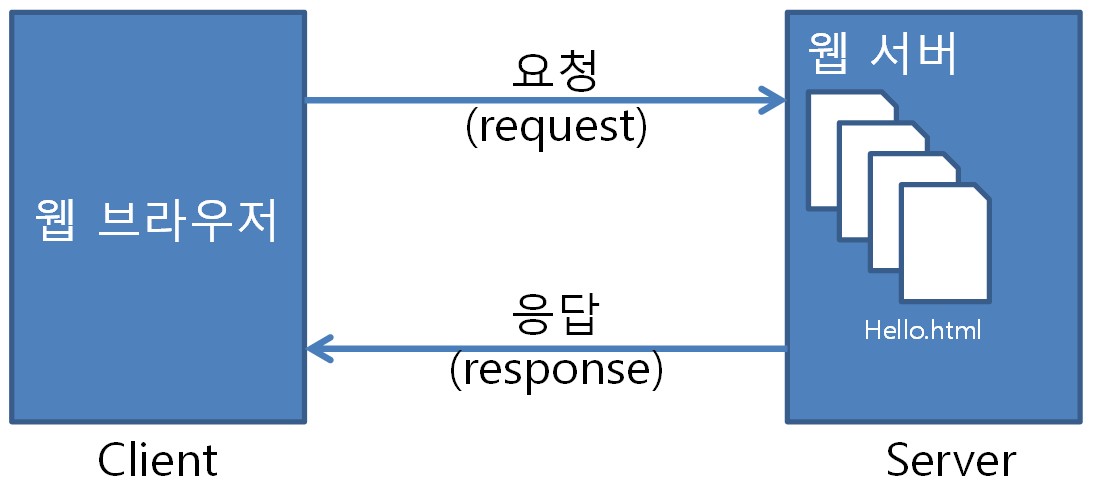
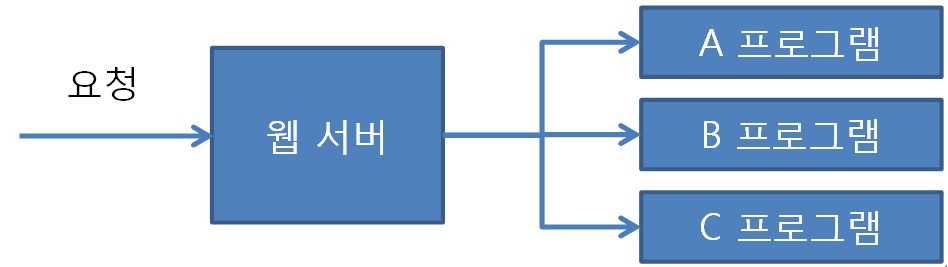
서버(Server)와 애플릿(Applet)의 약어인 서블릿(Servlet)은 썬마이크로시스템스에서 발표한 자바 기반 웹 프로그래밍 기술이다. 이번 강좌에서는 웹 애플리케이션의 역사를 통해 서블릿 등장 이전에는 어떤 기술들이 있었는지 살펴보고 서블릿의 장점과 동작 방법에 대해 알아본다.

**웹 애플리케이션 역사**  
초기 웹 애플리케이션은 현재와 같이 사용자마다 다른 정보를 보여줄 수 있는 동적인 웹페이지가 아닌 모든 사용자에게 같은 정보를 보여주는 정적인 웹페이지 형태로 표현됐다. 정적인 웹페이지는 다양한 비즈니스와 사용자의 요구사항을 충족시키지 못했고, 이러한 요구사항을 충족시키기 위해 동적인 웹페이지를 보여줄 수 있는 기술들이 등장하게 된다.

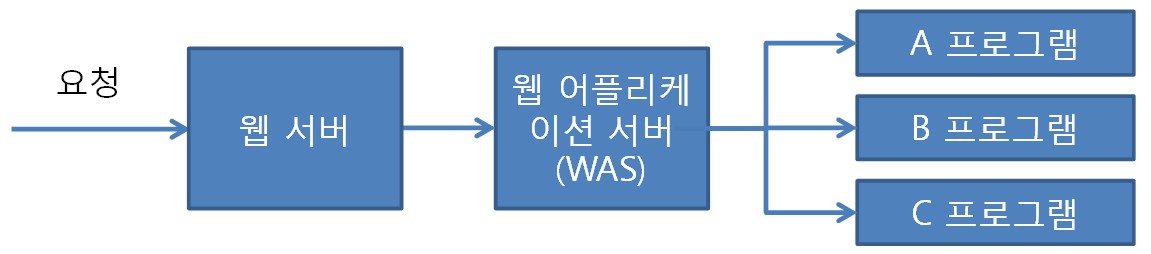
● 정적인 웹 기술  
HTML과 같이 언어로 작성된 웹페이지를 보여주는 방식이다. 정적인 웹페이지는 <그림 1>과 같이 요청에 따른 페이지만을 보여준다.

