1. **Servlet**
2. 서블릿이란

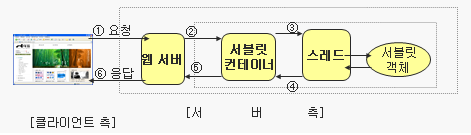
: JSP 표준이 나오기 전에 만들어 진 표준으로 **자바를 이용**해서 웹 애플리케이션을 개발하기 위해 만들어 졌으며 **자바 클래스를 웹에서 호출 및 실행** 할 수 있도록 한 표준.

1. 서블릿 프로그램의 작성법
2. 조건
3. 모든 서블릿의 접근 지정자는 항상 public이어야 한다.
4. 서블릿은 javax.servlet.http 패키지에서 제공하는 서블릿 클래스를 상속받아 구현한다.

* **Public** class HelloWorld extends **HttpServlet**{

1. 서블릿의 동작 원리

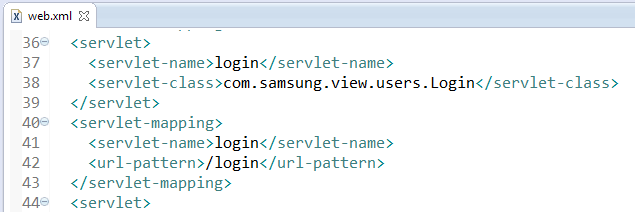
: 서블릿의 수행은 서블릿 컨테이너에 의해서 처리되고, 서블릿이 요청될 때 마다 스레드가 계속 생성되어 스블릿 객체를 생성하여 수행한다.



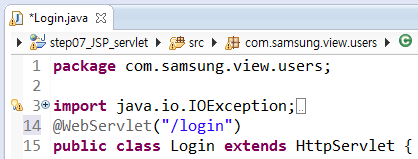
1. 브라우저에서 서블릿을 요청
2. WAS 안에 웹 서버가 서블릿 요청을 인식하여 서블릿 컨테이너에게 서블릿을 수행하도록 넘겨줌(교육의 경우 Tomcat을 사용)
3. 서블릿 컨테이너는 스레드를 기동하여 해당 서블릿 객체를 생성하여 이를 수행
4. 서블릿 수행 결과가 웹 서버에 전송
5. 이를 클라이언트에 전송
6. URL Mapping

: 실제 서블릿 클래스를 공개하지 않기 위해 사용한다. 매핑을 통해 사용자가 접근하는 URL 이름은 실제 서블릿의 물리적인 위치 정보가 아니기 때문에 이후 디렉토리 구조가 변하더라도 사용자에게 일일히 통보해줄 필요가 없다.

Web.xml 파일에서 url을 매핑하거나, 어노테이션(annotation)을 이용해 매핑할 수 있다.



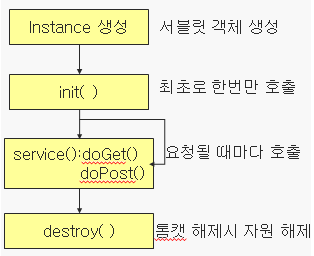
OR



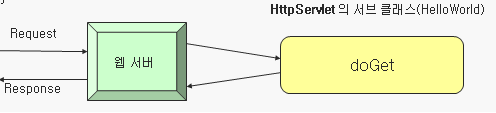
1. 서블릿의 라이프 사이클
2. 라이프 사이클

: 서블릿이 첫번째 요청인 경우에는 서블릿 클래스를 \WEB-INF\classes에서 찾아 메모리에 로딩하여 인스턴스(객체)를 생성한다.

이때 생성된 서블릿 인스턴스는 메모리에 계속 남아있게 되므로 이후부터는 섭르릿이 호출되어도 서블릿 인스턴스가 다시 생성되지 않고 이미 메모리에 로딩된 서블릿으로부터 서비스만 받기 때문에 수행속도가 빠르다.



