TestBean.jsp">
  name: <input type="text" name="userName">
  ID#: <input type="text" name="userID">
  <input type="submit">
 </form>

</body></html>
```

param 속성값은 폼 입력 필드 name 속성값에서 온 것입니다.

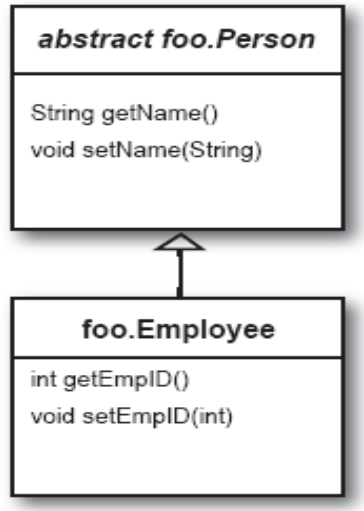
JSP >> <jsp:useBean>

■ param 속성을 사용하지 않아도 되는 경우

- 요청 파라미터와 빈 프로퍼티의 이름이 동일한 경우

HTML 폼 코드를 수정해서 입력 필드 이름하고 프로퍼티 이름하고 동일하게 만듭니다:

```
<html><body>

<form action="TestBean.jsp">
  name: <input type="text" name="name">
  ID#: <input type="text" name="userID">
  <input type="submit">
 </form>

</body></html>
```



이제 입력 필드 파라미터 이름과 빈 프로퍼티 이름이 똑같아졌습니다.

이제 JSP 코드를 아래와 같이 수정합니다.

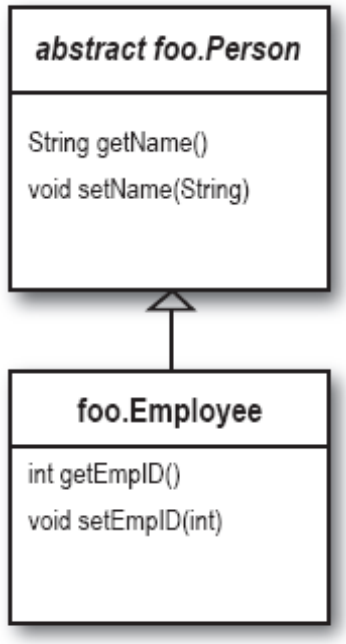
```
<jsp:useBean id="person" type="foo.Person" class="foo.Employee">
    <jsp:setProperty name="person" property="name" />
</jsp:useBean>
```

여기에 아무 것도 코딩하지 않았습니다.

JSP >> <jsp:useBean>

■ 기본형 변환

- 빈 태그는 기본 타입 프로퍼티를 자동으로 변환
- **String** 이나 기본형**(Primitive Type)**인 경우, 컨테이너에서 자동 변환



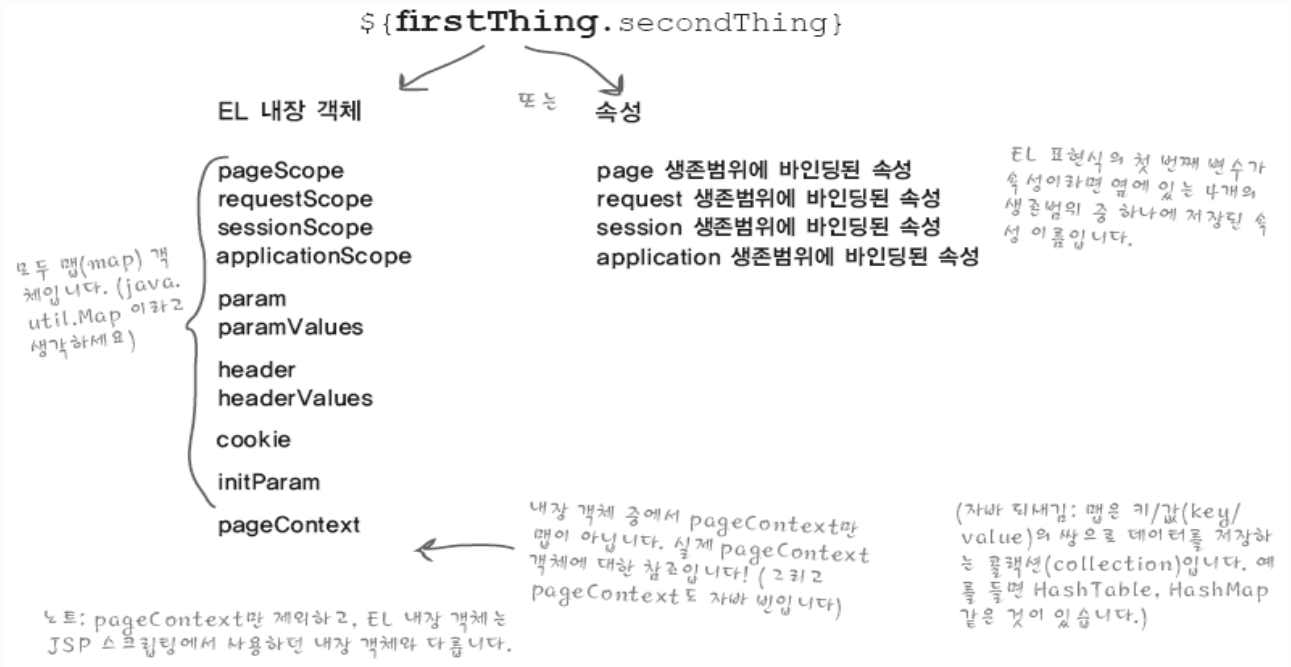
type이 Employee인 경우
(Person이 아니라)

```
<html><body>

    <jsp:useBean id="person" type="foo.Employee" class="foo.Employee" >
        <jsp:setProperty name="person" property="*"  />
    </jsp:useBean>

    Person is: <jsp:getProperty name="person" property="name" />
    ID is: <jsp:getProperty name="person" property="empID" />