  
<그림 1> 정적인 웹페이지

● CGI  
CGI(Common Gateway Interface)는 동적인 웹 애플리케이션 작성기술로 서블릿보다 먼저 등장해 많은 웹 애플리케이션에서 사용됐다. CGI는 <그림 2>와 같은 동작 방법으로 클라이언트에서 요청을 받아 요청에 해당하는 프로세스(프로그램)를 실행시킨다. CGI는 대부분의 프로그래밍 언어로 작성할 수 있지만 그중에서도 펄(Perl) 언어로 많이 작성됐다.

  
<그림 2> CGI 동작 방법

● 서블릿  
서블릿은 CGI와 동일하게 동적인 웹 애플리케이션을 작성할 수 있는 기술이다. 현재 서비스되고 있는 대부분의 웹 애플리케이션은 서블릿으로 작성됐다. CGI와 비슷하게 클라이언트의 요청을 받아 해당하는 프로그램을 실행시켜주지만 CGI와는 조금 다른 동작 형태를 보인다.

  
<그림 3> 서블릿 동작 방법

**서블릿이란?**  
지금까지 웹 애플리케이션에 대한 흐름을 알아봤다. 그렇다면 서블릿은 CGI 방식과 어떤 차이가 있는지 알아보자.

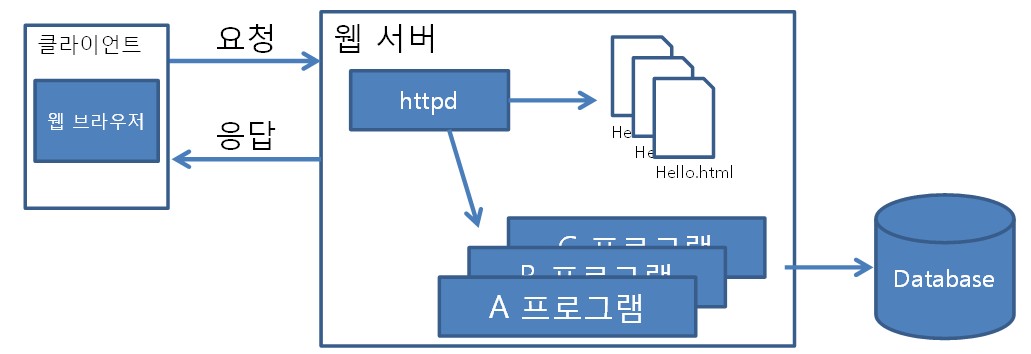
● 플랫폼 독립성  
자바로 작성된 서블릿은 자바의 특징 중 하나인 플랫폼 독립성을 지니고 있다. 플랫폼 독립성 지원이 갖는 장점은 하드웨어와 운영체제에 따라 다른 애플리케이션을 구현하지 않아도 된다는 점이다. 이러한 장점이 생기게 된 이유는 서블릿이 JVM 위에서 동작해 하드웨어와 운영체제가 달라져도 동일한 동작을 보여주기 때문이다

● 안전성  
자바로 작성된 서블릿은 자바의 장점 중 하나인 안전성을 가진다. 악의적이거나 잘못 작성된 서블릿이 서버의 파일시스템에 손상을 입히지 않도록 JVM에서 접근제어를 실행한다.

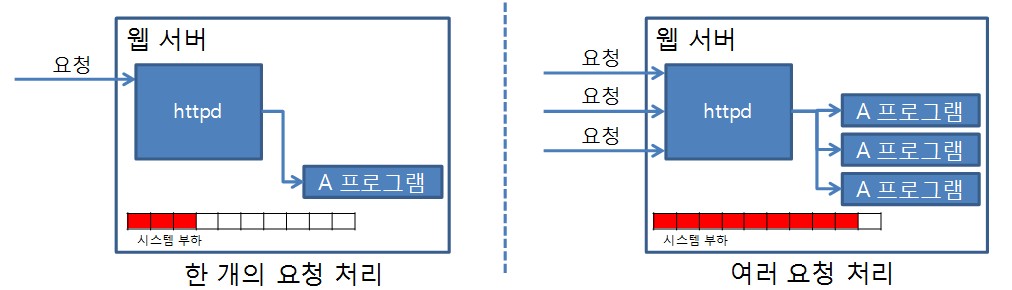
● 효율성  
서블릿은 CGI와 달리 효율적이다. CGI의 멀티프로세스 동작이 아닌 멀티스레드 동작으로 서블릿이 생성되면 서버가 종료되지 않는 이상 메모리로 남게 된다. 따라서 이후에 오는 요청에 대해서는 서블릿을 새로 생성하지 않고 동작을 이어갈 수 있기 때문에 시스템 자원(메모리)에서 큰 이점이 있다.

