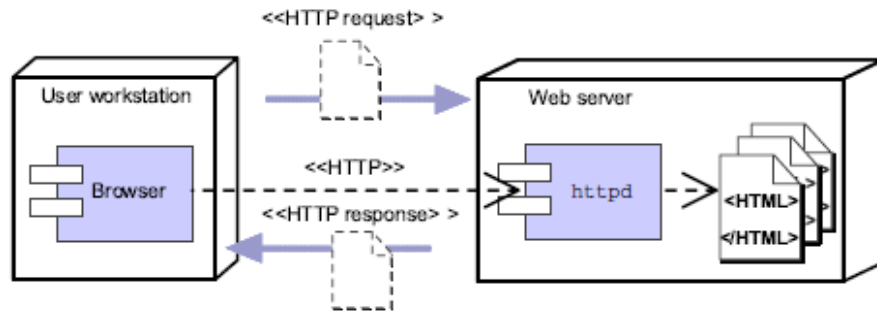


Servlet/JSP

Servlet

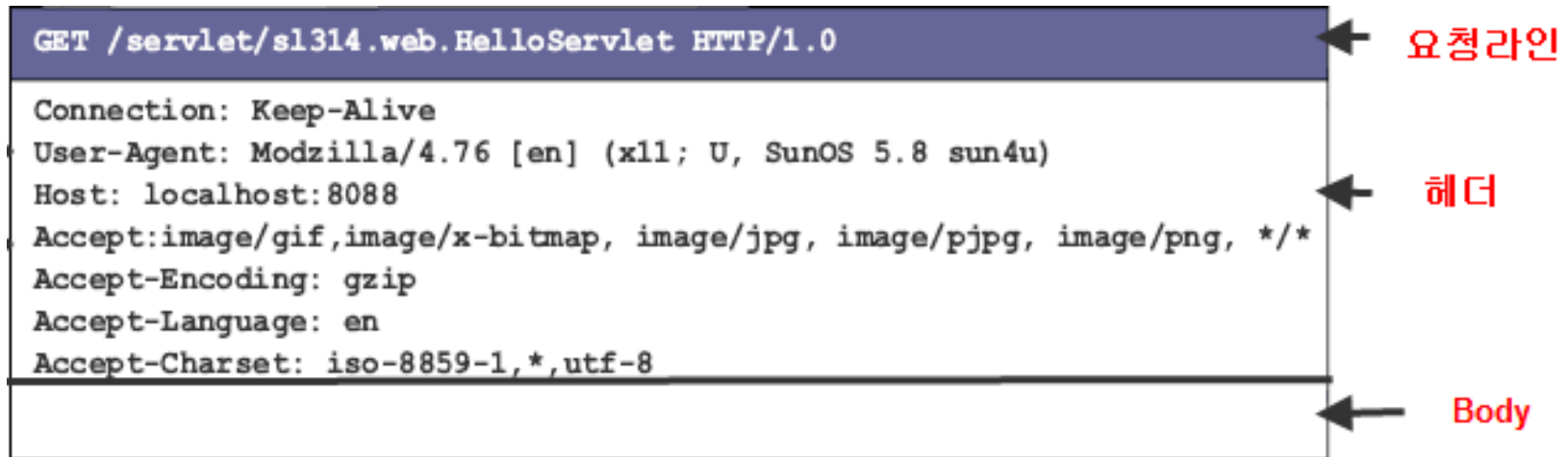
HTTP 프로토콜

- TCP/IP 기반



- Stateless 특징
 - 요청과 응답이 끝나면 연결을 종료
- HTTP Client
 - 웹 브라우저
- HTTP Server
 - 웹 서버 – Apache서버, IIS 등등
- HTML
 - HTTP 프로토콜 상에서 교환하는 문서를 작성하기 위한 언어
 - Markup 언어

HTTP 요청 및 요청방식



- GET방식

- 요청의 기본방식
- 데이터 요청이 목적
- URL을 통해 요청 파라미터 전달
 - 쿼리 스트링 : URL?name=value

- POST방식

- 서버에 데이터 전송이 목적
- Body를 통해 요청파라미터 전달

HTTP응답

HTTP/1.0 200 OK

응답라인

Content-Type: text/html

Date: Tue, 10 Apr 2001 23:36:58 GMT

Server: Apache Tomcat/4.0-b1 (HTTP/1.1 Connector)

Connection: close

Header

<HTML>

<HEAD>

<TITLE>Hello Servlet</TITLE>

</HEAD>

<BODY BGCOLOR='white'>

Hello World

</BODY>

</HTML>

Body

Web Application

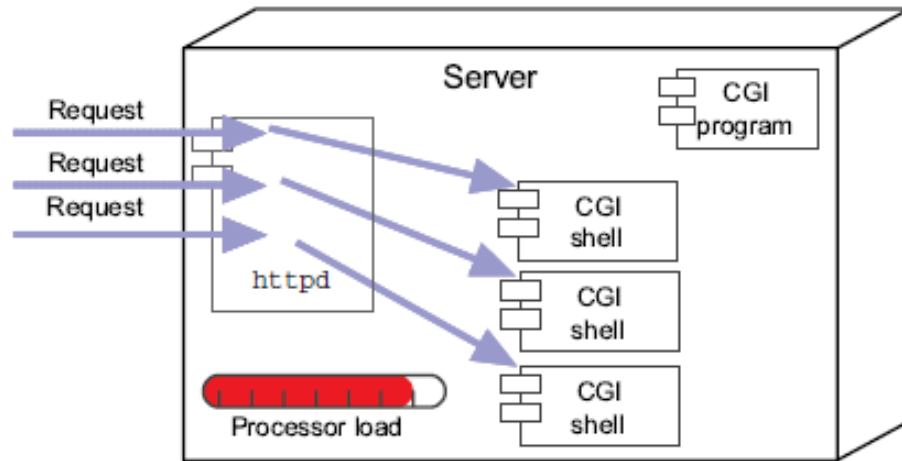
- Web 기반에서 실행되는 Application
 - Web Site(정적 서비스) + CGI(동적 서비스)
- Web Site
 - 정적 서비스 제공
 - HTML, Image..
- CGI
 - Common Gateway Interface
 - 동적인 서비스 제공
 - HTTP 프로토콜 하에서 실행되는 서버 단 프로그램 구현 interface
 - 대표적 기술
 - Servlet
 - JSP
 - ASP
 - PHP

