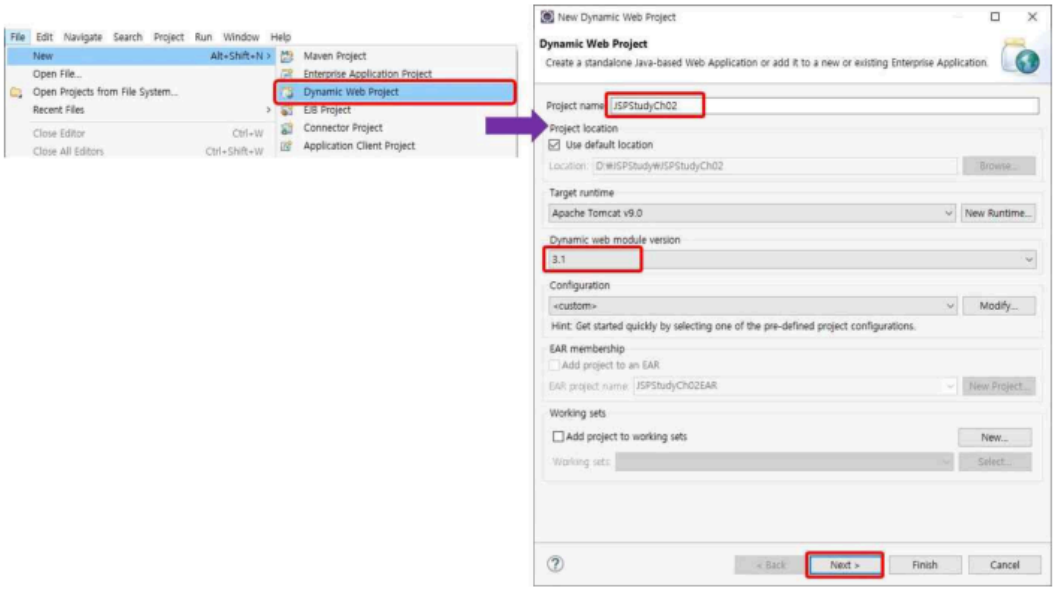
[02-01] JSP Day-3

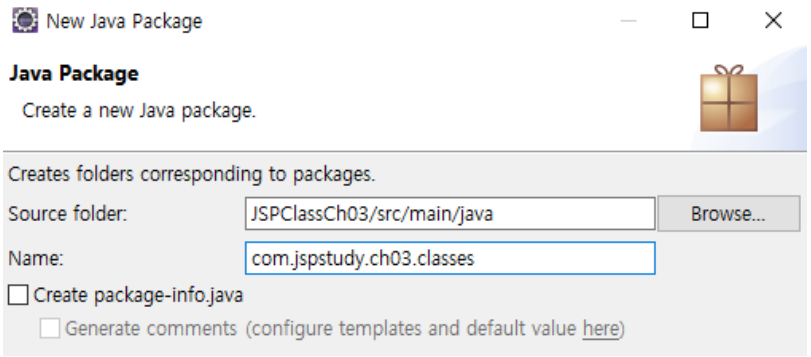
1. 새프로젝트 생성 : 웹서버(서블릿) 생성을 의미 (.java 파일) , 이 파일과 JSP파일이 연동된다.

(1) 새프로젝트 생성 : File > New > Dynamic Web Project > Project name(JSPClassCh02), Dynamic web module version(3.1) > next > next > web.xml은 배포파일을 의미, 체크박스 체크 > Finish