**왜 서블릿이 사용되는가?**  
CGI 기술이 있음에도 불구하고 서블릿이 등장한 가장 큰 이유는 CGI 기술의 동작 방법 때문이다. CGI 기술은 요청이 많으면 많을수록 시스템 자원을 많이 사용하는 구조다. 안전성을 위해서 시스템 자원을 넉넉하게 만들어야 하기 때문에 많은 비용이 들게 된다. 그에 비해 서블릿은 적은 시스템 자원으로 많은 요청을 처리할 수 있는 구조를 가지고 있다.

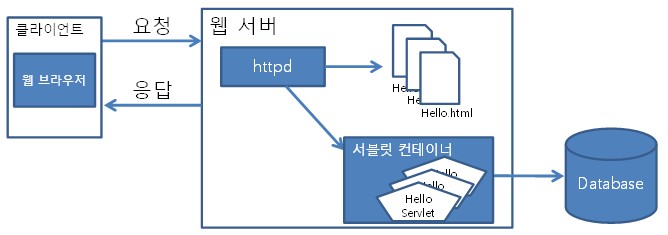
● CGI에서 사용하는 멀티프로세스 방법  
멀티프로세스 방법은 <그림 4>와 같이 웹서버에서 클라이언트의 요청을 받아 웹 애플리케이션을 직접 실행하는 구조를 가지고 있다. 요청에 대한 프로세스가 생성되면 요청을 처리하고 결과 값으로 클라이언트에 응답한 뒤 프로세스가 종료되는 형태다.

  
<그림 4> CGI에서 사용하는 멀티프로세스 동작

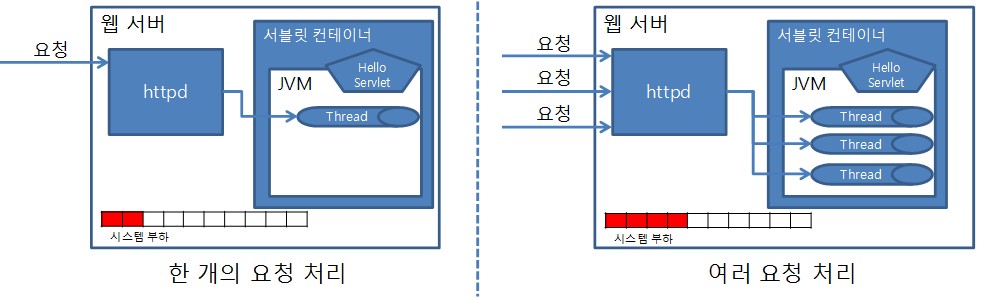
멀티프로세스 방식은 각각의 요청마다 프로세스를 생성하고 처리하는 구조다. 많은 요청이 들어오는 경우 프로세스를 계속 생성해야 하고 이는 시스템 부하로 연계돼 안정적인 서비스를 불가능하게 한다.

  
<그림 5> 멀티프로세스 방식에서 각각의 시스템 부하 비교

● 서블릿에서 사용하는 멀티스레드 방법  
서블릿에 사용되는 멀티스레드 방법은 <그림 6>과 같다. 클라이언트에서 받은 요청을 웹서버가 처리해 웹 애플리케이션을 거치지 않고 웹 컨테이너로 요청이 전달된다. 그리고 웹 컨테이너가 요청을 처리할 웹 애플리케이션을 실행시키는 형태다.

  
<그림 6> 서블릿에서 사용하는 멀티스레드 동작

멀티스레드 방식은 최초 요청 시 웹 애플리케이션을 실행한 후 종료하지 않은 상태에서 같은 요청이 여러 번 오는 경우, 실행되고 있는 웹 애플리케이션의 스레드를 생성해 요청을 처리하는 방법이다. CGI에서 사용하는 멀티프로세스 방식보다 시스템 부하를 줄여 안정적인 서비스를 제공할 수 있다.

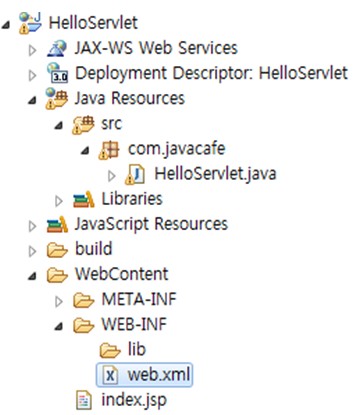
  
<그림 7> 멀티스레드 방식에서 각각의 시스템 부하 비교

**HelloServlet 실습**  
서블릿에 대해 좀더 자세히 알아보기 위해 HelloServlet을 작성하고 실행해 보자.  
  
다음 설명을 참고해 차근차근 따라하다 보면 HelloServlet을 작성할 수 있을 것이다.