서블릿의 특징

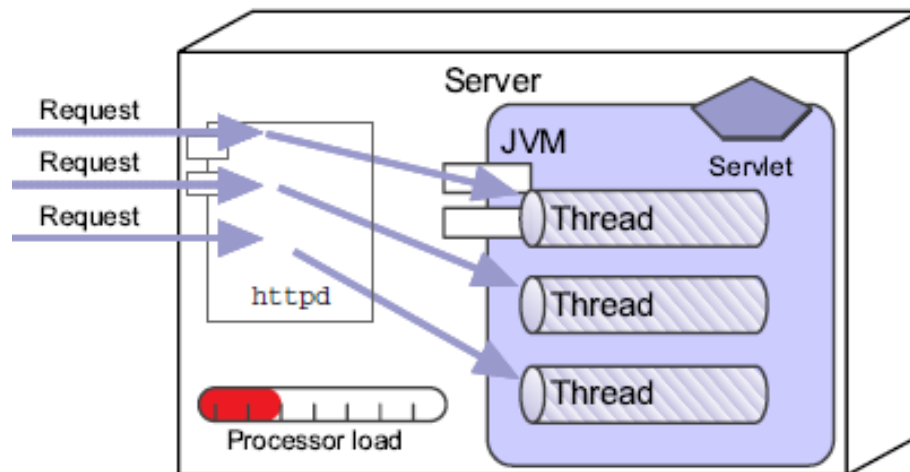
- Java 기반 CGI 기술
- Java의 다양한 API를 이용할 수 있다.
- 프로세스 기반이 아닌 스레드 기반으로 실행 됨