</body></html>
```

생성될 서블릿에 다음과 같은 코드가 들어가겠죠:
Person person = new Employee();가 아닌
Employee person = new Employee();

JSP >> EL (Expression Language)

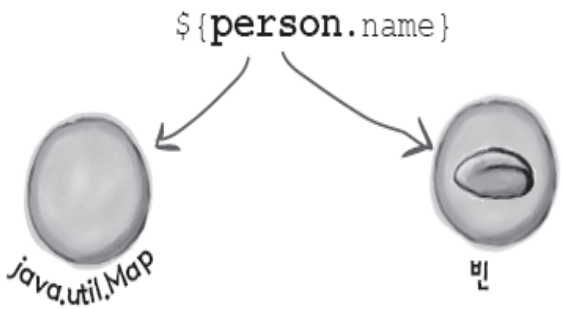
- JSP 스펙 2.0 에 추가
- 빈의 프로퍼티의 프로퍼티가 객체인 경우 사용 가능
- 기본 형태
  - 항상 중괄호({})로 묶고, 제일 앞에 달러 기호($)를 붙인다



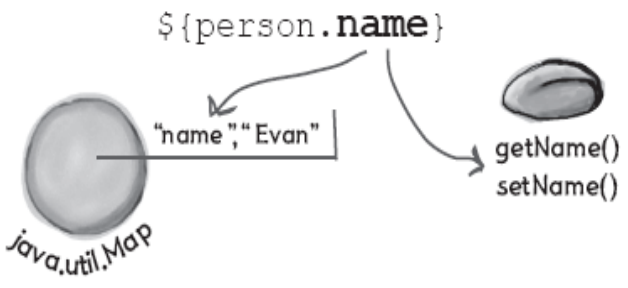
JSP >> EL (Expression Language)

■ 도트(.) 연산자

① 표현식에서 도트 연산자 왼쪽은 반드시 맵 또는 빈이여야 합니다.



② 표현식에서 도트 연산자 오른쪽은 반드시 맵의 키이거나 빈 프로퍼티여야 합니다.



③ 오른쪽에 오는 값은 식별자로서 일반적인 자바 명명 규칙을 따라야 합니다.

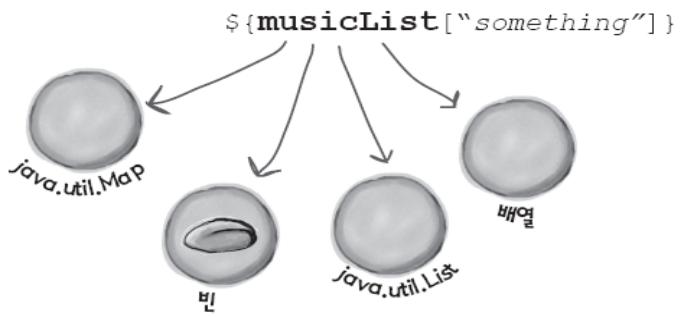
```
${person.name}
```

* 문자, _, $로 시작해야 합니다.
* 두 번째 글자부터 숫자를 써도 무방합니다.
* 자바 예약어(키워드)는 사용할 수 없습니다.

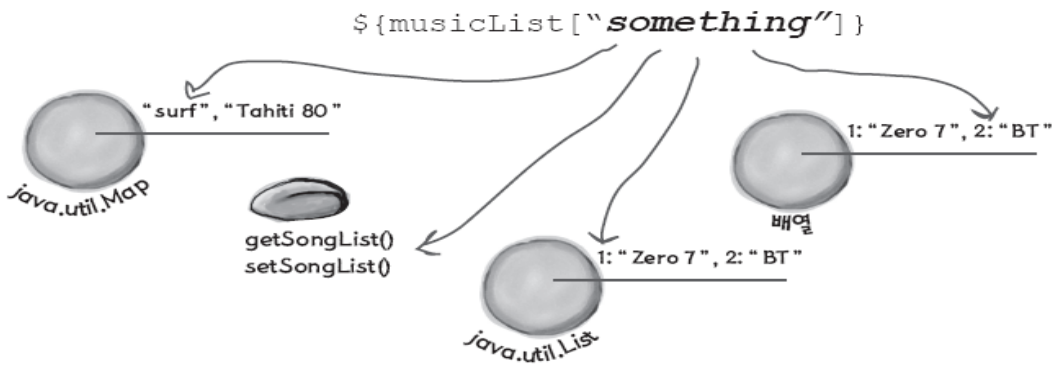
JSP >> EL (Expression Language)

■ [ ] 연산자

① [ ] 연산자의 왼편에는 맵, 빈, 배열, 리스트 변수가 올 수 있습니다.



② [ ] 연산자 안의 값이 문자열(따옴표로 묶여 있다면)이라면, 이것은 맵 키가 될 수 있고, 빈 프로퍼티 또는 리스트나 배열 인덱스가 될 수 있습니다.



JSP >> EL (Expression Language)

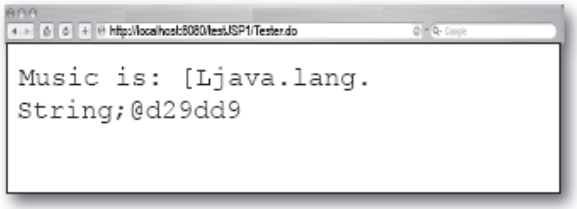
■ [ ] 연산자 : 배열에 사용

서블릿 코드

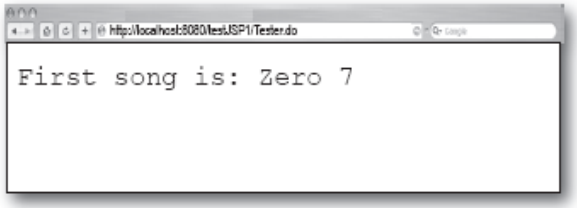
```
String[] favoriteMusic = {"Zero 7", "Tahiti 80", "BT", "Frou Frou"};
request.setAttribute("musicList", favoriteMusic);
```

JSP 코드

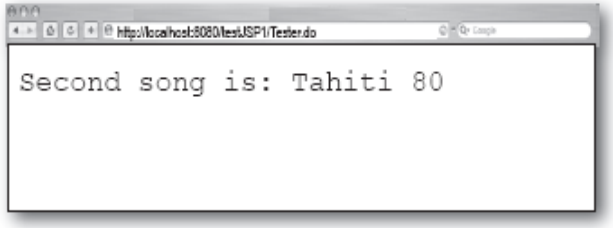
```
Music is:  ${musicList}
```