· JDK를 다운로드해 설치하자(자바카페 → 강좌 → JAVA 강좌 참고)  
· Tomcat을 다운로드해 설치하자(자바카페 → 강좌 → 설치 강좌 참고)  
· 이클립스를 다운로드해 설치하자(자바카페 → 강좌 → 설치 강좌 참고)

**01.** 이클립스(eclipse)를 실행하고 HelloServlet 프로젝트를 생성한다. <그림 8>과 같이 HelloServlet.java 파일과 web.xml 파일을 생성한다.

· 프로젝트 생성 방법 : [File → New → Other.. → Web → Dynamic Web Project]

  
<그림 8> HelloServlet.java와 web.xml 파일 생성

이클립스에서 Dynamic Web Project 생성 시 WebContent/WEB-INF/web.xml 파일이 생성 되지 않는 경우가 있다. web.xml 파일이 생성되지 않았다면 WebContent/WEB-INF 위치에 파일을 생성하자.

● web.xml 생성 방법 : [File → New → Other.. → XML → XML File]

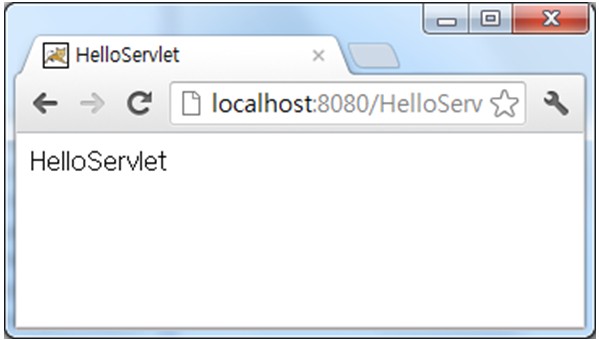
**02.** HelloServlet.java 파일에 <리스트 1>과 같이 작성한다.

|  |
| --- |
| **<리스트 1> HelloServlet.java** public class HelloServlet extends HttpServlet{ protected void doGet(HttpServletRequestreq, HttpServletResponseresp) throws ServletException, IOException { resp.setContentType("text/html");  PrintWriter out = resp.getWriter(); out.println("<html>"); out.println("<head><title>HelloServlet</title></head>"); out.println("<body>"); out.println("HelloServlet"); out.println("</body>"); out.println("</html>"); out.flush(); } } |

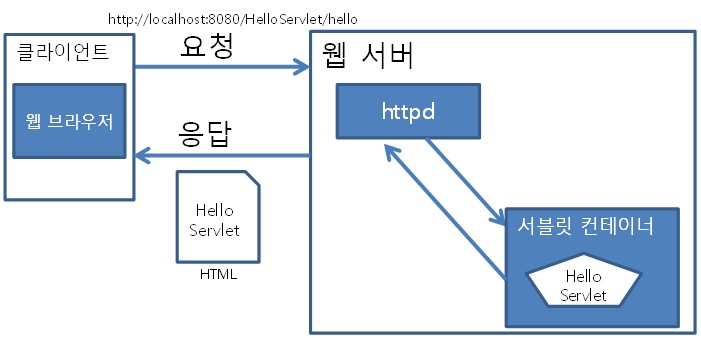
**03.** web.xml 파일에 <리스트 2>와 같이 작성한다.

|  |
| --- |
| **<리스트 2> web.xml**  <?xml version="1.0" encoding="UTF-8"?> <display-name>HelloServlet</display-name>  <servlet-mapping> <servlet-name>HelloServlet</servlet-name> <url-pattern>/hello</url-pattern> </servlet-mapping>  <servlet> <servlet-name>HelloServlet</servlet-name> <servlet-class>com.javacafe.HelloServlet</servlet-class> </servlet>  </web-app> |

**04.** 이클립스에서 HelloServlet을 실행하고 브라우저 주소창에 http://localhost:8080/HelloServlet/hello를 입력하면 <그림 9>와 같이 ‘HelloServlet’이 나오는 것을 확인할 수 있다.

  
<그림 9> HelloServlet 실행 화면

**HelloServlet 동작 흐름**  
HelloServlet은 어떤 흐름으로 동작하는지 알아보자.

  
<그림 10> HelloServlet 동작 흐름

**01.** 웹브라우저에서 웹서버로 HelloServlet을 요청한다(http://localhost:8080/HelloServlet/hello).

**02.** 요청 받은 웹서버는 서블릿 컨테이너에게 HelloServlet 처리를 요청한다.

**03.** 서블릿 컨테이너는 웹서버로부터 받은 정보로 HelloServlet을 찾아 실행한다.

**04.** HelloServlet은 처리결과(‘HelloServlet’을 보여주는 HTML 문서)를 웹서버에 전달한다.

**05.** 처리결과를 받은 웹서버는 결과 내용을 통해 웹브라우저에게 응답한다.

**서블릿 기술**  
HelloServlet의 동작 흐름에 이어 서블릿 사용을 위한 웹 기초와 서블릿 기본구조를 알아보자.

**(1) 서블릿 컨테이너**  
서블릿은 서블릿 컨테이너 위에서 동작한다. 먼저 서블릿 컨테이너는 무엇인지, 어떠한 기능을 제공하는지 살펴보자.

● 생명 주기 관리  
서블릿 컨테이너는 서블릿을 로드해 초기화 메소드(init 메소드)를 호출한다. 또 클라이언트의 요청으로 서블릿 메소드를 호출하며, 서블릿 컨테이너가 종료되면 서블릿의 종료 메소드(destroy 메소드)를 호출해 메모리를 정리한다.