CGI와 Servlet

- CGI 실행

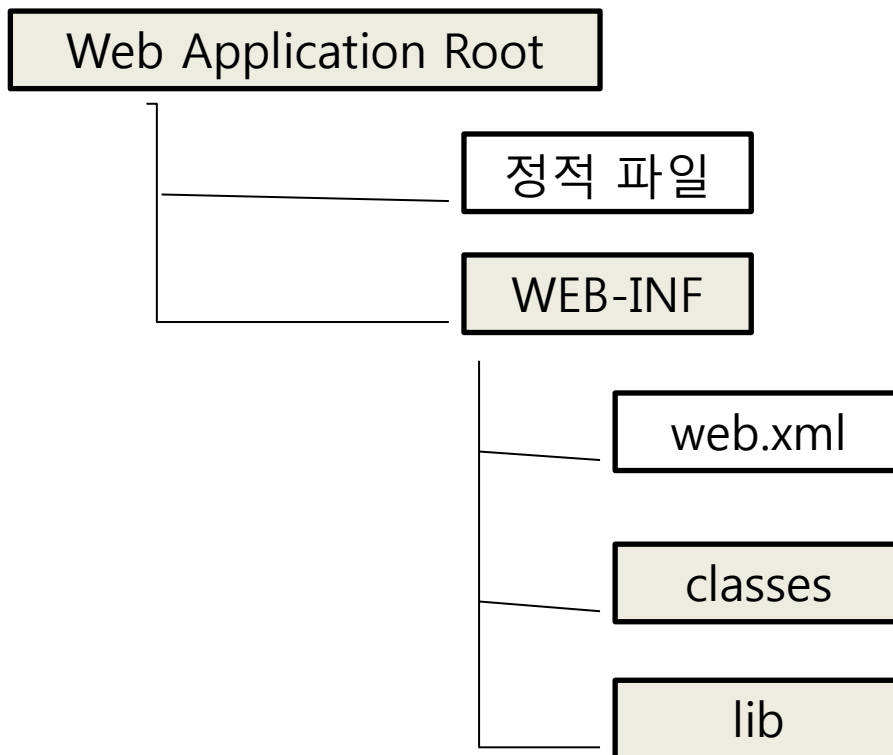


- Servlet 실행



Web Container와 배치

- Web Container
 - Servlet을 실행 시키는 환경
- 배치 경로



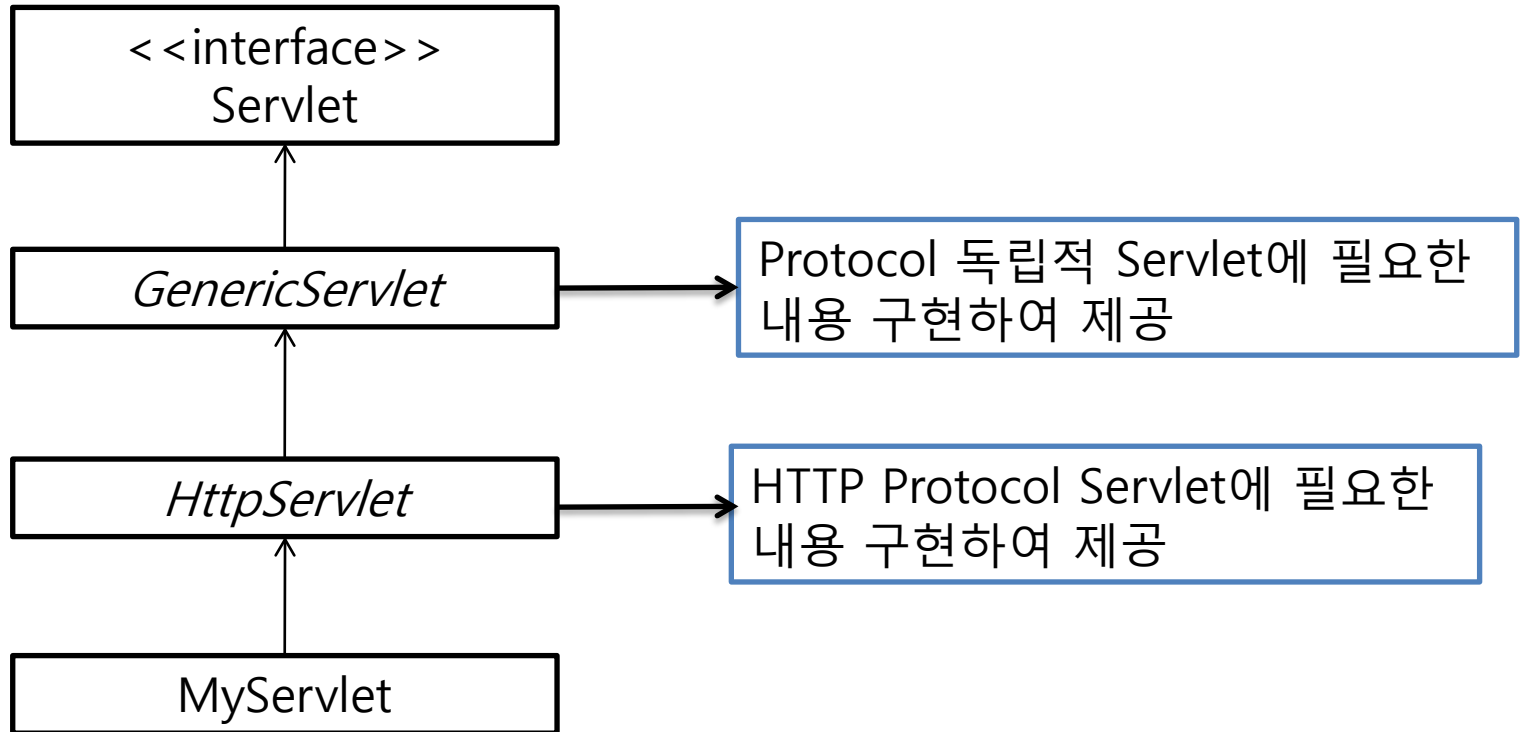
web.xml

- 배치(배포) 설명자 (Deployment Descriptor)
- Web Application은 반드시 하나의 DD파일을 가져야 함
- Web Application에 대한 설정을 하는 xml기반 파일
 - 서블릿 등록
 - 초기파라미터 등록
 - 보안 설정
- 위치
 - WEB-INF 경로 아래 위치
- Web Application 시작 시 메모리에 로딩된다.
 - 수정 시 Web Application을 재 구동해야 한다.

Servlet class작성 패턴

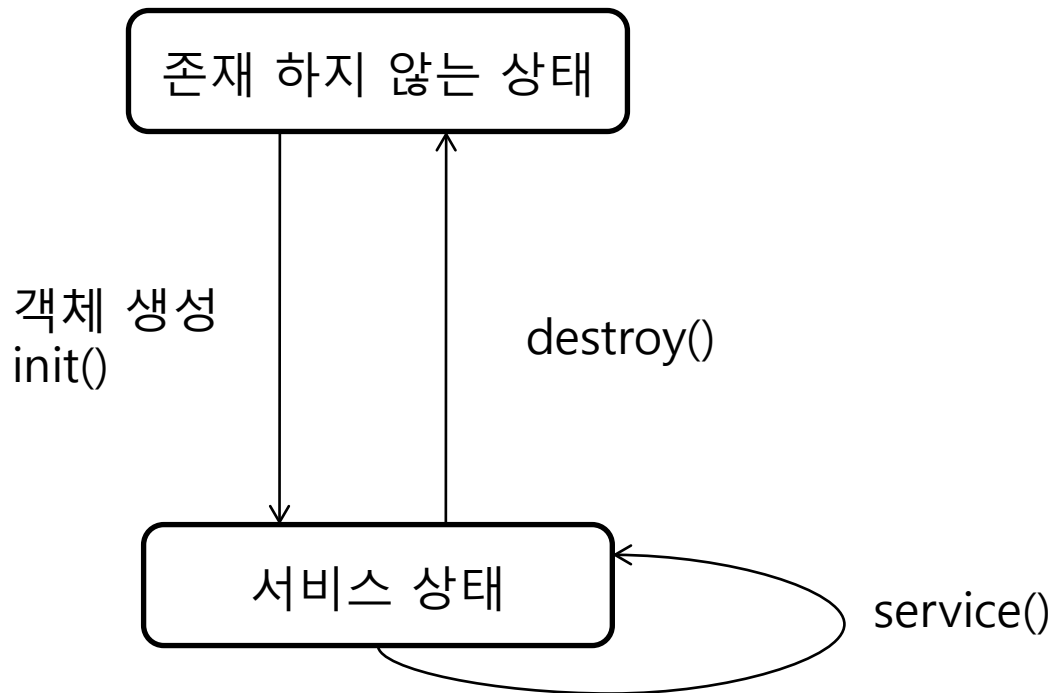
- public class로 작성
- Javax.servlet.HttpServlet 상속
- No-Argument 생성자 필요
- service 메소드 구현
 - public void doGet(request, response) 또는
public void doPost(request, response) 메소드 overriding
- web.xml에 등록
 - 서블릿 객체 등록
 - <servlet> 태그 이용
 - Client가 호출 하는 방법 등록
 - <servlet-mapping> 태그 이용

서블릿 상속구조



Servlet의 라이프 사이클

- Lifecycle(Callback) 메소드
 - init()
 - service()
 - destroy()



요청과 응답 관련 API

- HttpServletRequest
 - ServletRequest의 하위
 - 클라이언트의 요청정보를 가지고 있는 객체
 - 주요 업무
 - 요청 파라미터 조회
 - Session 객체 조회
 - request scope 상의 component간 데이터 공유 저장소 역할
- HttpServletResponse
 - ServletResponse의 하위
 - 요청한 클라이언트에게 응답하기 위한 정보를 가지고 있는 객체
 - 주요업무
 - 응답 Content type설정
 - 응답 Stream 조회
 - 응답 헤더 설정
 - 리다이렉트 방식으로 수행 넘기기

요청파라미터 처리

- 요청파라미터
 - 클라이언트가 CGI(서블릿) 으로 전송하는 데이터
 - HTML의 FORM 태그를 이용해 전송
- FORM 태그
 - 입력태그들을 묶어주는 역할
 - 속성
 - action – 전송할 CGI의 url 지정
 - method – 요청 방식 지정
- 입력 태그
 - select – 선택 태그
 - 속성
 - name : 전송할 값에 붙을 name값. cgi는 이 이름으로 값을 읽는다.
 - multiple : 다중 선택 (List화)
 - 하위 태그
 - option : 선택할 item들 등록

요청파라미터 처리

- textarea : 여러줄 입력 폼
 - 속성
 - cols : 열 수
 - rows : 행 수
 - name
- input : 다양한 입력 폼을 설정
 - 공통 속성
 - name
 - value
 - type : 입력 폼의 종류 지정
- input 태그의 종류
 - type 속성을 통해 종류 지정
 - text, password
 - file
 - checkbox, radio
 - hidden
 - submit, image, reset, button

서블릿에서 요청파라미터 처리

- HttpServletRequest를 통해 처리
 - 메소드
 - `getParameter("name") : String`
 - `getParameterValues("name") : String []`

```
<input type="text" name="id"/>
```

```
String value = request.getParameter("id");
```

```
<input type="checkbox" name="hobby" value="독서"/>
```

```
<input type="checkbox" name="hobby" value="음악감상"/>
```

```
String [] values = request.getParameterValues("hobby");
```

ServletConfig, ServletContext

- ServletConfig
 - Servlet객체가 일하는데 필요한 정보를 가지고 있는 객체
 - Web Container가 생성하여 Servlet객체 init() 호출 시 주입
 - Servlet객체당 하나씩 생성된다.
- ServletContext
 - Web Application 이 일하는데 필요한 정보를 가지고 있는 객체
 - Web Application이 처음 실행되는 시점에 Web Container가 생성
 - Servlet의 getServletContext() 를 통해 참조
 - Web Application당 하나 생성된다.