```
First song is: ${musicList[0]}
```




```
Second song is: ${musicList["1"]}
```



JSP >> EL (Expression Language)

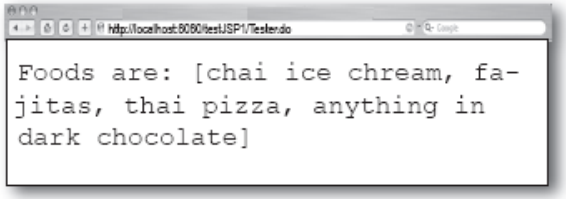
[ ] 연산자 : 배열과 리스트인 경우 문자로 된 인덱스값은 숫자로 변환

서블릿 코드

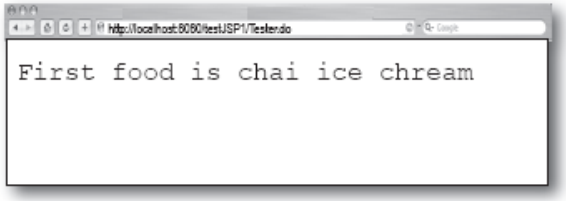
```
java.util.ArrayList favoriteFood = new java.util.ArrayList();
favoriteFood.add("chai ice chream");
favoriteFood.add("fajitas");
favoriteFood.add("thai pizza");
favoriteFood.add("anything in dark chocolate");
request.setAttribute("favoriteFood", favoriteFood);
```

JSP 코드

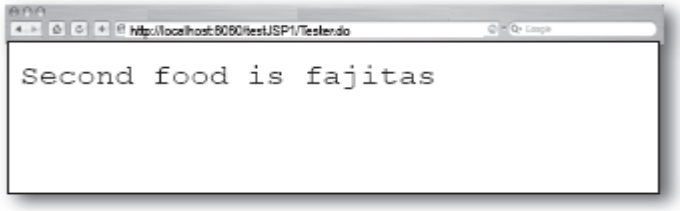
```
Foods are: ${favoriteFood}
```




```
First food is ${favoriteFood[0]}
```




```
Second food is ${favoriteFood["1"]}
```



JSP >> EL (Expression Language)

■ 빈과 맵 : 도트(.) 연산자, [ ] 연산자 둘 다 사용 가능

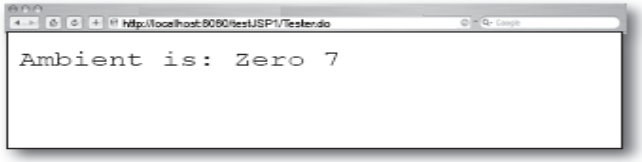
서블릿 코드

```
java.util.Map musicMap = new java.util.HashMap();
musicMap.put("Ambient", "Zero 7");
musicMap.put("Surf", "Tahiti 80");
musicMap.put("DJ", "BT");
musicMap.put("Indie", "Travis");
request.setAttribute("musicMap", musicMap);
```

맵을 만든 다음, String인 키와 객체를 설정합니다. 마지막으로 이를 request 속성에 넣어 두는 거죠.

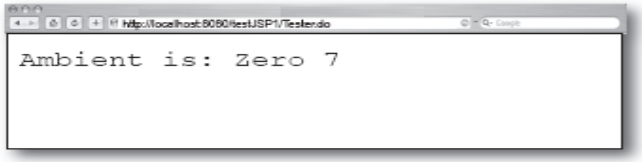
JSP 코드

```
Ambient is: ${musicMap.Ambient}
```



두 표현식 모두 "Ambient"를 맵 키로 처리합니다(musicMap은 맵입니다!)

```
Ambient is: ${musicMap["Ambient"]}
```



JSP >> EL (Expression Language)

■ 내장 객체

pageScope
requestScope
sessionScope
applicationScope

생존범위 속성 맵

param
paramValues

요청 파라미터 맵

header
headerValues

요청 헤더 맵

cookie

오~우, 꽁 어려운 놈이군요. 이것도 쿠키 맵인가요?

initParam

컨텍스트 초기화 파라미터 맵(서블릿 초기화 파라미터 아님)

pageContext

맵이 아닌 유일한 녀석. 얘만 실제 pageContext 객체에 대한 참조입니다. 그리고 이건 빈입니다. API 문서에서 pageContext 접근자(Getter)가 어떤 것이 있는지 한번 보세요.*

JSP >> EL (Expression Language)

■ param 과 paramValues : 요청 파라미터 처리

HTML 폼 코드

```
<form action="TestBean.jsp">
   Name: <input type="text" name="name">
   ID#: <input type="text" name="empID">

   First food: <input type="text" name="food">
   Second food: <input type="text" name="food">

   <input type="submit">
</form>
```

"name"과 "empID"는 값이 하나 밖에 없습니다. 하지만 "food" 파라미터의 경우 사용자가 두 개의 필드에 모두 값을 채운 다음 Submit 버튼을 클릭하면 파라미터 값은 두 개입니다.

JSP 코드

앞에서 봤듯이 param은 파라미터 이름/값이 저장된 맵 객체입니다. 따라서 도트 연산자 오른쪽에 폼 입력 필드의 이름이 와야 합니다.