1. 객체가 생성되면서 init 메소드가 단 한번 호출된다. Init 메소드에서 주로 초기화 작업을 한다.
2. 그 후에는 클라이언트의 요청이 있을 때 service 메서드가 실행된다.
3. 여러 클라이언트가 동시에 요청을 하더라도 스레드가 생성되어 동시에 service 메서드가 실행되기 때문에 수행 속도가 빠르다(서블릿의 장점)
4. 서블릿이 더 이상 서비스를 하지 않을 경우 destroy 메서드가 호출된다.
5. Get 방식과 Post 방식



1. Get 방식

: 리소스의 이름과 값(valure)를 url 뒤에 추가해서 전송(Query String)

* /login?id=aaa&password=aaa
* 보낼 수 있는 문자열의 길이에 한계가 있으며(2048 byte), 암호화가 가능하다.

1. 주소 입력란에 직접 요청하거나
2. Form 태그에서 method 속성 값을 get으로 주거나 혹은 생략할 경우,
3. <a> 태그에 의해서 하이퍼링크가 걸린 경우에 get 방식으로 요청한다.
4. Post 방식

: page header 입력해서 전송. 보낼 수 있는 문자열의 길이는 제한이 없고 바이너리로 데이터를 전송하는데, 암호화가 불가능하다.

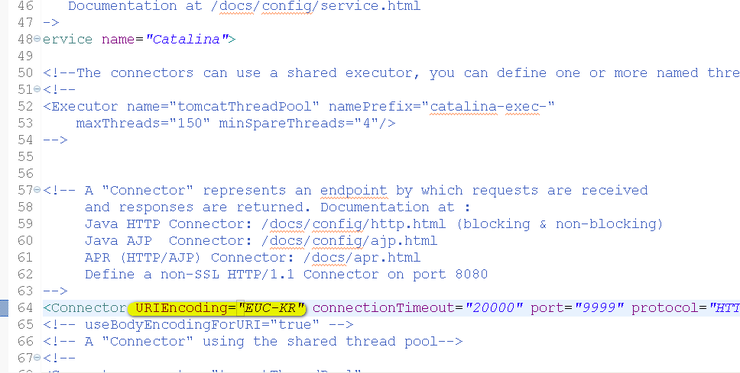
1. 한글화 처리
   1. Get 방식

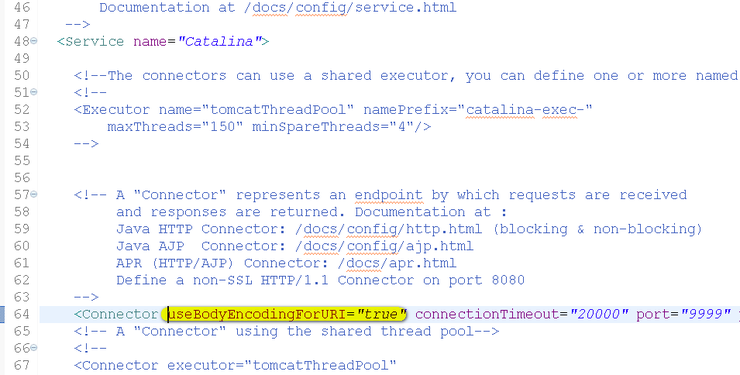
|  |  |
| --- | --- |
| 1. GET방식 이용시 파라미터 전송방법 | 인코딩 결정 |
| <a> 태그의 링크에 쿼리 문자열 추가 | 웹페이지 인코딩 사용 |
| HTML 폼(Form) 의 method속성값을 "GET"으로 지정해서 폼을 전송 | 웹페이지 인코딩 사용 |
| 웹브라우저의 주소입력란에 직접 쿼리문자열을 포함한 URL 입력 | 웹브라우저마다 다름 |

* 톰캣설치위치/conf/**sever.xml** 또는 이클립스의 Servers에 있는 **server.xml** 설정파일에 설정값 추가또는 변경.
* - server.xml 파일에서 <Connector>의 **useBodyEncodingForURI**속성의 값을 **true**로 지정하는 방법

GET방식으로 전달된 파라미터 값을 읽어올때 **request.setCharacterEncoding() 메서드로 지정한 캐릭터 셋이 적용**

* - server.xml 파일에서 <Connector>의 **URIEncoding** 속성의 값으로 원하는 캐릭터 셋을 지정하는 방법
* GET방식으로 전달된 파라미터 값을 읽어올때 **항상 URIEncoding 속성에서 지정한 캐릭터 셋이 적용**





② post 방식

: HttpServletRequest 객체의 setCharacterEncoding("인코딩방식");

메소드를 이용하여 아래와 같이 사용

* **request.setCharacterEncoding("UTF-8");**

1. 서블릿 핵심 API
2. HttpServletRequest API

- HTTP Request 관련 작업을 처리하는 핵심 API

- 클래스가 아닌 인터페이스로 제공된다.