● 통신 지원  
서블릿 컨테이너는 웹서버와 서블릿이 통신할 수 있는 기능을 지원한다. 예를 들면, 웹서버로부터 받은 요청을 분석해 요청에 따른 서블릿을 실행시키는 것이다. 또한 서블릿에서 웹서버의 정보를 확인할 수 있는 기능을 제공한다.

● 멀티스레딩 지원  
서블릿 컨테이너는 클라이언트의 요청에 따라 서블릿을 생성하고, 이미 생성된 서블릿에 대한 요청은 스레드로 생성해 서블릿을 실행한다.

서블릿 컨테이너의 종류에는 대표적으로 Apache Tomcat, JEUS(Java Enterprise User Solution), Weblogic, Webspear 등이 있으며 이런 서블릿 컨테이너들은 다양한 기능을 제공해 개발자가 신경 써야 하는 많은 부분들을 관리한다.  
  
이번 강좌에서 사용할 서블릿 컨테이너는 Apache Tomcat이다. 오픈소스인 Apache Tomcat은 무료로 사용할 수 있다.

**(2) HTTP 요청**  
클라이언트에서 웹서버로의 요청은 주로 HTTP 프로토콜이 사용된다. HTTP 프로토콜에는 여러 메소드들이 있지만 그 중에서도 GET과 POST가 많이 쓰인다.

● GET 메소드  
GET 메소드는 자원의 위치를 URL로 나타낸 것으로 GET 요청으로 해당 자원을 가져올 수 있다. GET 메소드는 ‘?’ 뒤에 쿼리 스트링이라고 불리는 문자열을 덧붙여 전송해 좀더 구체화된 정보를 요청할 수 있다. 단점은 요청한 정보가 URL에 표시돼 보안상 좋지 않다는 점과 문자열에 제한이 있어(브라우저에 따라 다를 수 있다) 많은 양의 정보를 보낼 수 없다는 것이다.

● POST 메소드  
GET 메소드와 마찬가지로 서버에 요청할 때 사용되며 차이점은 보내는 정보를 URL이 아닌 Body에 추가해 요청한다는 것이다. GET 메소드의 단점인 문자열 제한이 없고 URL에 요청 정보가 보이지 않아 보안적인 측면에서 GET보다 좋다.

간단한 예제로 GET 메소드와 POST 메소드 동작을 확인해 보자.

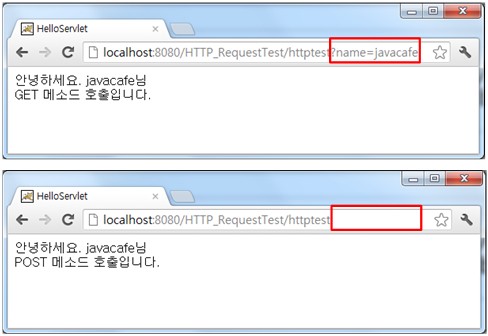
|  |
| --- |
| **<리스트 3> SEPT ① : Get 메소드 호출을 위한 입력 페이지 작성**  <!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01 Transitional//EN""http://www.w3.org/TR/html4/loose.dtd"> <html> <head> <metahttp-equiv="Content-Type"content="text/html; charset=EUC-KR"> <title>HTTP Test</title> </head> <body> GET 메소드로 이름을 전송합니다. <formaction="/HTTP\_RequestTest/httptest"method="get"> 이름 : <inputtype="text"name="name"size="10"> <inputtype="submit"value="전송"> </form> </body> </html> |

|  |
| --- |
| **<리스트 4> STEP ② : Post 메소드 호출을 위한 입력 페이지 작성**  <!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01 Transitional//EN""http://www.w3.org/TR/html4/loose.dtd"> <html> <head> <metahttp-equiv="Content-Type"content="text/html; charset=EUC-KR"> <title>HTTP Test</title> </head> <body> POST 메소드로 이름을 전송합니다. <formaction="/HTTP\_RequestTest/httptest"method="post"> 이름 : <inputtype="text"name="name"size="10"> <inputtype="submit"value="전송"> </form> </body> </html> |