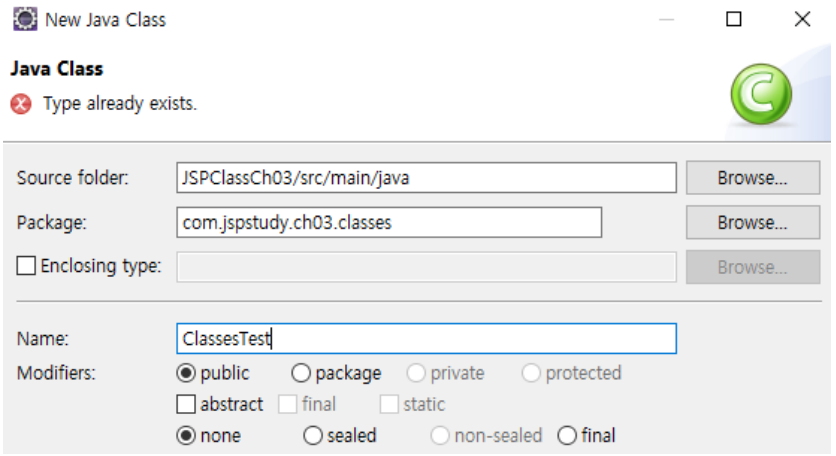


(2) 패키지 생성 : Java Resource > src/main/java > new > package > com.jspstudy.ch03.classes

- 도메인 네임이 중복되지 않도록, 패키지 네임은 거꾸로 쓰면 인식하기 유용하다



(3) 자바 파일 생성 : 생성된 패키지파일 우클릭 > new > class > name : ClassesTest > Finish



2. JSP 파일 생성 : .jsp 파일로 웹서버에 요청 및 응답을 주고 받는 파일

JSP 파일 생성은 webapp > new > jsp 로 만들거나 webapp > new > other > web > jsp 경로도 있다.