초기파라미터

- 개요

- 변경가능성 있는 문자열을 web.xml에 설정해 놓고 Servlet/JSP에서 호출해서 사용
- 문자열 변경 시 소스코드 변경 없이 수정 가능
- 메소드
 - `getInitParameter(String name) : String`

- ServletConfig를 이용한 방법

- Servlet 객체가 사용할 초기파라미터 설정
- `<servlet>` 태그의 sub태그 `<init-param>` 을 이용해 설정

- ServletContext를 이용한 방법

- Application 내 모든 Servlet이 사용할 수 있는 초기 파라미터 설정
- `<context-param>` 태그 이용

Attribute

- Attribute란
 - Servlet간 공유하는 객체
- Scope
 - Attribute를 저장하는 공간으로 공유 범위에 따라 다음 세가지로 나뉨
 - **request scope** : HttpServletRequest 이용
 - Request가 살아있는 동안 공유
 - **session scope** : HttpSession 이용
 - 한 명의 client(Web browser) 가 살아있는 동안 공유
 - **application scope** : ServletContext
 - Application이 시작해서 종료할 때 까지 공유
- 관리메소드 – Attribute는 key-value 쌍으로 관리됨
 - setAttribute(String key, Object value)
 - getAttribute(String key) : Object
 - removeAttribute(String key)
 - getAttributeNames() : Enumeration

클라이언트 요청 이동시키기

- 요청 디스패치(Dispatch) 방식
 - 리퀘스트가 살아있는 상태에서 수행을 넘긴다.
 - Request Scope를 이용해 데이터(Attribute)를 공유할 수 있다.
 - RequestDispatcher 객체 이용
 - RequestDispatcher rdp = request.getRequestDispatcher(String url)
rdp.forward(request, response)
- 리다이렉트(Redirect) 방식
 - 클라이언트에게 요청을 넘길 url을 알려 주어 다시 요청하게 하는 방식
 - 요청과 응답이 한번더 일어난다.
 - Request Scope를 이용해 데이터(Attribute)를 공유할 수 없다.
 - Response.sendRedirect("url")

Session관리란?

- Session
 - 하나의 클라이언트가 프로그램을 시작해서 종료 될 때까지를 하나의 Session 이라 함. 하나의 세션 동안 여러 번의 요청과 응답이 반복 될 수 있다.
- Session관리
 - 하나의 Session동안 사용자와 관련된 Data를 계속 유지 하도록 관리하는 것
 - Http Protocol의 Stateless한 특징의 해결책
- Session 관리방법
 - Session
 - Cookie
 - URLRewriting

Cookie를 이용한 Session관리

- 쿠키

- 서버가 브라우저(client)로 전송하는 작은 Text 정보이다
- 클라이언트의 정보를 클라이언트 컴퓨터에 저장한다.
- 클라이언트는 서버에 요청시 자신이 가진 데이터를 Http요청정보에 담아서 보내며 key-value형태로 관리된다..
- 저장 데이터(쿠키정보)는 문자열만 가능하고 저장량에도 제한이 있다.
(데이터 크기 : 4KB, 사이트당 :20개, 총 : 300개)

- 장점

- 서버의 부하를 줄인다.

- 단점

- 관리할 수 있는 데이터의 종류, 크기 제약, 보안상 취약

- 쿠키생성

- javax.servlet.http.Cookie를 사용
- Cookie c = new Cookie("popup","no");
- response.addCookie(c); //응답시 쿠키가 전송된다.

- 클라이언트가 보내온 쿠키정보 조회

Cookie [] cc = request.getCookies(); //보내온 쿠키가 없으면 빈 배열 리턴

- 주요메소드

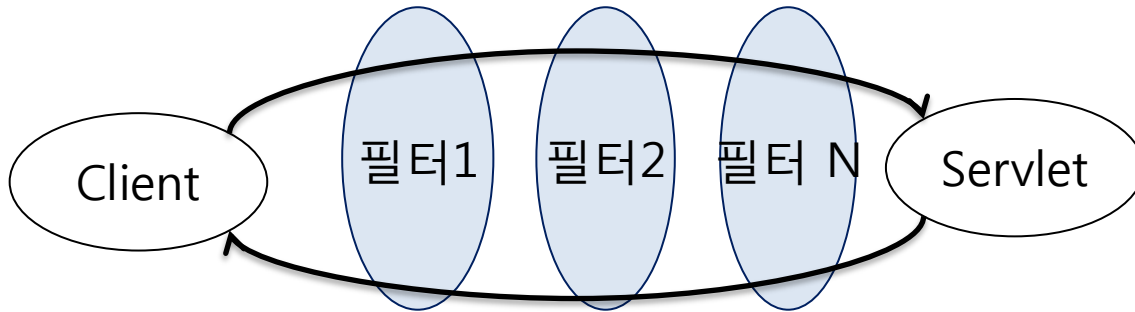
- String getName()
- String getValue()
- setMaxAge(int sec)

Session을 이용한 Session관리

- 클라이언트의 정보를 Server측에 저장.
- `Javax.servlet.http.HttpSession`객체를 이용하여 관리.
- 장점
 - 저장 데이터 타입이나 크기에 제한 없다.
 - 보안상 유리
- 단점
 - 서버에 부담
- 생성방법
 - `HttpSession session = request.getSession();`
 - 기존 세션이 있으면 기존 세션객체를, 없으면 새로 생성해서 return
 - `HttpSession session = request.getSession(false);`
 - 기존 세션이 있으면 기존 세션객체를, 없으면 null 리턴
- `HttpSession`의 주요메소드
 - `setAttribute(String name, Object value)`
 - `Object getAttribute(String name)`
 - `removeAttribute(String name)`
 - `setMaxInactiveInterval(int sec)`
 - `invalidate()`

Filter

- 클라이언트의 요청을 받은 component가 일하기 전 처리, 일한 후 처리를 담당.