```
Request param name is: ${param.name} <br>

Request param empID is: ${param.empID} <br>

Request param food is: ${param.food} <br>

First food request param: ${paramValues.food[0]} <br>
Second food request param: ${paramValues.food[1]} <br>

Request param name: ${paramValues.name[0]}
```

"food"는 값이 두 개지만 param 내장 객체를 사용할 수 있습니다. 물론 첫 번째 값만 출력되겠죠.

JSP >> EL (Expression Language)

■ header : Request 의 Header 정보 처리

"host" 헤더 정보 읽기

스크립팅으로 코딩하면

```
Host is: <%= request.getHeader("host") %>
```

EL 내장 객체 header를 사용해서 코딩하면

```
Host is: ${header["host"]}

Host is: ${header.host}
```

header EL 내장 객체도 모든 헤더 정보를 가지고 있는 맵입니다. 도트 연산자를 사용하든 아니면 [] 연산자를 사용하던 헤더 이름을 넘기면 헤더 정보를 얻을 수 있지요(노트: 헤더 값이 하나 이상일 경우 EL 내장 객체 headerValues를 사용하세요. 사용법은 paramValues와 똑같습니다).

JSP >> EL (Expression Language)

cookie

쿠키 "userName"의 값을 출력해봅시다

스크립팅으로 이를 짜보면

```
<% Cookie[] cookies = request.getCookies();

for (int i = 0; i < cookies.length; i++) {
   if ((cookies[i].getName()).equals("userName")) {
      out.println(cookies[i].getValue());
   }
} %>
```

스크립팅으로 짜면 프로그램이 좀 길죠. request 객체에 getCookie(cookieName) 이런 메소드가 있었으면 좋았을텐데. Cookie 배열을 읽어와 원하는 것을 찾을 때까지 루핑을 돌릴 수 밖에...

EL에서는 Cookie 내장 객체를 사용하면 되죠

```
${cookie.userName.valu}
```

이야~ 이렇게 쓰을 수가. 그냥 이름만 주면, Cookie 맵에서 값이 그냥 나오는군.

JSP >> EL (Expression Language)

initParam

컨텍스트 초기화 파라미터값을 출력해봅시다

DD에 다음과 같은 파라미터가 설정되어 있다고 한다면

컨텍스트 파라미터(애플리케이션 전체에 공유가 되는)를 설정하는 것을 눈여겨 보세요. 서블릿 초기화 파라미터하고 같지 않죠.

```
<context-param>
    <param-name>mainEmail</param-name>
    <param-value>likewecare@wickedlysmart.com</param-value>
 </context-param>
```

스크립팅으로 이를 짜보면