|  |
| --- |
| **<리스트 5> STEP ③ : 요청 받아 이름을 보여주는 doGet/doPost 메소드 작성**  publicclassHttpRequestTestextends HttpServlet{ protectedvoid doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException { req.setCharacterEncoding("euc-kr");  resp.setContentType("text/html; charset=euc-kr"); PrintWriter out = resp.getWriter(); out.println("<html>"); out.println("<head><title>HelloServlet</title></head>"); out.println("<body>"); out.println("안녕하세요. " + req.getParameter("name") + "님 <br>"); out.println("GET 메소드 호출입니다."); out.println("</body>"); out.println("</html>"); out.flush(); }  protectedvoid doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException { req.setCharacterEncoding("euc-kr");  resp.setContentType("text/html; charset=euc-kr"); PrintWriter out = resp.getWriter(); out.println("<html>"); out.println("<head><title>HelloServlet </title></head>"); out.println("<body>"); out.println("안녕하세요. " + req.getParameter("name") + "님 <br>"); out.println("POST 메소드 호출입니다."); out.println("</body>"); out.println("</html>"); out.flush(); }  } |

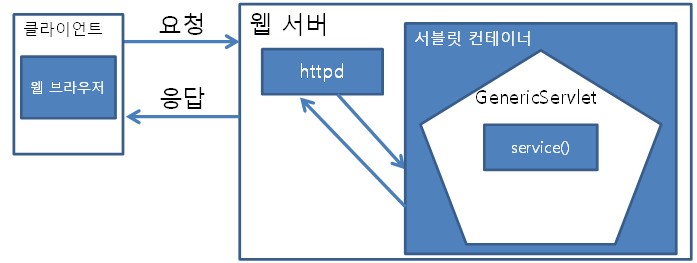
|  |
| --- |
| **<리스트 6> STEP ④ : web.xml 작성**  <?xml version="1.0" encoding="UTF-8"?> <display-name>HTTP\_RequestTest</display-name>  <servlet-mapping> <servlet-name>HttpTestServlet</servlet-name> <url-pattern>/httptest</url-pattern> </servlet-mapping>  <servlet> <servlet-name>HttpTestServlet</servlet-name> <servlet-class>com.javacafe.HttpRequestTest </servlet-class> </servlet>  </web-app> |

예제를 실행해보면 GET 메소드와 POST 메소드 요청 시 URL에 차이가 나는 것을 확인할 수 있다.

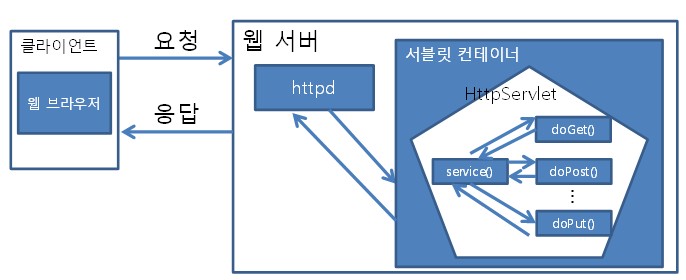
  
<그림 11> 예제 실행 결과

**서블릿 종류**  
서블릿에는 GenericServlet(javax.servlet.GenericServlet)을 상속받아 구현하는 방법과 HttpServlet(javax.servlet.http Servlet)을 상속받아 구현하는 두 가지 방법이 있다.

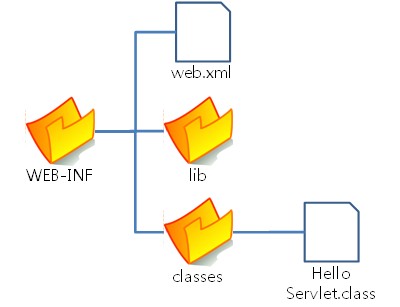
● Generic 서블릿  
GenericServlet은 프로토콜 독립적인 서블릿으로 service()를 오버라이딩해 사용해야 한다. <그림 12>는 GenericServlet이 동작하는 흐름이다.

  
<그림 12> 요청을 다루고 있는 GenericServlet

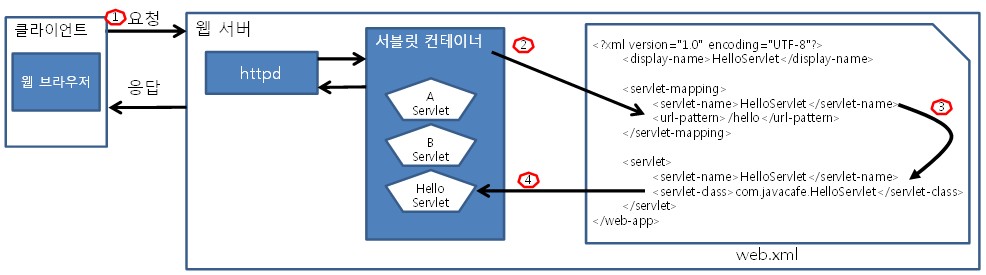
● HTTP 서블릿  
HttpServlet은 GenericServlet을 상속받아 HTTP 프로토콜에 관련된 기능을 구현한 것이다. 일반적으로 GenericServlet을 상속받아 사용하는 것보다 편리하다. HttpServlet은 클라이언트 요청에 따라 doGet, doPost, doPut 등의 메소드들을 호출한다. <그림 13>은 HttpServlet의 동작 흐름이다.