- 구현
 - javax.servlet.Filter 를 implements 하여 작성
 - life cycle 메소드 구현
 - init(FilterConfig)
 - doFilter(ServletRequest, ServletResponse, FilterChain)
 - destroy()
 - web.xml에 filter 등록
 - <filter> 태그 : filter 객체 등록
 - <filter-mapping> 태그 : 필터 객체가 일 할 url을 매핑

JSP

Java Server Page

JSP LifeCyle

- 1) JSP 파일을 Servlet 파일로 변환 (.jsp->.java)
- 2) 변환된 Servlet을 컴파일(.java->.class)
- 3) Servlet 클래스 로딩
- 4) Servlet 클래스 객체 생성
- 5) jspInit() 호출 : Servlet의 init() 역할. Servlet이 처음 호출되었을 때 한번 일한다.
- 6) _jspService() 호출 : Servlet의 service() 역할. 클라이언트의 요청에 응답한다.
- 7) jspDestory() 호출 : Servlet의 destroy() 역할. 객체가 사라지기 직전에 호출된다.

JSP Tag

- 스크립트 태그(Script tag)
- 액션 태그 (Action tag)
 - 표준 액션 태그
 - Custom Tag

스크립트태그

- 지시자 태그(Directive Tag)
- 표현식 태그(Expression Tag)
- 선언 태그(Declaration Tag)
- 스크립트릿 태그(Scriptlet Tag)
- 주석

지시자 태그

- 컨테이너가 JSP를 Servlet으로 변환 할 때 특정 지시를 내리기 위해 사용되는 태그
- 종류
 - page : 페이지의 기능, 특징을 지시
 - include : JSP 페이지에 다른 페이지를 포함시키는 것을 지시
 - taglib : custom tag 사용과 관련된 지시
- 구문
<%@ 지시어 속성="값" 속성="값"...%>

Page 지시자태그

- 페이지 관련 환경을 정의하는 태그
- 주요 속성

1. **import** – jsp에서 사용할 패키지들을 import 시 사용. 속성들 중 유일하게 한 페이지에서 여러 번 올 수 있다.

```
<%@ page import="java.util.*, java.io.*"%>
```

2. **contentType** – jsp가 생성하는 응답의 MIME TYPE을 지정 – Servlet으로 변환 시 response.setContentType()으로 변환됨

```
<%@ page contentType="text/html; charset=euc-kr"%>
```

3. **session** – session 내장 객체의 생성여부를 결정 – true또는 false를 값으로 가진다. true가 default값

```
<%@ page session="false"%>
```

4. **errorPage** – JSP 페이지가 실행 할 때 오류(Exception)가 발생하면 수행을 옮길 page를 기술 한다. 요청 디스패치 방식으로 이동한다.

```
<%@ page errorPage="/error.jsp"%>
```

: 만약 페이지가 실행 중 에러가 발생하면 수행을 error.jsp로 옮겨 error.jsp가 응답하게 된다.

5. **pageEncoding** – response stream의 문자셋(character set)을 지정한다.

```
<%@ page pageEncoding="euc-kr"%>
```

6. **그 이외의 속성** : language, info, buffer, isErrorPage, autoFlush, isThreadSafe

표현식 태그

- Response Stream 을 통해 출력 하기 위한 태그
- 구문 : `<%= 출력내용%>`
- 예
 - `<%= “출력합니다.”%>`
 - `<% int i = 10%> <%=i%>`
- Servlet으로 변환 시 `out.write(출력내용);`
(예) `out.print(“출력합니다”)` 로 변환됨

선언태그

- JSP가 Servlet으로 변환 시 Servlet클래스의 member variable, member method를 선언할 때 사용
- 구문 : <%! 선언 %>
- 예 : <%!

```
public static int RED = 1;  
public int name;  
public void go(){  
}
```

%>

스크립트 릿

- 클라이언트 요청에 대한 동적 서비스 하는 코드
_jspService()의 구현부로 삽입되는 Java 코드를 작성
- 구문 : <% Java Code%>
- 예 : <% if(i>10){%>
10보다 큽니다.
<%}%>

주석

- HTML 주석 : `<!-- 내용 -->`
- JSP 주석 : `<%-- 내용 --%>`
- Java 주석 : 자바주석은 스크립트릿과 선언 태그 내에서만 사용가능
`//내용, /* 내용 */`

내장 객체(Implicit Variable)

- JSP 내에서 기본적으로 제공하는 객체
 - 생성이나 참조 코드 없이 바로 사용할 수 있다.
 - JSP 가 Servlet으로 변환될 때 container가 작성
- request : HttpServletRequest
- response : HttpServletResponse
- out : JspWriter - Servlet에서 PrintWriter 와 같은 역할
- session - HttpSession
- application - ServletContext
- config - ServletConfig
- pageContext - Servlet에는 없는 것으로 다른 내장객체들을 생성 할 수 있다.
- page – this. JSP가 서블릿으로 변환 되었을 때 그 Servlet 객체 자신
- exception – Throwable. JSP가 errorPage일 경우 사용 가능

JSP에서 Session

- JSP 는 default로 HttpSession객체가 생성됨 - 내장객체
- Login을 해야지만 볼수있는 페이지인 경우 - 2가지
 - 내장객체 session이 생성되지 않도록 page 지시자 태그에서 설정 후
HttpSession session = request.getSession(false)
코드 삽입
 - Login 처리 페이지에서 session scope에 로그인 여부를 체크할 데이터를 binding 한 뒤 jsp page에서 그 데이터를 읽어 들여 로그인 여부를 체크

액션 태그(Action tag)

- JSP 내에서 자바코드를 줄이기 위한 방법으로 제안된 태그들
- 액션태그는 JSP 내에 Java 코드 없이 특정 업무를 처리 하기 위한 태그
- 종류
 - 표준 액션태그 – JSP Spec에서 제공하는 표준 액션태그
 - Custom Tag – 사용자 정의 액션 태그
- 구문
 - <prefix:태그명 속성="value" [..]/>
 - 태그는 대소문자를 구분한다.
 - 속성의 값은 " 또는 ' 로 감싸준다.
 - 태그는 반드시 닫아야 한다.
<c:forEach items='list' var='mvo'> </c:forEach>
<jsp:getProperty name="cvo" property="name"/>

표준 액션태그

- JSP 스펙에서 제공하는 액션태그
- 구문 <jsp:태그명 속성="value"/>
- 주요 태그
 - 빈과 연동하기 위한 태그
 - useBean, setProperty, getProperty
 - 수행을 이동하기 위한 태그
 - include, forward, param

Bean과 연동하기 위한 액션태그

- `<jsp:useBean>` tag
- 속성 영역(scope : page, request, session, application)에 binding(등록)된 속성객체를 얻어온다. 만약 가져오지 못하면 새로 생성하고 그 영역에 넣어준다.
- 구문 `<jsp:useBean id="value" class="value" scope="value"/>`
- 속성
 - id : 자바 식별자 (variable)
 - class : 사용할 자바 클래스 이름
 - scope : 빈 객체가 살아있게 될 영역
값 - page, request, session, application