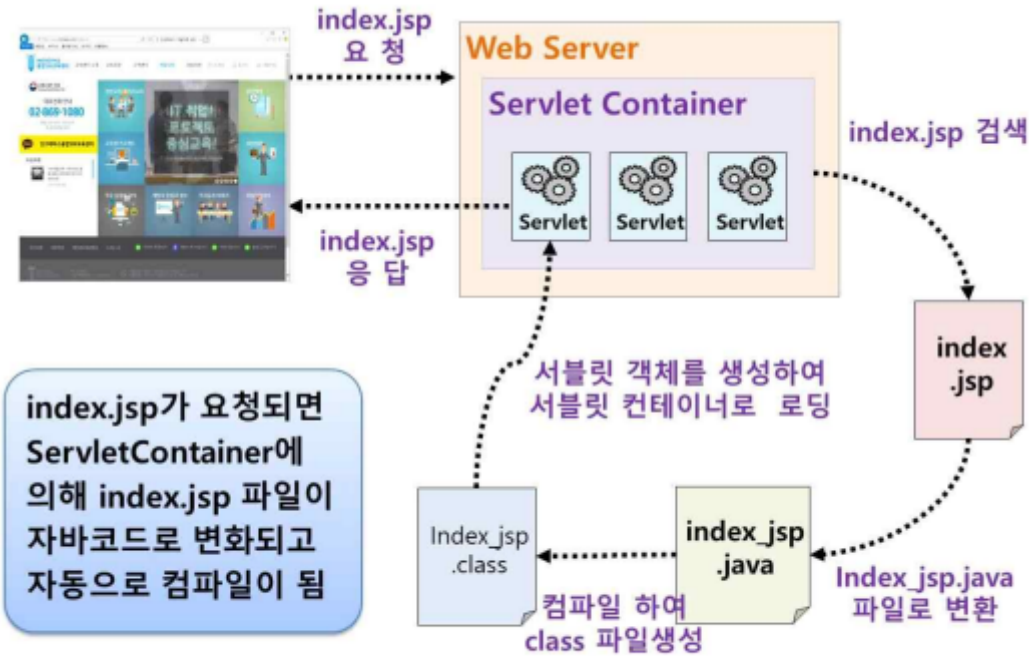


그림 3-1 JSP의 동작원리

클라이언트가 웹 브라우저를 통해 웹 서버에 요청을 전달하면 일반적인 HTML 문서와 같은 정적인 문서는 웹 서버에 의해 그대로 웹 브라우저로 전송된다. 하지만 JSP 페이지는 그림 3-1과 같이 웹 서버의 요청을 받은 서블릿 컨테이너(Servlet Container, 웹 컨테이너라고도 함)에서 실행되어 그 결과만 HTML 문서 형태로 변환된 후 웹 서버를 통해 브라우저에게 전송된다. 요즘과 같이 복합적이고 다양한 요청을 처리해야 하는 웹 애플리케이션에서는 단순히 HTML 코드로 구성된 페이지는 찾아보기 힘들고 HTML 태그와 프로그래밍 요소가 하나로 구성된 JSP와 같은 페이지가 대부분이다. 이렇게 정적요소를 포함한 여러 가지 프로그래밍 요소로 작성되는 JSP 페이지는 클라이언트의 요청이 웹 서버로 전달되면 웹 서버는 다시 클라이언트의 요청을 서블릿 컨테이너로 전달하여 서블릿 컨테이너에게 처리를 요청하게 된다. 웹 서버로부터 요청을 받은 서블릿 컨테이너는 해당 JSP 페이지를 서블릿 클래스(자바 클래스의 일종)의 소스 파일로 변환하고 변환된 Java 소스 파일을 컴파일한 후 서블릿 객체를 생성한다. 서블릿 컨테이너에 의해 생성된 서블릿 객체는 서블릿 컨테이너에 적재되어 클라이언트의 요청을 처리하고 그 결과를 HTML 문서로 변환하여 서블릿 컨테이너를 통해 웹 서버에 전달하게 된다. 그리고 웹 서버는 서블릿 컨테이너로부터 전달받은 HTML 문서를 클라이언트의 요청에 대한 응답으로 브라우저로 전송한다. 이후 동일한 요청에 대한 처리는 JSP 페이지가 변경되기 전까지 별도의 소스 파일 변환이나 컴파일 작업 없이 서블릿 컨테이너에 적재된 서블릿이 처리하게 된다.

- Web Server 에 요청을 보내,
- Servlet Container 내의 여러 Servlet 들이 담당하는 내용과 일치(감색)되면
- .jsp 를 .java 로 변환, class 파일 생성의 과정을 거치고
- 요청자의 응답을 완료

3. JSP 의 스크립팅

JSP는 동적처리를 지원하기위해 주로 문서 초반부에 스크립틀릿과 표현식, 변수, 선언부를 작성한다.
JSP의 스크립팅 3요소 : Scriptlet(스크립틀릿), Expression(표현식), Declaration(선언문)

1) Scriptlet(스크립틀릿) : <% %> 자바코드를 작성하는 곳이며 여기서 선언된 변수는 HTML 본문에서나 사용가능하다. 또한 HTML 어디서든 위치할 수 있다. //JS 의 <script> 와 유사

2) Expression(표현식) : <%= %> , 값을 출력하는 곳

3) Declaration(선언문) : <%--! --%> , 안쓰이는 추세, 주석2처럼 쓰자

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ page import = "java.util.*" %>
<%-- @붙으면 지시자(Directive) page 지시자 --%>
<%--! 여기는 선언문, 현재 페이지 전체에서 사용하는 전역변수, 메서드 정의, 안쓰이는 추세 --%>
<% //스크립틀릿
  int sum = 0;
  for(int i = 0; i <= 10; i++){
    sum += i;
  }
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
 <h1>hi jsp</h1>
<%-- 이것은 표현식(Expression) : 값을 출력하기 위한 곳 --%>
 <%= sum%>
<%
 for(int i = 0; i <= 10; i++){
%>
 <%=i%><br>
<% }sum=100; %>
sum: <%= sum%>
</body>
</html>
```

hi jsp

55 0
1
2
3
4
5
6
7
8
9
10
sum: 100

4. 지시자(Directive)

JSP 페이지에서 필요한 설정 정보를 지정할 수 있도록 제공하는 것이 지시자이다.
인코딩이나 컨텐츠 타입과 같이 JSP 페이지에서 필요한 설정 정보나 클래스의 임포트 등에 사용할 수 있는 page 지시자, 표준 태그 라이브러리를 사용할 수 있는 taglib 지시자, 정적으로 다른 JSP 페이지를 포함 시킬 수 있는 include 지시자 등이 있다.

1) jspPageDirective.jsp : page Directive는 페이지 설정정보나 클래스 임포트에 사용