1. HttpServletResponse API

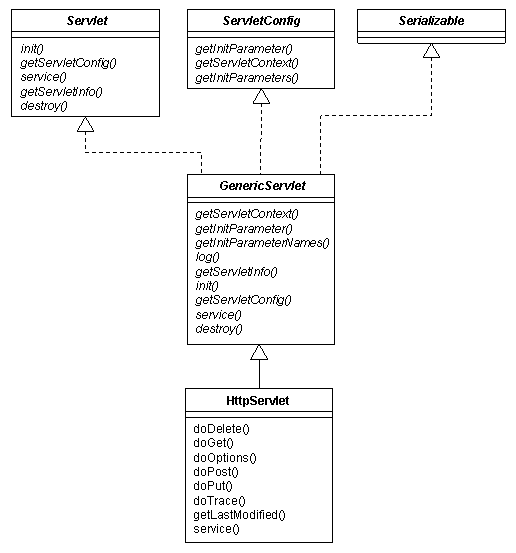
- HTTP Response 관련 작업을 처리하는 핵심 API

- 클래스가 아닌 인터페이스로 제공된다.

1. HttpServlet API

- 서블릿을 구현하기 위한 핵심 API

- 일반 클래스가 아닌 추상 클래스로 제공된다.

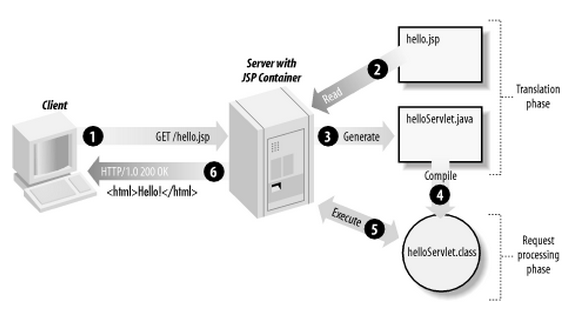


* Servlet 인터페이스에선 service(), init(), destroy() 이 세개가 생명주기(라이프사이클) 메소드이다.
* GenericServlet 클래스는 추상클래스이다. 필요한 대부분의 서블릿 메소드를 구현하였으며, 여기에는 Servlet 인터페이스에 정의된 것도있다. 이클래스를 상속받아 클래스를 생성할 일은 거의 없지만 대부분 서블릿의 '서블릿 행위' 라고하는 것들은 바로 이 클래스로부터 나왔다.
* HttpServlet 클래스 역시 추상클래스이다. HTTP적인 측면을 반영하기 위해 service()메소드를 재정의한 것이다. 즉 service() 메소드는 오로지 HTTP request와 HTTP response만 받아들이고 다른 어떤 서블릿 Request 와 response는 받지 않는다.

1. **JSP**
2. JSP란?

: JSP는 Sun Microsystems사(oracle사에 인수됨)의 자바 서블릿(Servlet) 기술을 확장시킨 기술로써 웹 환경 상에서 100% 순수 자바만으로 서버측 모듈을 개발하기 위한 기술

1. JSP의 동작 구조



① 클라이언트가 브라우저에서 hello.jsp를 요청한다.

② 웹 서버는 hello.jsp를 찾아서 jsp 컨테이너에게 해당페이지를 처리할 것을 요구한다.

③ jsp 파일을 서블릿으로 변환(파싱)한다. 만약 이전에 이미 요청되었었던 페이지라면 다시 파싱할 필요가 없으므로 5단계로 넘어간다. 한번만 컴파일하면 되기에 효율적이다. 물론 이전과 내용이 바뀌었다면 원래대로 4단계로 간다.

④ 서블릿 파일을 자바에서 실행 가능한 상태인 클래스 파일로 컴파일한다.

⑤ 클래스 파일은 메모리에 적재되어 실행된다. 이 후 웹서버로 결과가 넘겨진다.

⑥ 웹 서버는 브라우저가 인식할 수 있는 HTML형태로 결과를 클라이언트에게 응답한다.