```
email is: <%= application.getInitParameter("mainEmail") %>
```

EL로 하면 더 쉽죠

```
email is: ${initParam.mainEmail}
```

JSP >> EL (Expression Language)

■ 함수 사용 : 작성 순서

- 정적인 공용 메소드를 제공하는 클래스를 작성한다

  /WEB-INF/classes 디렉토리에 배포한다.

- 태그 라이브러리 서술자 파일을 만든다

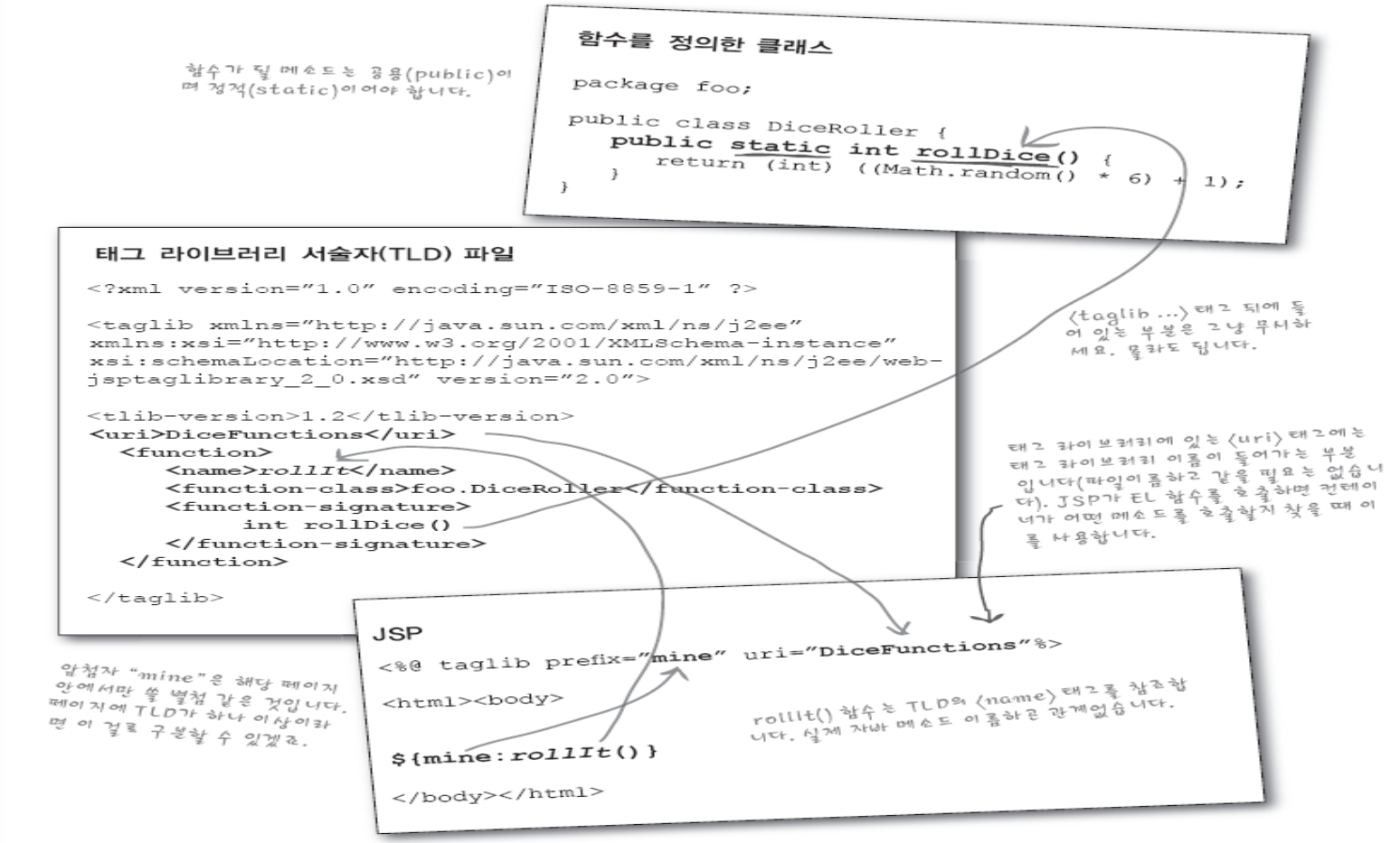
  TLD(Tag Library Descriptor) 파일에 함수를 정의한 자바 클래스와 이를 호출할 **JSP**를 매핑한다.

- **JSP**에 **taglib** 지시자를 코딩한다.

- 함수를 호출하는 **EL** 을 작성한다.

JSP >> EL (Expression Language)

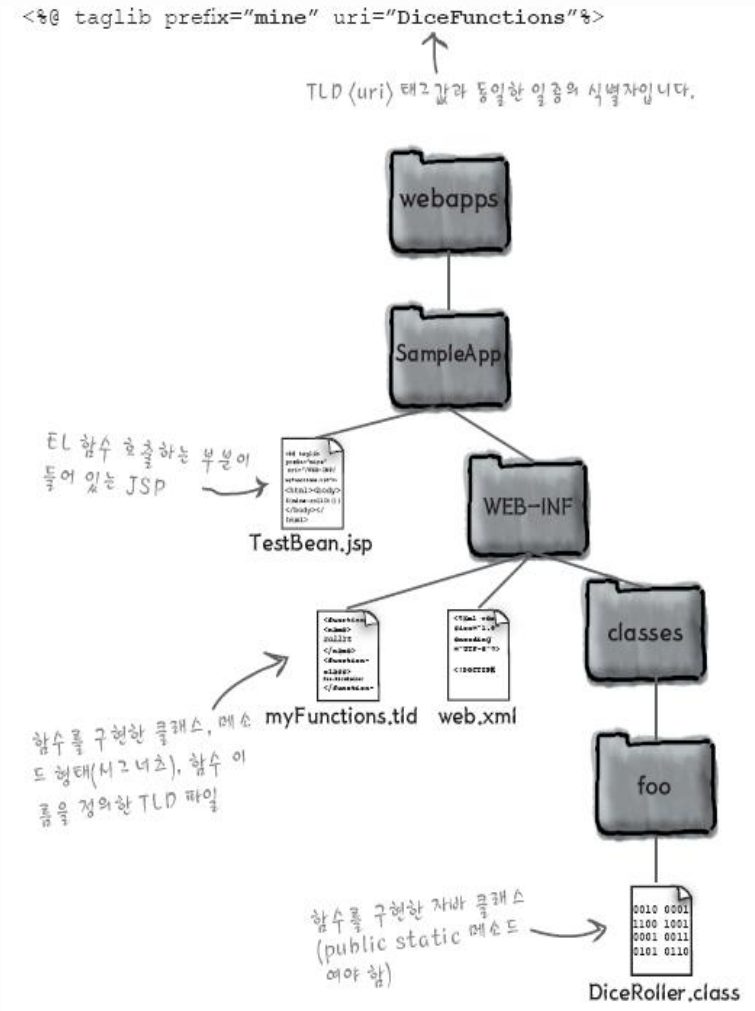
■ 함수 사용 : 함수 클래스, TLD, JSP



JSP >> EL (Expression Language)

■ 함수 사용 : 배포

- 함수를 구현한 클래스는 서블릿, 리스너, 빈 클래스에서 접근 할 수 있도록 **WEB-INF/classes** 에 위치해야 한다.
- **TLD** 파일도 **WEB-INF** 밑에 두도록 한다



JSP >> EL (Expression Language)

EL 연산자

산술 연산자 (5)

더하기:	+
빼기:	-
곱하기:	*
나누기:	/와 div
나머지:	%와 mod

0으로 나눌 수 있습니다. 오류가 아닌 무한대(infinity)가 나오겠죠.

0에 대해서 이 연산자를 쓸 수 없습니다. 오류가 떨어집니다.

논리 연산자 (3)

AND:	&&와 and
OR:	||와 or
NOT:	!와 not

관계 연산자 (6)

등호:	==와 eq
부등호:	!=와 ne
~보다 작다:	<와 lt
~보다 크다:	>와 gt
~보다 작거나 같다:	<=와 le
~보다 크거나 같다:	>=와 ge

JSP >> EL (Expression Language)

■ 널(NULL) 처리

EL	출력값
${foo}	
${foo[bar]}	
${bar[foo]}	
${foo.bar}	
${7 + foo}	7
${7 / foo}	무한대
${7 - foo}	7
${7 % foo}	예외가 떨어집니다

출력값이 없습니다. 예를 들어 "출력값은: ${foo} 입니다"라고 코딩하면, "출력값은: 입니다"라고 나오겠죠

EL 산술 연산에서는 널(null)을 0으로 취급합니다.

EL	출력값
${7 < foo}	false
${7 == foo}	false
${foo == foo}	true
${7 != foo}	true
${true and foo}	false
${true or foo}	true
${not foo}	true

EL 논리 연산에서는 정의되지 않은 값은 "거짓(false)"으로 처리합니다

JSP >> <jsp:include>

<jsp:include> 사용

표준 머리말 파일("header.jsp")

```
<html><body>

<img src="images/Web-Services.jpg" > <br>
<em><strong>We know how to make SOAP suck less.</strong></em> <br>

</body></html>
```

이 부분이 모든 페이지 앞부분에 나와야 합니다.

웹 애플리케이션에 있는 JSP 하나("Contact.jsp")