```
<%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>
<%-- page 지시자는 현재 JSP 페이지에 대한 설정을 하는 곳 --%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<%      int sum = 0;
        for(int i=0; i<=100; i++){
                sum += i;         }
%>1~ 100 합 : <%= sum %>
</body>
</html>
```

2) includeDirective.jsp : 정적으로 다른 JSP 페이지를 포함
2-1) file 1

```
<%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>
<%@ page import="java.util.*" %>
<%
Calendar cal = Calendar.getInstance();
Date now = cal.getTime();
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>오늘의 메뉴</h2>- 밥 1,000원 <br>- 면 2,000원 <br>- 국 3,000원 <br>
<%@ include file="includeMenu.jsp" %>
<b><%= String.format("%1$TY년 %1$Tm월 %1$Td일", now) %></b>
</body>
</html>
```

2-2) file 2

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<h3>저녁 메뉴</h3>- 참치회 정식 30,000원<br><br>
```

5. 표현 언어(Expression Language)

extend가 없으면 object 를 자동으로 상속받거나, 연관된게 하나도 없으면 컴파일러 과정에서 자동 연결해준다.

1) 학생 한 명의 정보를 저장하는 VO(Value Object) 클래스

```
package com.jspstudy.ch03.vo;
public class Student {
	private String name; //정보보호를 위해 private 언급했는데 이해 부족
	private int age;
	private String gender;

	public Student() {}
	public Student(String name, int age, String gender) {
		this.name = name;
		this.age = age;
		this.gender = gender;	}
	public String getName() {
		return name;	}
	public void setName(String name) {
		this.name = name;	}
	public int getAge() {
		return age;	}
	public void setAge(int age) {
		if(age>150) {
			return;	}
		this.age = age; }
	public String getGender() {
		return gender;  }
	public void setGender(String gender) {
		this.gender = gender;	}	}
```

2) JSPStudyCh03/src/main/webapp/studentInfo.jsp

```
// Student 클래스를 사용하기 위해 page 지시자를 이용해 import 하고 있다.
<%@ page import="com.jspstudy.ch03.vo.Student"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

// Student 객체를 생성하고 각 프로퍼티를 설정하고 있다.
<%
Student s1 = new Student();
s1.setName("adsf"); s1.setAge(25); s1.setGender("남자");
Student s2 = new Student("어머나", 33, "여자");

// request 내장객체에 속성 이름을 지정하여 Student 객체를 저장한다. 업캐스팅은 자동으로 진행됨
request.setAttribute("s1",s1); request.setAttribute("s2",s2);

// 요청을 처리한 결과를 다른 JSP에 출력하기 위해 pageContext 내장객체의 forward() 메서드를 이용해
studentInfoResult.jsp로 포워딩 하여 프로그램의 제어 흐름을 이동시킨다. 일반적으로 서버 프로그래밍에서 요청을 처리하는
코드와 요청을 처리한 결과를 출력하는 코드를 분리하여 작성하는데 이럴 경우 아래와 같이 포워딩 기법을 사용해 프로그램
흐름을 다른 JSP로 이동시킨다.
pageContext.forward("studentInfoResult.jsp");%>
```

2-1) 파일의 실행 순서에 주의

위쪽의 스크립틀릿에서 request 내장객체의 속성 값을 읽어서 변수에 저장하고 그 변수의 값을 forward를 통해 다른 페이지에서 가져다 쓸 수 있게 된다. 만약 studentInfoResult.jsp 에서 파일을 실행한다면 request 영역의 속성을 읽어 올 때 null 값을 받으므로 NullPointerException이 발생한다.
그러므로 studentInfo.jsp를 실행하여 그 페이지에서 Student 클래스의 인스턴스가 생성되고 request 내장객체의 속성에 student 라는 이름으로 객체가 저장되어 이 페이지로 포워딩 될 수 있도록 studentInfo.jsp를 먼저 실행해야 한다.