  
<그림 13> GET과 POST 요청을 다루는 HttpServlet

**WEB-INF 폴더**  
WEB-INF 폴더는 웹 애플리케이션에서 사용되는 자바 클래스와 설정 정보가 들어있는 폴더다. WEB-INF 폴더에는 classes 폴더와 lib 폴더가 포함돼 있다. classes 폴더에는 웹 애플리케이션에서 사용될 클래스와 서블릿들이 들어가며, lib 폴더에는 JAR 파일로 저장된 클래스 파일들이 들어간다.

  
<그림 14> WEB-INF 폴더 구조

**배포 서술자**  
배포 서술자는 서블릿 컨테이너가 서블릿을 실행하기 위한 정보를 가진 XML 파일이다. 배포 서술자는 web.xml 파일로 WEB-INF 폴더 아래에 있다. 서블릿을 등록하고 서블릿이 호출되는 URL과 URL에 해당하는 서블릿을 매핑할 수 있다. 배포 서술자는 서블릿뿐 아니라 filter, welcome page, session, context 등의 다양한 정보를 포함할 수 있다.   
  
배포 서술자에서 가장 많이 사용되는 URL 매핑법과 흐름을 살펴보자.

  
<그림 15> URL 매핑 흐름

|  |
| --- |
| **<리스트 7> web.xml의 URL 매핑 코드**  <servlet-mapping> <servlet-name>HelloServlet</servlet-name> <url-pattern>/hello</url-pattern> </servlet-mapping>  <servlet> <servlet-name>HelloServlet</servlet-name> <servlet-class>com.javacafe.helloservlet.HelloServlet</servlet-class> </servlet> |

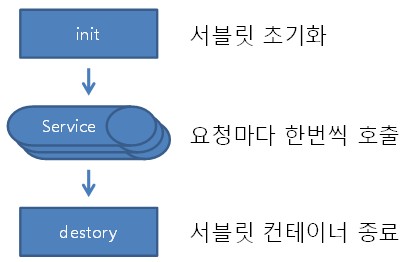
**01.** 클라이언트에서 http://localhost:8080/helloservlet /hello를 요청한다.

**02.** 컨테이너는 배포 서술자 <url-pattern> 항목에서 /hello를 찾는다.

**03.** 배포 서술자에서 <url-pattern> 항목과 쌍을 이루는 <servlet-name>을 반환한다.

**04.** 반환된 <servlet-name>을 가지고 <servlet-class>를 찾아 서블릿 클래스를 실행한다.

**서블릿 라이프사이클**  
HelloServlet 실습으로 클라이언트에서 요청한 정보를 처리하고 응답하는 흐름을 확인해 봤다. 이제 서블릿이 어떻게 실행되고 어떻게 종료되는지 알아보기 위해 서블릿 라이프사이클을 살펴본다. 호출되는 메소드에 대한 간단한 설명과 방문자 카운트 예제 코드를 만들어 전체적인 흐름을 파악해보자.  
  
서블릿 라이프사이클은 최초 실행하면 <그림 16>과 같이 init 메소드가 호출되고 초기화된 후 service 메소드를 호출해 요청을 처리한다. 서블릿이 더 이상 필요 없을 경우 destroy 메소드가 호출된다.  
  
init 메소드와 destroy 메소드가 필요하지 않아도 오버라이딩할 필요는 없다.

  
<그림 16> 서블릿 라이프 사이클

● init  
init 메소드의 호출 시점은 서블릿 컨테이너 구현 방법에 따라 조금씩 다를 수 있다. Apache Tomcat에서 다른 설정을 하지 않으면 서블릿 인스턴스 생성 후, 생성된 서블릿이 처음으로 요청을 받아 init 메소드를 호출한다. init 메소드에서는 서블릿 초기화 작업을 수행한다. init 메소드에서 초기화에 필요한 환경변수가 있다면 getInitParameter 메소드로 web.xml에 정의된 환경변수를 읽어올 수 있다.  
  