```
<html><body>

<jsp:include page="Header.jsp" />

<br>
<em>We can help.</em> <br><br>
Contact us at: ${initParam.mainEmail}
</body></html>
```

옆에 있는 코딩을 번역해보면 "현재 이 페이지 바로 여기에 Header.jsp 파일 전부를 그대로 포함하세요. 나머진 이 페이지에 있는 대로 보여주면 됩니다"

JSP >> <jsp:include>

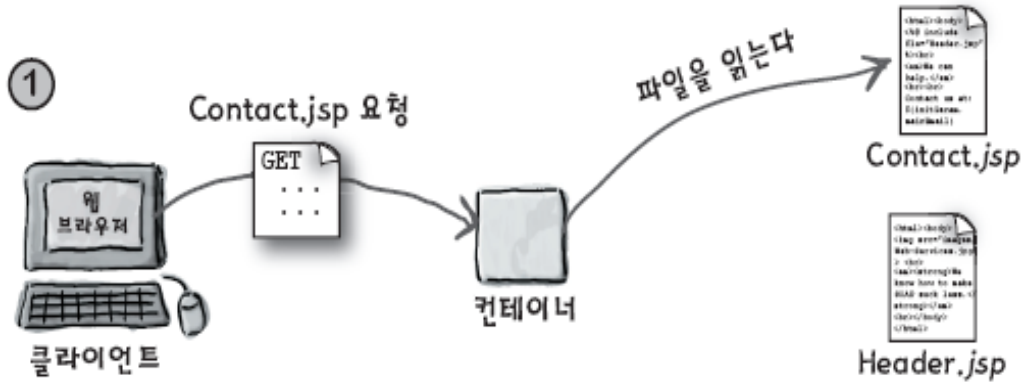
<jsp:include> 와 <%@ include ...%>

- include 지시자(<%@ include ... %>) 는 JSP 페이지의 서블릿 변환 시에 실행이 되는 것으로 단순 파일 내용을 붙이는 작업이다.

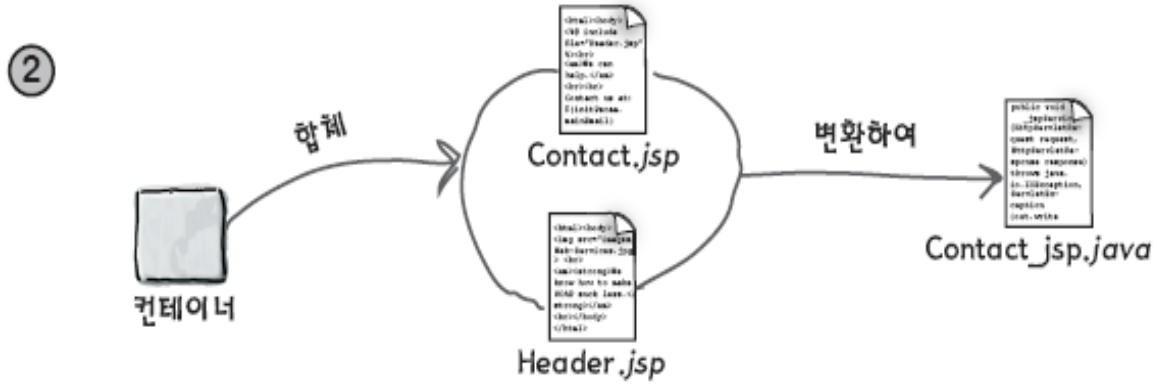
- include 표준 액션(<jsp:include>) 는 실행 시에 include 되는 페이지의 응답(response)를 포함하는 형식이다.

JSP >> <jsp:include>

■ include 지시자 (<%@ include ...%>) 실행 순서 (1)



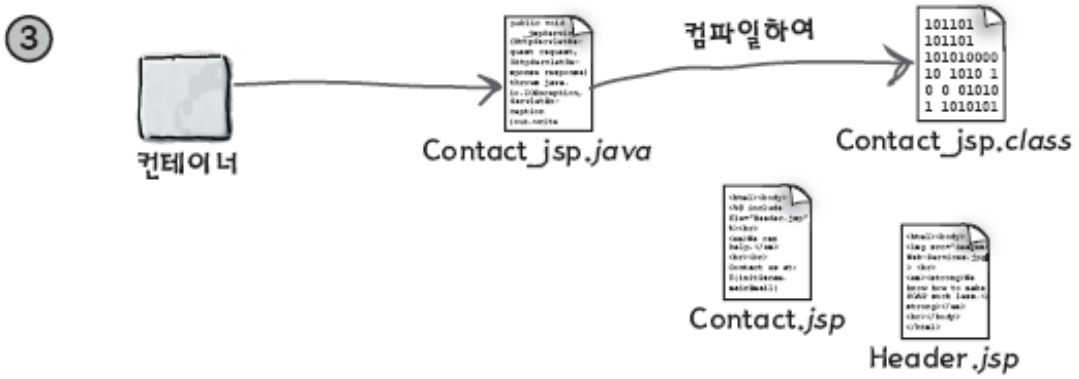
클라이언트가 아직 변환이 이루어지지 않은 Contact.jsp를 요청합니다. 컨테이너는 Contact.jsp 파일을 읽은 다음 변환 작업을 시작합니다.



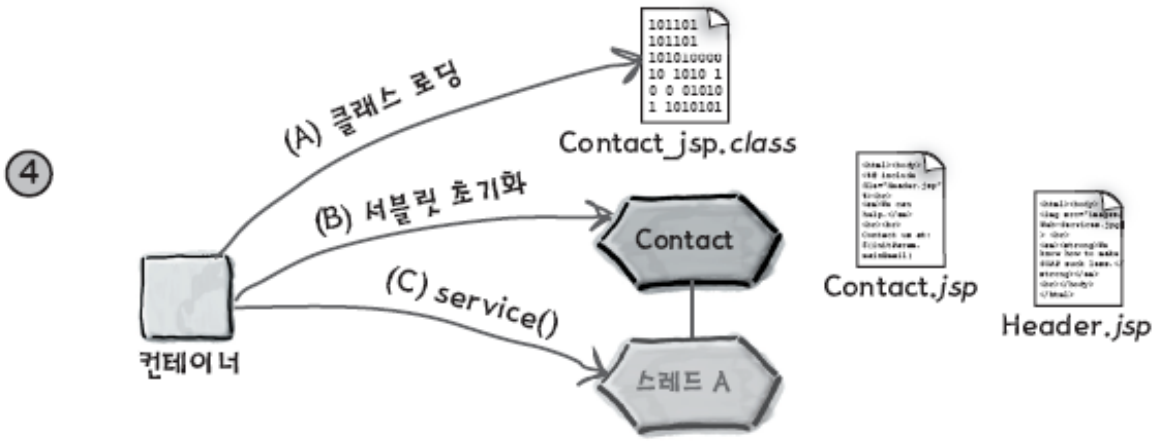
컨테이너가 읽어 들인 파일에서 include 지시자를 만나면 소스인 Header.jsp를 여기에 합쳐 새로운 파일을 만들고 이를 변환하여 자바 소스 파일을 생성합니다.

JSP >> <jsp:include>

■ include 지시자 (<%@ include ...%>) 실행 순서 (2)



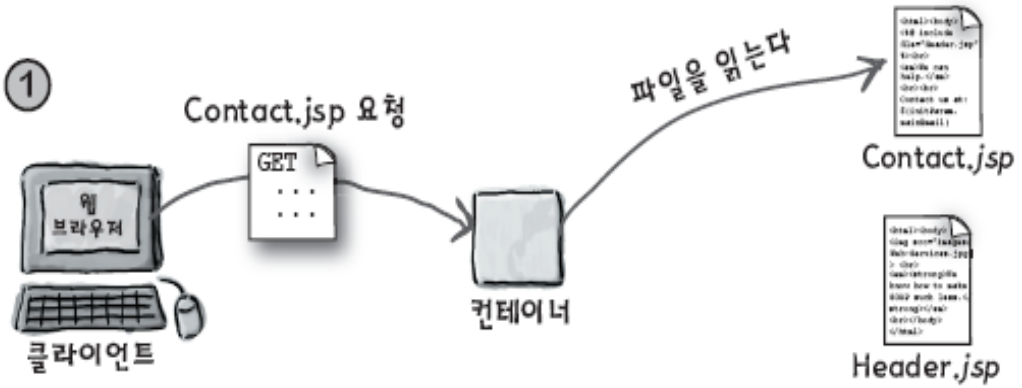
그 다음 이를 컴파일하여 서블릿 클래스 파일을 만듭니다. 이 작업은 여타 JSP와 차이가 없습니다. 앞 단계 작업은 Contact.jsp 파일이 바뀌지 않는 한 다시 일어나지 않습니다. (아주 똑똑한 컨테이너라서 Header.jsp가 변경된 걸 알고 자동으로 이 작업을 하지 않는 한 말입니다)