3) JSPStudyCh03/src/main/webapp/studentInfoResult.jsp

```
<%@ page import="com.jspstudy.ch03.vo.Student"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

// JSP 페이지로 부터 포워딩되어 넘어온 request 영역의 속성에 저장된 데이터를 읽는다. Object 타입으로 넘어오기 때문에 다운 캐스팅이 필요하다. 아래에서 request.getAttribute("student")를 호출했을 때 request 내장객체의 속성에 student 라는 이름을 가진 객체가 존재하지 않으면 null을 반환한다. 그러므로 (Student)를 작성해서 다운 캐스팅을 해줘야 한다.
<%
Student s1 = (Student) request.getAttribute("s1");
Student s2 = (Student) request.getAttribute("s2");
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
	<h2>s1의 정보</h2>
	이름 : <%= s1.getName() %><br>
	<h2>s2의 정보</h2>
	이름 : <%= s2.getAge() %><br>
</body>
</html>
```

3-1) EL을 이용한 출력

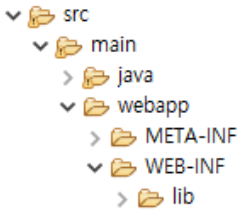
EL은 get 할 필요없이 ${브레이시스} 로 표현, 출력전용이라 다루기 어려우므로 JSTL로 다룬다.
Java 문법은 s3.Name 하면 오류가 생기지만 EL은 공란으로 출력되어 비교적 편리하다.

```
<%@ page import="com.jspstudy.ch03.vo.Student"%>
<%@ page language="java" contentType="text/html; charset=UTF-8"  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<h2>학생정보 출력하기 - 표현 언어(Expression Language)</h2>
이 름 : ${ s1.name }<br/>나 이 : ${ s1.age }<br/>성 별 : ${ s1.gender }<br/>
</body>
</html>
```

6. 표준 태그 라이브러리**(JSTL)**와 커스텀 태그**(Custom Tag)**

- JSTL(JSP Standard Tag Library) : 많이 사용되는 커스텀 태그중 공식인정되어 표준 태그로 격상, XML 태그 형태로 되어 있어 스크립틀릿과 반복문을 사용해 보다 간편하게 코드를 구성할 수 있고 가독성이 뛰어난 JSP 페이지를 작성할 수 있다. (ex: taglib)
- 커스텀 태그 : 별도로 라이브러리를 클래스 패스에 참조해야 사용가능 하다.

- taglib지시자를 사용하려면 아래 파일을 다운로드해 lib 경로에 넣어줘야한다.
- 다운로드 : https://tomcat.apache.org/download-taglibs.cgi > Impl, Spec > lib 에 직접 이동



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

1) com.jspstudy.ch03.vo > LottoNum.java

```
package com.jspstudy.ch03.vo;
public class LottoNum {
	private String times;
	private int num1;
	private int num2;
	private int num3;
	private int num4;
	private int num5;
	private int num6;
	private int bonusNum;
	//생성자 생성
	public LottoNum(String times, int num1, int num2, int num3,
				int num4, int num5, int num6, int bonusNum) {
				this.times = times;
				this.num1 = num1;
				this.num2 = num2;
				this.num3 = num3;
				this.num4 = num4;
				this.num5 = num5;
				this.num6 = num6;
				this.bonusNum = bonusNum;}
	//get & set 을 위해 마우스우클릭 > source > Generate Getters and Setters
	public String getTimes() {
			return times;}
	public void setTimes(String times) {
			this.times = times;}
	public int getNum1() {
			return num1;}
	public void setNum1(int num1) {
			this.num1 = num1;}
	public int getNum2() {
			return num2;}
	public void setNum2(int num2) {
			this.num2 = num2;}
	public int getNum3() {
			return num3;}
	public void setNum3(int num3) {
			this.num3 = num3;}
	public int getNum4() {
			return num4;}
```