Bean과 연동하기 위한 액션태그

- 사용 예

```
<jsp:useBean id="cust" class="exam.Customer"  
  scope="request"/>  
<% cust.setName("홍길동");%>
```

의미 ->

```
Customer cust = (Customer)request.getAttribute("cust");  
if(cust==null){  
    cust = new Customer();  
    request.setAttribute("cust", cust);  
}  
cust.setName("홍길동");
```

Bean과 연동하기 위한 액션태그

- `<jsp:setProperty>`
- java bean에 값을 setting하는 태그
- 특정 값을 명시적으로 할당할 수도 있고 form tag를 넘어 온 Data를 이 태그를 이용해 넣을 수 있다.
- `<jsp:setProperty name="cust" property="name" [value="value"] [param="value"]/>`
- 속성
 - name : 프라퍼티에 값을 넣을 빈 객체의 이름. `<jsp:useBean>` tag에서의 id값 사용
 - property : 할당할 bean의 property 이름 (setter method의 set을 뺀 이름),
 - value : 프라퍼티에 할당 될 값, -> parameter 값보다 우선된다.
 - param : request객체로부터 전달되는 파라미터 값을 직접 넘겨 주려고 할 때 form tag의 name값을 넣어 준다. param의 값과 property의 값이 동일 할 경우 생략 할 수 있다.

Bean과 연동하기 위한 액션태그

사용예

```
<jsp:setProperty name="pvo" property="id" value="abc"/>
```

의미->

```
pvo.setId("abc");
```

```
<jsp:setProperty name="cust" property="name"/>
```

의미->

```
cust.setName(request.getParameter("name"));
```

```
<jsp:setProperty name="mvo" property="id" param="pid"/>
```

->의미

```
mvo.setId(request.getParameter("pid"));
```

```
<jsp:setProperty name="mvo" property="*/>
```

->의미 : 요청파라미터의 값을 mvo객체의 프라퍼티에 할당한다. 이때 서로의 이름이 같은 것끼리 매칭시켜 한번에 처리한다.

Bean과 연동하기 위한 액션태그

- `<jsp:getProperty>`
- Bean의 Property값을 조회하고 출력 태그
- 구문 : `<jsp:getProperty name="id" property="propertyName"/>`
- 속성
 - name : 프라퍼티를 읽어올 빈 객체의 이름. `<jsp:useBean>` tag에서의 id값 사용
 - property: bean의 프라퍼티 이름 (getter method의 get을 뺀 이름)

Bean과 연동하기 위한 액션태그

- 사용 예

```
<jsp:getProperty name="cust" property="age"/>
```

의미->

```
out.write(cust.getAge());
```

Include

- JSP page내에 다른 JSP (또는 HTML) page를 포함 시키는 기술
- include 지시자 태그를 사용하는 방법
 - 구문 : `<%@include file="파일명"%>`
- Action tag를 사용하는 방법
 - 구문 : `<jsp:include page="url"/>`
`<jsp:include page="url">`
`<jsp:param name="id" value="abc"/>`
`</jsp:include>`

Forward

- JSP의 수행을 다른 JSP(또는 HTML)로 옮긴다. – 요청 Dispatch 방식으로 옮긴다.
- Action tag를 이용한다.
 - 구문 : `<jsp:forward page="url"/>`
`< jsp:forward page="url">`
`<jsp:param name="id" value="abc"/>`
`</ jsp:forward >`

EL(Expression Language)

EL 이란

- JSP 2.0에서 새롭게 추가된 스크립트 언어.
- 기존의 Script tag의 표현식(Expression) tag에서 업그레이드 version.
- 주요 특징
 - 4개 속성영역에 저장된 Java bean의 Property 출력
 - 리터럴 데이터, 다양한 연산결과 출력
 - JSTL과 연동

표현방법

- 구문
 - `${value1.value2[.value3...]}`
 - `${value1["value2"]}`
 - value1은 **EL 기본객체**나 **속성명**이 들어와야 한다.
 - 구문 실행 결과가 null일 경우 출력 출력하지 않음
- JSP의 script(scriptlet, 표현식, 선언부)요소에는 사용 못함
- 예)
 - `<jsp:include page="/abc/${dir.page}"/>`
 - `${sessionScope.mvo["id"]}`님 환영합니다.
 - 첫 번 데이터는 `${requestScope.list[1]}` 입니다.

기본 객체

- EL은 11개의 기본객체를 제공하며 별다른 추가 코드 없이 사용할 수 있다.
 - `pageContext`를 제외하고 모두 Map(key-value)형식이다.
- `pageContext` : JSP의 `pageContext`와 같다.
- **`pageScope`** : JSP의 `page scope`와 동일하다.
- **`requestScope`** : `request scope`에 접근하기 위한 객체
- **`sessionScope`** : `session scope`에 접근하기 위한 객체
- **`applicationScope`** : `application scope`에 접근하기 위한 객체
- `cookie` : client가 전송한 쿠키의 값을 조회시 사용

기본 객체

- param : 요청 파라미터(form data들)를 읽어 올 때 사용.
- paramValues : 같은 이름으로 여러 개 설정 된 요청 파라미터(form data들) 값 조회시 사용
- header : 요청정보의 Header의 값을 읽어올 때 사용.
- headerValues : 같은 이름으로 여러 개 설정 된 요청정보의 Header의 값을 조회시 사용.
- initParam : Application 레벨의 초기파라미터 조회

4개의 저장영역에서 값 가져오기

- EL은 4개의 저장영역(page, request, session, application scope)에서 값을 쉽게 가져와 출력 할 수 있다.
 - `${cust.name}` 4영역의 scope에서 cust로 저장된 객체를 참조하여 `getName()`를 호출한 값을 출력.
 - 동일한 이름의 객체가 여러 영역에 있는 경우 기본객체를 사용하여 명시적으로 선언한다.
 - `${requestScope.cust.name}`

Collection에서 값 가져오기

- [] 연산자를 사용한다.
- 예:

```
String [] strs = {"a", "b", "c"};  
Map mp = new HashMap();  
mp.put("name1", "value1");  
session.setAttribute("array", strs);  
session.setAttribute("map", mp);
```

```
${array[0]}, ${array["1"]}, ${array[2]};  
${map["name1"]} 또는 ${map.name1}
```