<리스트 8>은 init 메소드가 호출되고 환경변수를 읽어오는 예제다.

|  |
| --- |
| **<리스트 8> 환경변수를 불러오는 예제 코드**  <servlet> <servlet-name>InitializeTest</servlet-name> <servlet-class>com.javacafe.InitializeTest</servlet-class>  <init-param> <param-name>version</param-name> <param-value>1.0</param-value> </init-param> </servlet> [web.xml] public class InitializeTest extends HttpServlet{ private String mVersion;  public void init() throws ServletException{ mVersion = getInitParameter("version"); }  protected void doGet(HttpServletRequestreq, HttpServletResponseresp) throws ServletException, IOException { resp.setContentType("text/html"); PrintWriter out = resp.getWriter(); out.println("<html>"); out.println("<head><title> InitializeTest</title></head>"); out.println("<body>"); out.println("version : " + mVersion); out.println("</body>"); out.println("</html>"); out.flush(); } } |

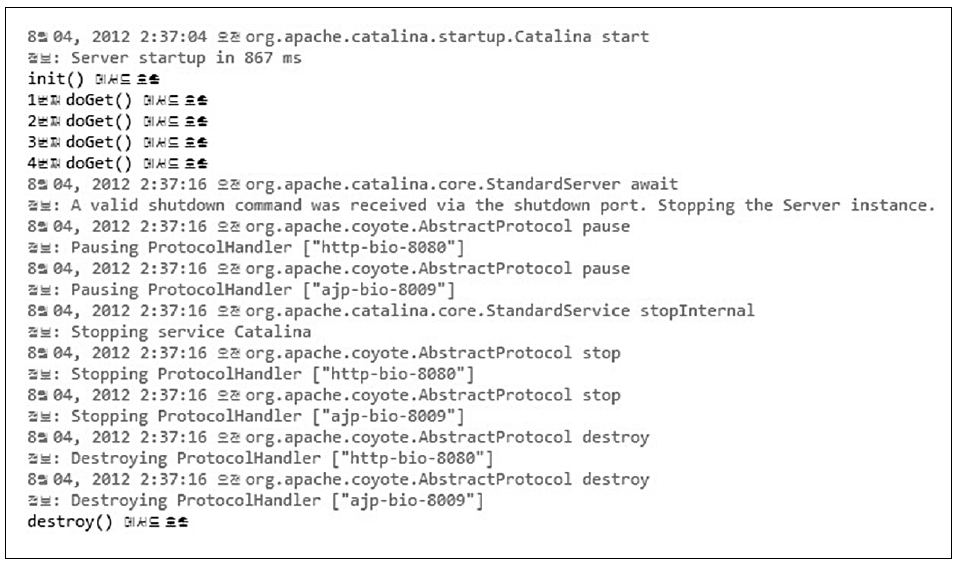
● service  
service 메소드는 클라이언트로 받은 요청을 처리하고 응답하는 메소드다. 클라이언트 요청마다 service 메소드가 한 번씩 실행된다. 서블릿이 HttpServlet을 상속받아 구현됐다면 service 메소드에서는 클라이언트 요청 방식(GET/POST)에 따라 GET 방식이면 doGet 메소드를 호출하고, POST 방식이면 doPost 메소드를 호출하게 된다.

● destory  
destroy 메소드는 init 메소드와 동일하게 서블릿 인스턴스 생성 후, 한 번만 실행되는 메소드로 서블릿 컨테이너가 종료될 때 호출된다. destroy 메소드가 호출되면 메모리를 정리하는 Garbage Collector가 동작하게 된다.

**서블릿 라이프사이클 실습**  
<리스트 9>로 예제의 흐름을 파악하며 서블릿 라이프사이클을 정확하게 이해했는지 확인해보자.

|  |
| --- |
| **<리스트 9> 서블릿 라이프사이클 예제 코드**  public class LifeCycleTest extends HttpServlet{ private intmCount; public void init() throws ServletException{ mCount = 0; System.out.println("init() 메소드 호출"); }  public void doGet(HttpServletRequestreq, HttpServletResponseresp) throws ServletException, IOException { mCount++; System.out.println(mCount + "번째 doGet() 메소드 호출"); } public void destroy() { System.out.println("destroy() 메소드 호출"); } } |

여러 번의 요청을 받아 처리하더라도 init 메소드와 destroy 메소드는 단 한 번만 실행되는 것을 확인할 수 있다. <그림 17>은 LifeCycleTest 서블릿을 네 번 호출한 후 서블릿 컨테이너를 종료시킨 로그다.

  
<그림 17> 서블릿 컨테이너 종료 후 로그

지금까지 웹 애플리케이션의 기초적인 부분과 서블릿 동작 흐름을 알아봤다. 이번 시간에 살펴본 내용들은 웹 프로그램을 하기 위한 기초 지식으로 앞으로 나올 JSP, Spring 등의 기술들을 이해하는 데 도움이 될 것이다. 서블릿이 발표되고 많은 시간이 흐르면서 서블릿 기반의 여러 기술들과 프레임워크들이 등장했다. 다음 시간에는 서블릿에서 좀더 발전한 기술인 JSP에 대한 내용을 알아본다.