2) JSPStudyCh03/src/main/webapp/lottoNum.jsp

```
<%@ page import="com.jspstudy.ch03.vo.LottoNum"%>
<%@ page import="java.util.ArrayList"%>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%
//회 차별 로또 당첨 번호를 저장할 ArrayList 객체 생성
ArrayList<LottoNum> lottoList = new ArrayList<LottoNum>();
LottoNum lotto = new LottoNum("968회", 2, 5, 12, 14, 24, 39, 33);
lottoList.add(lotto);
lotto = new LottoNum("969회", 3, 9, 10, 29, 40, 45, 7);
lottoList.add(lotto);
lotto = new LottoNum("970회", 9, 11, 16, 21, 28, 36, 5);
lottoList.add(lotto);
lotto = new LottoNum("971회", 2, 6, 17, 18, 21, 26, 7);
lottoList.add(lotto);
lotto = new LottoNum("972회", 3, 6, 17, 23, 37, 39, 26);
lottoList.add(lotto);
// Request 영역에 속성 이름을 지정하고 위에서 추출한 로또번호 리스트를 저장
request.setAttribute("lottoList", lottoList);
// 요청을 처리한 결과를 다른 JSP에 출력하기 위해 pageContext 내장객체의  forward() 메서드를 이용해 lottoNumResult.jsp로 포워딩 하여 프로그램의 제어 흐름을 이동시킨다. 일반적으로 서버 프로그래밍에서 요청을 처리하는 코드와 요청을 처리한 결과를 출력하는 코드를 분리하여 작성하는데 이럴 경우 아래와 같이 포워딩 기법을 사용해 프로그램 흐름을 다른 JSP로 이동시킨다.
pageContext.forward("lottoNumResult.jsp");%>
```

3) JSPStudyCh03/src/main/webapp/lottoNumResult.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%-- 표준태그라이브러리(JSTL)의 코어 라이브러리를 사용하기 위해 taglib 지시자를 사용해 접두어와 URI 식별자를 지정하고 있다. JSTL 라이브러리는 JSP 페이지에서 공통으로 사용하는 코드의 집합으로 코어, 포맷팅, 데이터베이스, XML처리, 함수지원 등의 기능을 제공한다. JSTL 라이브러리를 사용하기 위해서는 필요한 라이브러리를 구분하여 지정해야 하는데 바로 URI 식별자가 이 라이브러리를 구분하는 역할을 한다. 접두어는 JSP 페이지에서 라이브러리를 간편히 사용할 수 있도록 하기 위해 사용하는 접두어 일 뿐이다. taglib 지시자를 사용해 아래와 같이 지정하게 되면 코어 라이브러리의 URI 식별자가 접두어 c에 연결되고 JSP 페이지에서는 접두어 c를 사용하기만 하면 된다. 코어 라이브러리는 말 그대로 핵심적인 기능을 제공하는 라이브러리로 프로그래밍에서 필요한 변수 선언, 조건문, 반복문 등의 기능을 간편하게 사용할 수 있도록 지원하는 라이브러리 이다. --%>

// prefix에서 c는 관례적 사용
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<!DOCTYPE html><html><head>
<meta charset="UTF-8">
<title>EL과 JSTL을 이용해 로또 당첨 번호 리스트 출력</title><style>
ul {list-style-type: none;}
ul > li {height: 30px;}
li > span {color: blue;}
</style></head><body>
<h2>로또 당첨 번호 리스트</h2>

<c:if test="${not empty lottoList}">
<ul>
<%-- <c:forEach> 태그는 자바의 for문과 forEach문 두 가지 기능을 제공한다. 자바의 for문과 같이 초깃값부터 순차적으로 반복하기 위해 아래와 같은 속성을 지정하여 사용한다. var 속성에는 forEach문 안에서 사용할 변수 이름을 지정하고 begin 속성은 var 속성에 지정한 변수의 시작 값을 지정하며 end 속성에는 변수의 끝 값을 지정한다.(i <= end 가 적용된다.) step 속성은 생략할 수 있고 생략하게 되면 기본 값은 1이 된다. <c:forEach var="i" begin="0" end="10" step="1" > 아래에 사용된 코드는 forEach문의 기능을 하는 코드로 items 속성에 배열 변수의 이름을 지정하고 var 속성에 forEach문 안에서 배열의 각 항목을 저장하는 변수의 이름을 지정한다. items 속성에 지정할 수 있는 변수는 스크립틀릿에서 선언한 변수를 표현식<%=
```