연산자

- 산술연산자 : +, -, *, /(div), %(mod)
 - `${"10"+1}` - "10"을 숫자로 변환 후 계산
 - `${"일"+1}` - 오류 발생
 - `${null+1}` - null은 0으로 처리된다.
- 비교연산자
 - `==` 또는 `eq`
 - `!=` 또는 `ne`
 - `<` 또는 `lt`
 - `>` 또는 `gt`
 - `<=` 또는 `le`
 - `>=` 또는 `ge`
- 논리연산
 - `&&` 또는 `and`
 - `||` 또는 `or`
 - `!` 또는 `not`
 - `empty`

EL 비활성화

- JSP 페이지의 page 지시자 태그에서 `isELIgnored="true"` 로 설정

JSTL

(JSP Standard Tag Library)

JSTL이란

- 자주 사용될 수 있는 custom tag들을 모아 표준으로 정해놓은 tag library
- apache에서 다운받아 lib에 추가.
- custom tag이므로 사용 JSP에서 taglib 지시자 태그 설정 필요
 - `<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>`
- 참고 사이트
 - <http://java.sun.com/products/jsp/jstl>
 - <http://jakarta.apache.org/taglibs>

태그 종류

라이브러리	하위기능	접두어	관련 URI
코어	변수지원, 흐름제어, URL처리	c	http://java.sun.com/jsp/jstl/core
XML	XML관련 처리 Xml 흐름제어	x	http://java.sun.com/jsp/jstl/xml
국제화	국제화처리, 메시지 관리	fmt	http://java.sun.com/jsp/jstl/fmt
데이터베이스	데이터베이스 접근	sql	http://java.sun.com/jsp/jstl/sql
함수	Collection 처리 String 처리	fn	http://java.sun.com/jsp/jstl/functions

Core

- 변수 지원

- set :JSP에서 사용 될 변수설정
- remove :설정한 변수 제거

- 흐름제어

- if :조건문 처리
- choose: 다중 조건 처리 (else if)
- forEach: collection 또는 map의 각 항목을 처리 할때 사용 -loop
- forTokens: 구분자로 분리된 각각의 토큰을 처리시 사용

변수지원

- <set>
 - 4개의 속성저장 영역(scope)에 값(Attribute)을 binding 하거나 영역(scope)에 저장되어있는 bean의 property를 지정할 때 사용
 - 구문 : <c:set var="변수명" scope="scope" value="변수값"/>
 - var : 값을 지정할 변수의 이름
 - scope : 변수를 저장할 영역지정. page, request, session, application중 하나.
 - value : 변수의 값. 표현식이나 EL을 사용해 값을 차후에 지정할 수 있다. value가 null이면 var에 있는 기존의 속성값은 삭제된다.
 - target : 값을 세팅시킬 객체(bean이나 map만 가능함) 지정. var와 같이 쓸 수 없다.
 - property : 값을 설정할 프라퍼티명. Map의 key나 Bean의 setter 이름
- 예
 - <c:set var="name" scope="request" value="이순신"/>
 - 사용 : \${name}
 - <c:set target="\${requestScope.mvo}" property="name" value="홍길동"/>

변수지원

- `<remove>`
 - 특정 scope에 설정된 속성을 제거시 사용.
 - 구문
`<c:remove var="name" scope="scope"/>`
 - scope : 삭제할 scope
 - var : 속성명
 - 예) `<c:remove var="mvo" scope="session"/>`

흐름 제어

- <if>

- if문과 동일. 조건을 처리한다. 그러나 중첩된 if-else는 사용할 수 없다.

- 구문

- <c:if test="조건">

- 내용

- </c:if>

- 속성 : test – 조건이 true이면 내용을 실행.

흐름 제어

- <choose>

- 조건이 여러 개인 경우의 조건문을 처리하기 위한 태그
- 구문

```
<c:choose>
```

```
  <c:when test="조건">
```

```
    내용
```

```
  </c:when>
```

```
  <c:when test="조건">
```

```
    내용
```

```
  </c:when>
```

```
  <c:otherwise>
```

```
    내용
```

```
  </c:otherwise>
```

```
</c:choose>
```


흐름 제어

- <forEach>

- 특정 횟수만큼 구문을 반복하거나, 배열, Collection 또는 Map 에 저장된 값을 반복문을 이용해 조회하려고 할 때 사용. Java 의 반복문

- 구문

- <c:forEach var="변수" items="아이템">

- 내용

- \${변수}

- </c:forEach>

- 속성 : var : Collection에서 처리한 값을 저장할 변수

- items : 처리할 Collection

흐름 제어

- <forEach>

- for문 처럼 사용할 경우

- 구문

- ```
<c:forEach var="i" begin="1" end="20" step="2">
```

- ```
출력 : ${i} <br>
```

- ```
</c:forEach>
```

- map의 경우 key값은 .key, value는 .value로 출력

- ```
<c:forEach var="i" items= "${map}" >
```

- ```
 ${i.key}=${i.value}

```

- ```
</c:forEach>
```

JDBC

- JDBC
 - Java Program의 데이터베이스와 연동을 처리하는 API
 - DBMS Vender들이 표준 API를 구현해 제공한다.
 - Java.sql 패키지
- 프로그래밍 패턴
 1. Driver loading
 2. Connection(연결)
 3. Statement/PreparedStatement
 4. ResultSet(Select의 경우)
 5. close(Connection, Statement, ResultSet)

JDBC Programming Pattern

1. Driver Loading

Driver : DB와 P/G의 연결을 관리

```
Class.forName("Driver Class");
```

예)

```
String driver = "oracle.jdbc.driver.OracleDriver"
```

```
Class.forName(driver);
```

JDBC Programming Pattern

2. DB와 연결

DB와 연결을 위해 URL과 계정정보 필요

DB url : jdbc:subprotocol:subname

URL은 DBMS vender마다 다름.

- Oracle url - "jdbc:oracle:thin:@server-ip:1521:SID"

연결메소드 : `DriverManager.getConnection(url, id, pwd)` : `Connection`

```
String url = " jdbc:oracle:thin:@ 127.0.0.1:1521:XE"
```

```
Connection con =
```

```
    DriverManager.getConnection(url, "user", "pass");
```

JDBC Programming Pattern

3. Create a Statement

```
Statement stmt = con.createStatement();
```

Statement : query의 내용이 run-time에 결정되어진다.

(dynamic, 실행은 늦지만 융통성이 좋다.)