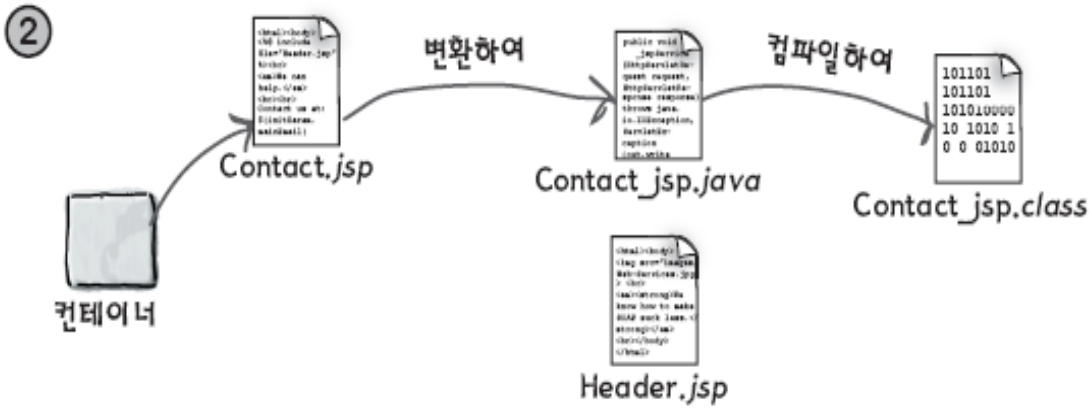
요청을 완료하기 위해, 컨테이너는 바로 컴파일된 클래스를 로딩한 뒤, 서블릿을 초기화합니다. (초기화 작업은 서블릿의 init() 메소드 호출로 이루어집니다) 다음 스레드를 새로 하나 할당하여 _jspService() 메소드를 부릅니다. 두 번째 요청부터 컨테이너는 스텝 (C)만 호출합니다. (스레드를 새로 하나 할당하여 _jspService() 메소드를 부릅니다)

JSP >> <jsp:include>

include 표준 액션 (<jsp:include>) 실행 순서 (1)



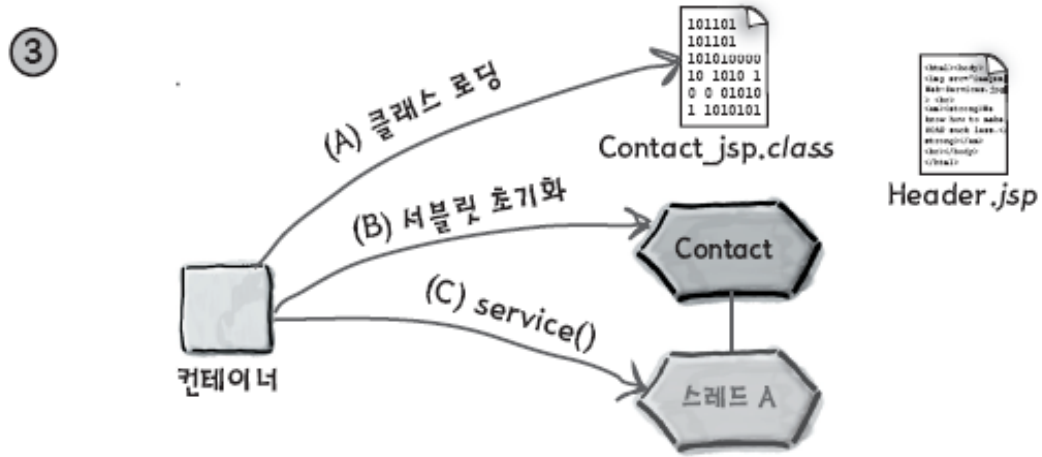
클라이언트가 아직 변환이 이루어지지 않은 Contact.jsp를 요청합니다. 컨테이너는 Contact.jsp 파일을 읽은 다음 변환 작업을 시작합니다.



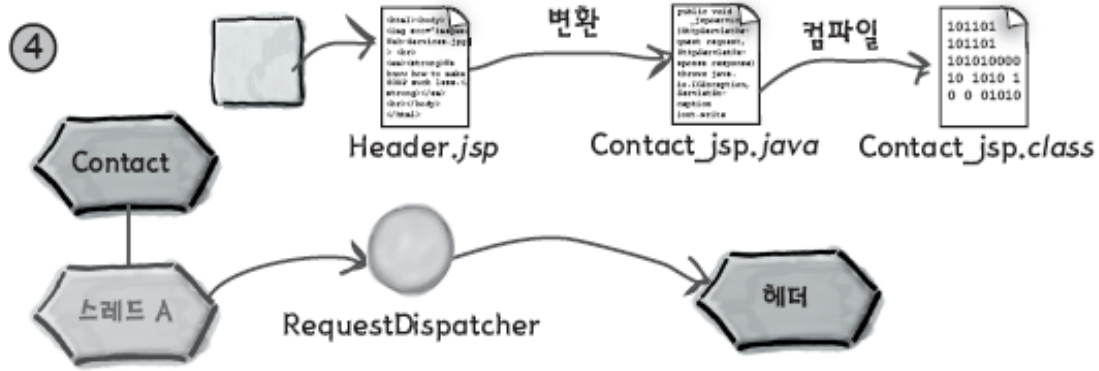
컨테이너가 읽어 들인 파일에서 〈jsp:include〉 표준 액션을 만나면 메소드 호출을 생성할 서블릿 코드에 자동으로 삽입하는데, 어떤 메소드냐하면, 실행(런타임) 시 Header.jsp의 응답을 동적으로 Contact.jsp에 합체할 메소드입니다. 컨테이너가 각 JSP 파일을 위한 서블릿 코드를 따로 만듭니다(사실 스펙에는 이 부분에 대하여 언급이 없습니다만 이렇게 작동해야 되지 않겠냐는 관점에서 설명해봅니다).

JSP >> <jsp:include>

include 표준 액션 (<jsp:include>) 실행 순서 (2)



컨테이너는 변환된 소스 파일을 컴파일하여 서블릿 클래스를 만듭니다. 이 작업은 여타 JSP와 차이가 없습니다. 생성된 서블릿 클래스를 가상 머신(JVM)으로 로딩한 다음, 초기화합니다. 그 다음 스레드를 새로 하나 할당하여 _jspService() 메소드를 호출합니다.



Contact 서블릿은 동적인 include(이 부분의 실제 구현은 벤더마다 다릅니다)를 실행합니다. 중요한 것은 Header 서블릿의 실행 결과인 응답을 Contact 서블릿 응답의 적절한 지점에 합체한다는 것입니다. (그림에는 없는 부분: Header.jsp를 변환, 컴파일하여 서블릿 클래스로 만들고, 로딩하고, 초기화하는 부분)

JSP >> <jsp:param>

■ 다른 페이지에 파라미터 전달 시 사용

메인 JSP