```
member %>으로 지정할 수 있고 아래와 같이 속성에 저장된 속성 이름을 EL 식을 이용해 지정할 수도 있다. items 속성에
지정할 수 있는 데이터는 배열, Collection 객체, Iterator 객체, Enumeration 객체, Map 객체와 콤마(,)로 구분된 문자열 등을
지정할 수 있다. 이전 페이지인 lottoNum.jsp에서 request.setAttribute("lottoList", lottoList);를 이용해 하나의 요청 범위 안에서
유효한 request 객체의 속성에 "lottoList"라는 이름으로 저장했으므로 아래와 같이 EL을 이용해 items 속성에 값으로 지정하면
lottoList의 길이만큼 forEach 문이 반복되어 로또 당첨 번호 리스트가 출력된다.--%>

<c:forEach var="lottoNum" items="${ lottoList }" >

<%--표현언어(EL)로 객체의 프로퍼티를 읽기 위해서는 아래와 같이 속성 이름에 dot(.) 연산자를 사용해 객체의 프로퍼티
이름을 지정하면 읽어 올 수 있는데 이때 해당 객체는 반드시 getter 메서드를 가지고 있어야한다. 그렇지 않으면
JasperException이 발생한다. --%>

<li>${ lottoNum.times }차 - ${ lottoNum.num1 }, ${ lottoNum.num2},
${ lottoNum.num3}, ${ lottoNum.num4}, ${ lottoNum.num5},
${ lottoNum.num6} + <span>보너스번호</span>
${ lottoN}</li>
</c:forEach>
</ul>
</c:if>

<c:if test="${ empty lottoList }">
로또번호가 존재하지 않습니다.
</c:if></body></html>



<%@page import="com.jspstudy.ch03.vo.LottoNum"%>
<%@page import="java.util.ArrayList"%>
<%@ 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>
<%
  ArrayList<LottoNum> lottoList =  (ArrayList<LottoNum>)request.getAttribute("lottoList");
%>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <h2>로또 당첨번호 리스트 - 스크립틀릿</h2>
<%
  if(! lottoList.isEmpty()){
%>
  <ul>

 <% for(int i = 0; i < lottoList.size(); i++) {
    LottoNum lotto = lottoList.get(i);  %>

 <% //for(int i = 0; i < lottoList.size(); i++) {
            for(LottoNum lotto : lottoList){
    //LottoNum lotto = lottoList.get(i);
  %>
```

```
    <li><%= lotto.getTimes() %>차 - <%= lotto.getNum1() %>,
    <%= lotto.getNum2() %>, <%= lotto.getNum3() %>,
    <%= lotto.getNum4() %>, <%= lotto.getNum5() %>,
    <%= lotto.getNum6() %> +
    <span>보너스 번호</span><%= lotto.getBonusNum() %></li>
  <% } %>
  </ul>
<%  } %>
<%  if(lottoList.isEmpty()){ %>
  로또 번호가 존재하지 않습니다.
<%  } %>

  <!-- 로또 당첨번호가 있으면 -->
  <c:if test="${not empty lottoList }">
    <ul>
    <c:forEach var="lotto" items= "${lottoList}">
      <li>${lotto.times }차 - ${lotto.num1}, ${lotto.num2}, ${lotto.num3}, ${lotto.num4}, ${lotto.num5},
      ${lotto.num6} + <span>보너스 번호</span> ${lotto.bonusNum }</li>
    </c:forEach></ul></c:if>
  <!-- 로또 당첨번호가 없으면 -->
  <c:if test="${empty lottoList }">  로또 번호가 존재하지 않습니다.  </c:if></body></html>
```