1. JSP 페이지의 구성 요소
2. 디렉티브(Directive)

: <%@ 디렉티브이름 속성1="값1" 속성2="값2" ... %>

* JSP가 제공하는 디렉티브

- page : JSP페이지에 대한 정보를 지정. JSP가 생성하는 문서의 타입, 출력 버퍼의 크기, 에러 페이지등 JSP페이지에서 필요로하는 정보를 입력

- taglib : JSP 페이지에서 사용할 태그 라이브러리를 지정

- include : JSP 페이지의 특정 영역에 다른 문서를 포함 시킴

1. 스크립트 요소

* 표현식(Expression) : <%= ... %> , 값으 출력
* 스크립트릿(Scriptlet) : <% ... %> , 자바 코드를 실행
* 선언부(Declaration) : <%! ... %> , 자바 메서드(함수) 정의 및 전역변수 선언

1. 표현언어(Expression Language; EL)

: JSP페이지 내부에서 사용되는 간단한 스크립트 언어, JSP2.0부터 추가, 스크립트릿과 표현식 대신에 쉽고 간단하게 사용할수 있다.

1. 기본 객체 (Implicit Object)

: request, response, session, application, page 등....

1. 정적인 데이터 (HTML, CSS.. 등)
2. 표준 액션 택그 (Action Tag)

: 액션 태그는 JSP페이지 내에서 많이 사용하는 기능을 XML 기반의 태그로 제공하는 것

* <jsp:액션태그이름>

[참고] 디렉티브 및 스크립트 요소를 다음과 같이 액션태그를 이용해서 사용할 수있다.

<jsp:expression> : 표현식, <%= %>

<jsp:declaration> : 선언부 <%! %>

<jsp:scriptlet> : 스크립트릿, <% %>

<jsp:directive.page> : 페이지 지시자, <%@ %>

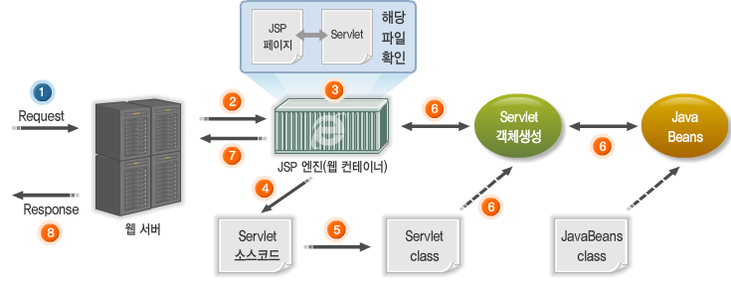
1. 커스텀 태그(Custom Tag)와 표준 태그 라이브러리(JSTL)

커스텀 태그는 JSP를 확장시켜주는 기능, xml 기반. JSTL(JaverServer Pages Standard Tag Library) 는 커스텀 태그 중에서 자주 사용되는 것들을 별도로 표준화한 태그 라이브러리 이다.

1. JSP 라이프 사이클

웹 client -------> JSP 요청 -----> 웹서버 ------> JSP요청

-----> 웹 컨테이너 -> 처리결과 ----> 웹서버 -----> 처리결과 ------>웹 client



1) client 요청이 웹서버에 전달되고 웹서버는 JSP요청이 들어온것을 확인하고

2) 바로 웹컨테이너인 JSP 엔진에게로 요청을 넘긴다.

JSP엔진은 3) 요청한 JSP페이지에 해당하는 Servlet이 있는지 확인하고, 없다면

4) Servlet코드를 생성하며 5) 자바 컴파일러에 의해 컴파일 과정을 거쳐 Servlet class가 만들어진다.

정상적으로 Servlet class가 만들어지면 6)웹 컨테이너는 메모리에 클래스 로딩 즉, 객체 생성과정을 거치고 내부적으로 callback 메소드가 수행되며 필요하다면 JavaBeans클래스도 메모리에 올려 사용한다. 7) callback 메소드인 서비스 메소드가 완료되면 웹 컨테이너는 결과값을 웹 서버에게 넘겨주고 8) 웹 서버는 웹 Client에게 응답을 주고 나면 이 연결은 끊어진다.

