**Chapter 5**

In this chapter we will talk about the technology specifications of Servlet in deep. So, let's get started.

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.The javax.servlet package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.The javax.servlet.http package contains interfaces and classes that are responsible for http requests only.

**Servlet Interface**

The Servlet interface defines the methods that must be implemented by all the servlets. In short, all servlet must implement Servlet interface directly or indirectly (To implement this interface, you can write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet).

**Servlet Interface methods**  
  
1) **void destroy()**: This method ends the life cycle of a servlet, it is called by Servlet container. This method is only called once all threads within the servlet’s service method have exited or after a timeout period has passed. Once destroy() method is called the servlet container does not call the service() method for that servlet.

2) **void init(ServletConfig config) throws ServletException**: This methods initializes the servlet. It is called by servlet container after instantiating the servlet.

3) **void service(ServletRequest req, ServletResponse res) throws ServletException, java.io.IOException**: It responds the requests that comes from client.

4) **ServletConfig getServletConfig()**: Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.

5) **java.lang.String getServletInfo()**: Returns information about the servlet, such as author, version, and copyright.

**GenericServlet Class**

GenericServlet class implements Servlet, ServletConfig and Serializable interfaces. It provides the implementation of all the methods of these interfaces except the service method.

GenericServlet class can handle any type of request so it is protocol-independent.

You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

GenericServlet code is mentioned below, for the better understanding.

public abstract class GenericServlet implements Servlet, ServletConfig,Serializable {

private static final long serialVersionUID = 1L;

private transient ServletConfig config;

public void destroy() {}

public String getInitParameter(String name) {

return getServletConfig().getInitParameter(name);

}

public Enumeration<String> getInitParameterNames() {

return getServletConfig().getInitParameterNames();

}

public ServletConfig getServletConfig() {

return config;

}

public ServletContext getServletContext() {

return getServletConfig().getServletContext();

}

public String getServletInfo() {

return "";

}

public void init(ServletConfig config) throws ServletException {

this.config = config;

init();

}

public void init() throws ServletException {

}

public void log(String msg) {

getServletContext().log(getServletName() + ": " + msg);

}

public void log(String message, Throwable t) {

getServletContext().log(getServletName() + ": " + message, t);

}

public abstract void service(ServletRequest paramServletRequest,

ServletResponse paramServletResponse) throws ServletException,

IOException;

public String getServletName() {

return config.getServletName();}}

**HttpServlet Class**

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

HttpServlet code is mentioned below, for the better understanding.

public abstract class HttpServlet extends GenericServlet {

private static final long serialVersionUID = 1L;

private static final String METHOD\_DELETE = "DELETE";

private static final String METHOD\_HEAD = "HEAD";

private static final String METHOD\_GET = "GET";

private static final String METHOD\_OPTIONS = "OPTIONS";

private static final String METHOD\_POST = "POST";

private static final String METHOD\_PUT = "PUT";

private static final String METHOD\_TRACE = "TRACE";

private static final String HEADER\_IFMODSINCE = "If-Modified-Since";

private static final String HEADER\_LASTMOD = "Last-Modified";

private static final String LSTRING\_FILE = "javax.servlet.http.LocalStrings";

private static ResourceBundle lStrings = ResourceBundle

.getBundle("javax.servlet.http.LocalStrings");

protected void doGet(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

String protocol = req.getProtocol();

String msg = lStrings.getString("http.method\_get\_not\_supported");

if (protocol.endsWith("1.1")) {

resp.sendError(405, msg);

} else {

resp.sendError(400, msg);

}

}

protected long getLastModified(HttpServletRequest req) {

return -1L;

}

protected void doHead(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

if (DispatcherType.INCLUDE.equals(req.getDispatcherType())) {

doGet(req, resp);

} else {

NoBodyResponse response = new NoBodyResponse(resp);

doGet(req, response);

response.setContentLength();

}

}

protected void doPost(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

String protocol = req.getProtocol();

String msg = lStrings.getString("http.method\_post\_not\_supported");

if (protocol.endsWith("1.1")) {

resp.sendError(405, msg);

} else {

resp.sendError(400, msg);

}

}

protected void doPut(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

String protocol = req.getProtocol();

String msg = lStrings.getString("http.method\_put\_not\_supported");

if (protocol.endsWith("1.1")) {

resp.sendError(405, msg);

} else {

resp.sendError(400, msg);

}

}

protected void doDelete(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

String protocol = req.getProtocol();

String msg = lStrings.getString("http.method\_delete\_not\_supported");

if (protocol.endsWith("1.1")) {

resp.sendError(405, msg);

} else {

resp.sendError(400, msg);

}

}

private static Method[] getAllDeclaredMethods(Class<?> c) {

if (c.equals(HttpServlet.class)) {

return null;

}

Method[] parentMethods = getAllDeclaredMethods(c.getSuperclass());