참고: PreparedStatement ; query의 내용이 compile-time에 결정되어진다. (static, 융통성은 떨어지지만 실행은 빠르다)

JDBC Programming Pattern

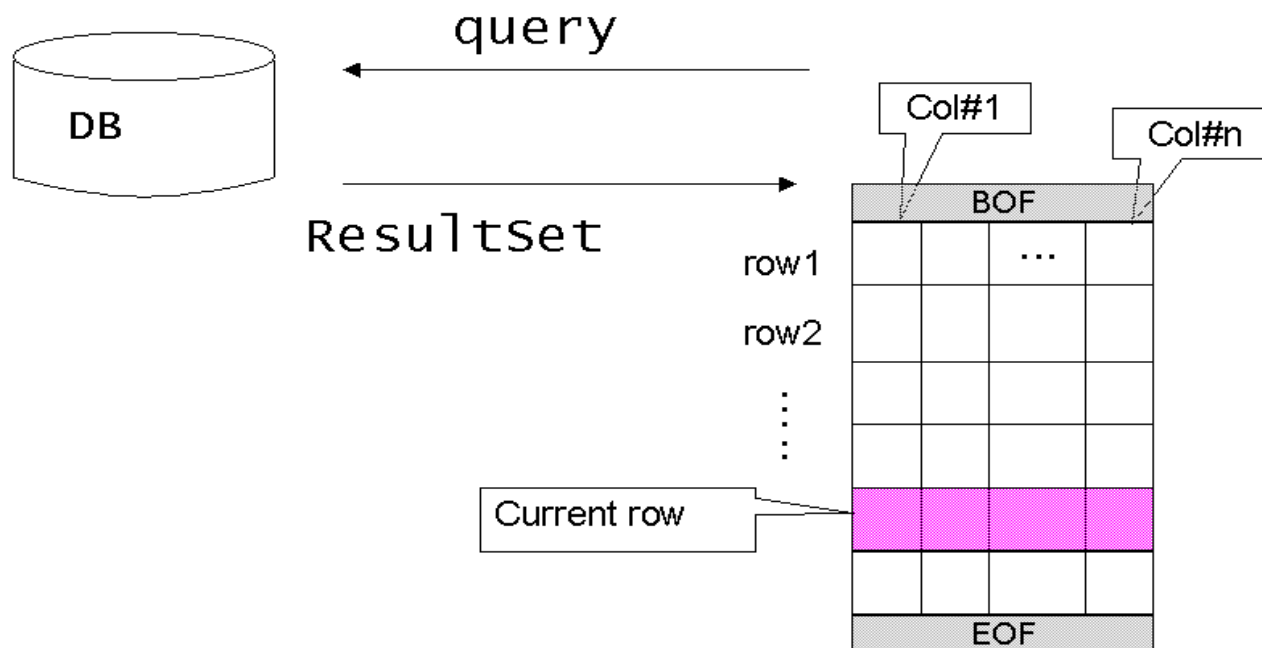
4. 쿼리 전송

```
String query = "sql문";  
int count = stmt.executeUpdate(query); //DML  
ResultSet rset = stmt.executeQuery(query); //DQL
```

- * SQL Query가 INSERT, DELETE, UPDATE인 경우에는 executeUpdate()를 사용한다. Query 실행 후 반영된 결과 record의 개수가 반환된다.*
- * SQL Query가 SELECT인 경우에는 executeQuery()를 사용한다. Query 실행후 SELECT한 결과가 ResultSet type으로 반환된다.*

- **ResultSet의 형태**

Retrieving Result



JDBC Programming Pattern

6. Retrieving Result (throws SQLException)

rset : ResultSet

```
1. while(rset.next()) {                                //Cursor를 한행씩 내린다.
    String str = rset.getString(1); //Column의 값을 가져온다.
    int i = rset.getInt("field_name"); //Column의 값을 가져온다.
}

2. if(rset.next()) {
    String str = rs.getString(1);
}
```

- 1) *SELECT* 결과가 여러 개일 경우
- 2) *SELECT* 결과가 하나일 경우

JDBC Programming Pattern

7. Close Resource (throws SQLException)

```
rset.close();  
stmt.close();  
con.close();
```

PreparedStatement

- **생성** - SQL 구문을 정의하고 변경 될 값은 치환문자(?)를 이용해 쿼리 전송 전에 값을 setting 한다.

```
String sql = "INSERT INTO MEMBER (ID, NAME, ADDRESS)  
VALUES(?,?,?)";
```

```
PreparedStatement pstmt = conn.prepareStatement(sql);
```

- **값 세팅** - 쿼리 전송전에 Column의 데이터 타입에 맞는 set 메소드를 호출하여 ? 에 값을 setting한다.

```
setXXX(int ?index, value);
```

- **쿼리 전송**

```
executeQuery() : ResultSet
```

```
executeUpdate() : int
```

Connection Pool과 DataSource

- DBMS와 연결 : 속도가 느리다.
- 해결책 : Connection Pool
- 내용
 - Connection Pool이란 Connection을 관리하는 객체 Pool
 - Connection들을 미리 생성 하여 Pool에 저장한 뒤 필요 시마다 빌려 쓰는 개념
 - 사용 후에는 다시 Connection Pool로 반납한다.
- Connection Pool은 작성하거나 API로 제공 되는 것을 사용한다.
 - Apache의 DBCP api
- DataSource : DriverManager의 upgrade version으로 Connection Factory 이다.
 - JDBC 2.0에서 지원
 - DataSource 객체는 연결할 DB에 대한 정보(driver, url, 계정) 를 가지고 있다.
 - 내부적으로 Connection Pool을 지원할 수 있다.
 - 구현 방식 : Vendor 마다 다름.
 - 요청 시 Connection을 생성하여 제공
 - Connection Pool을 이용해 Connection을 미리 생성후 요청 시 제공

MVC 패턴

- GUI기반 Application 설계 패턴
 - Macro 디자인 패턴
- Model
 - Business Logic 처리
 - Business Service(Manager)
 - Business 로직의 Work flow 처리
 - DAO (Data Access Object)
 - Database 관련 Business Logic 담당
- View
 - Presentation Logic 처리
- Controller
 - 프로그램의 실행흐름을 관리
 - 사용자의 요청에 대한 실행을 위해 Model과 Controller사이의 일의 흐름을 처리

Model2 패턴

- MVC 패턴을 Web Application에 적용한 것
- Model
 - Java Beans가 담당
- View
 - JSP가 담당
- Controller
 - Servlet이 담당

Front Controller 패턴

- 클라이언트의 요청을 집중 시키는 Controller를 생성하는 패턴
 - 요청을 한 곳으로 집중할 수 있다.
 - 모든 컨트롤러들에 공통적 로직을 처리 할 수 있다.