```
<html><body>

<jsp:include page="Header.jspf" >
   <jsp:param name="subTitle" value="We take the sting out of SOAP." />
</jsp:include>

<br>
<em>Web Services Support Group.</em> <br><br>
Contact us at: ${initParam.mainEmail}
</body></html>
```

마침을 알리는 /가 없는 것 보이죠.

<jsp:include> 태그는 몸체(<jsp:include>와 </jsp:include> 사이에 있는 것이 몸체입니다)를 가질 수 있습니다. 따라서 포함될 페이지에서 사용할 요청 파라미터를 추가(또는 대체)할 수 있습니다.

새로 만든 파라미터를 사용할 머리말 파일("Header.jspf")

```
<img src="images/Web-Services.jpg" > <br>
<em><strong>${param.subTitle}</strong></em> <br>
```

포함된 파일 안에서 <jsp:param>으로 추가된 파라미터는 다른 요청 파라미터와 똑같습니다. 여기서는 EL을 사용해서 읽어오는군요.

노트: include 지시자에는 파라미터 관련 부분이 없습니다. 동적인 포함이 아니지 않습니까. 따라서 <jsp:include> 표준 액션에만 적용되는 내용입니다.

JSP >> <jsp:forward>

■ JSP 에서 다른 JSP로 이동 시 사용

조건부 포워딩할 JSP(Hello.jsp)

```
<html><body>
Welcome to our page!

<% if (request.getParameter("userName") == null) { %>

    <jsp:forward page="HandleIt.jsp" />

<% } %>

Hello ${param.userName}

</body></html>
```

파라미터가 널인지 체크합니다.

파라미터가 널이면, 요청을 page 속성에 명기한 곳으로 넘깁니다(RequestDispatcher 사용하고 비슷합니다).

userName이 들어 있을 경우 실행하겠죠. 요청이 다른 곳으로 넘어 간다면 이 페이지에서 응답에 출력하는 것은 없습니다.

요청을 받을 JSP(HandeIt.jsp)

```
<html><body>
We're sorry... you need to log in again.

<form action="Hello.jsp" method="get">
Name: <input name="userName" type="text">
<input name="Submit" type="submit">
</form>

</body></html>
```

사용자로부터 요청 파라미터로 사용자 이름을 입력받아 위에 있는 Hello.jsp로 요청을 보내는 일반적이 페이지입니다.