다시 한번 정리하자면 ,

.jsp 파일은 웹 컨테이너에서 .java형태로 코드 변환 -> 컴파일 -> 메모리 로드 -> 서비스

- JSP에 해당하는 서블릿이 존재하지 않을 경우 JSP페이지를 컴파일하여 서블릿을 생성후, 생성된 서블릿을 사용하여 클라이언트의 요청을 처리한다.

- JSP에 해당하는 서블릿이 존재하는 경우, 곧바로 서블릿을 사용하여 클라이언트의 요청을 처리한다.

즉 , JSP페이지를 요청할때에는 JSP를 직접 실행하는 것이 아니라 JSP를 자바소스코드로 변환한 뒤 컴파일해서 생성된 서블릿을 실행하는 것이다.

JSP페이지를 자바코드로 변경하는 단계를 변환(translation) 단계" 라고 하고,

자바코드를 서블릿 클래스로 변경하는 단계를 "컴파일(compile) 단계" 라고 한다.

1. **EL**

: 해석 그대로 표현 언어를 의미한다. jsp스크립트를 대신헤 속성 값들을 편리하게 출력하기 위해 제공된 언어이다.

① <%=hello%> - > ${hello}

<%= %> - > % { }

② test 변수를 표현할 때

- > ${test}

③ hello 객체의 test 변수를 표현할때

- > ${hello.test} ${hello['test']}

1) EL 내장 객체

|  |
| --- |
| pageScope = > Page 영역에 존재한느 객체를 참조할 때  requestScope = > Request 영역에 존재하는 객체를 참조할 때  sessionScope = > Session 영역에 존재하는 객체 참조할 때  applicationScope = > Application 영역에 존재하는 객체 참조할 때  param = > 파라미터 값 얻어올때  paramValues = > 파라미터 값을 배열로 얻어올 때  header = > Header 정보를 얻어올 때  headerValues = > Header 정보를 배열로 얻어올 때  cookie = > 쿠키 객체를 참조할 때  initParam = > 컨텍스트의 초기화 파라미터를 의미  pageContext = > pageContext 객체를 참조할 때 |

1. **JSTL**
2. JSTL 이란?

: JSTL은 JSP 표준라이브러리(JSP Standard Tag Library)의 약어이다. 라이브러리(library)는 여러 프로그램이 공통으로 사용하는 코드를 모아 놓은 코드의 집합을 말합니다. JSTL도 공통으로 사용되는 코드의 집합이다. 하지만 이 라이브러리는 우리가 보통 프로그래밍을 할 때 사용하는 라이브러리와는 달리 JSP페이지 안에서 사용할 수 있는 커스텀과 함수를 제공한다.

1. JSTL로 할 수 있는 일

- 간단한 프로그램 로직의 구사(자바의 변수 선언, if문, for 문 등에 해당하는로직)

- 다른 JSP페이지 호출 (<:redirect>,<c:import>)

- 날짜, 시간, 숫자의 포맷

- JSP 페이지 하나를 가지고 여러 가지 언어의 웹 페이지 생성

- 데이터베이스로의 입력, 수정, 삭제, 조회

- XML문서의 처리

- 문자열을 처리하는 함수 호출

3) 라이브러리

|  |  |  |  |
| --- | --- | --- | --- |
| 라이브러리 | 기능 | URI식별자 | 접두어  (prefix) |
| **코어** | 일반 프로그램이 언어에서 제공하는 것과 유사한 변수 선언, 실행 흐름의 제어기능을 제공하고, 다른 JSP페이지로 제어를 이동하는 기능도 제공합니다. | http://java.sun.com/jsp/jstl/core | c |
| **포매팅** | 숫자, 날짜, 시간을 포매팅하는 기능과 국제화, 다국어 지원 기능을 제공합니다. | http://java.sun.com/jsp/jstl/fmt | fmt |
| **데이터베이스** | 데이터베이스의 데이터를 입력/수정/삭제/조회하는 기능을 제공합니다. | http://java.sun.com/jsp/jstl/sql | sql |
| **XML처리** | XML문서를 처리할 때 필요한 기능을 제공합니다. | http://java.sun.com/jsp/jstl/xml | x |
| **함수** | 문자열을 처리하는 함수를 제공합니다. | http://java.sun.com/jsp/jstl/fuctions | Fn |

* <%@taglib prefix=”c” uri=” http://java.sun.com/jsp/jstl/core” %>