Method[] thisMethods = c.getDeclaredMethods();

if ((parentMethods != null) && (parentMethods.length > 0)) {

Method[] allMethods = new Method[parentMethods.length

+ thisMethods.length];

System.arraycopy(parentMethods, 0, allMethods, 0,

parentMethods.length);

System.arraycopy(thisMethods, 0, allMethods, parentMethods.length,

thisMethods.length);

thisMethods = allMethods;

}

return thisMethods;

}

protected void doOptions(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

Method[] methods = getAllDeclaredMethods(getClass());

boolean ALLOW\_GET = false;

boolean ALLOW\_HEAD = false;

boolean ALLOW\_POST = false;

boolean ALLOW\_PUT = false;

boolean ALLOW\_DELETE = false;

boolean ALLOW\_TRACE = true;

boolean ALLOW\_OPTIONS = true;

for (int i = 0; i < methods.length; i++) {

Method m = methods[i];

if (m.getName().equals("doGet")) {

ALLOW\_GET = true;

ALLOW\_HEAD = true;

}

if (m.getName().equals("doPost"))

ALLOW\_POST = true;

if (m.getName().equals("doPut"))

ALLOW\_PUT = true;

if (m.getName().equals("doDelete")) {

ALLOW\_DELETE = true;

}

}

String allow = null;

if (ALLOW\_GET)

allow = "GET";

if (ALLOW\_HEAD)

if (allow == null)

allow = "HEAD";

else

allow = allow + ", HEAD";

if (ALLOW\_POST)

if (allow == null)

allow = "POST";

else

allow = allow + ", POST";

if (ALLOW\_PUT)

if (allow == null)

allow = "PUT";

else

allow = allow + ", PUT";

if (ALLOW\_DELETE)

if (allow == null)

allow = "DELETE";

else

allow = allow + ", DELETE";

if (ALLOW\_TRACE)

if (allow == null)

allow = "TRACE";

else

allow = allow + ", TRACE";

if (ALLOW\_OPTIONS) {

if (allow == null)

allow = "OPTIONS";

else

allow = allow + ", OPTIONS";

}

resp.setHeader("Allow", allow);

}

protected void doTrace(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

String CRLF = "\r\n";

StringBuilder buffer = new StringBuilder("TRACE ")

.append(req.getRequestURI()).append(" ")

.append(req.getProtocol());

Enumeration<String> reqHeaderEnum = req.getHeaderNames();

while (reqHeaderEnum.hasMoreElements()) {

String headerName = (String) reqHeaderEnum.nextElement();

buffer.append(CRLF).append(headerName).append(": ")

.append(req.getHeader(headerName));

}

buffer.append(CRLF);

int responseLength = buffer.length();

resp.setContentType("message/http");

resp.setContentLength(responseLength);

ServletOutputStream out = resp.getOutputStream();

out.print(buffer.toString());

out.close();

}

protected void service(HttpServletRequest req, HttpServletResponse resp)

throws ServletException, IOException {

String method = req.getMethod();

if (method.equals("GET")) {

long lastModified = getLastModified(req);

if (lastModified == -1L) {

doGet(req, resp);

} else {

long ifModifiedSince;

try {

ifModifiedSince = req.getDateHeader("If-Modified-Since");

} catch (IllegalArgumentException iae) {

ifModifiedSince = -1L;

}

if (ifModifiedSince < lastModified / 1000L \* 1000L) {

maybeSetLastModified(resp, lastModified);

doGet(req, resp);

} else {

resp.setStatus(304);

}

}

} else if (method.equals("HEAD")) {

long lastModified = getLastModified(req);

maybeSetLastModified(resp, lastModified);

doHead(req, resp);

} else if (method.equals("POST")) {

doPost(req, resp);

} else if (method.equals("PUT")) {

doPut(req, resp);

} else if (method.equals("DELETE")) {

doDelete(req, resp);

} else if (method.equals("OPTIONS")) {

doOptions(req, resp);

} else if (method.equals("TRACE")) {

doTrace(req, resp);

} else {

String errMsg = lStrings.getString("http.method\_not\_implemented");

Object[] errArgs = new Object[1];

errArgs[0] = method;

errMsg = MessageFormat.format(errMsg, errArgs);

resp.sendError(501, errMsg);

}

}

private void maybeSetLastModified(HttpServletResponse resp,

long lastModified) {

if (resp.containsHeader("Last-Modified"))

return;

if (lastModified >= 0L) {

resp.setDateHeader("Last-Modified", lastModified);

}

}

public void service(ServletRequest req, ServletResponse res)

throws ServletException, IOException {

HttpServletRequest request;

HttpServletResponse response;

try {

request = (HttpServletRequest) req;

response = (HttpServletResponse) res;

} catch (ClassCastException e) {

throw new ServletException("non-HTTP request or response");

}

service(request, response);}}

NOTE : Why HttpServlet class is declared as abstract even there is no abstract method in that class?

Probably, to prevent direct instantiation of HttpServlet by creating its instance. This is done as it follows the Template Method design pattern. The doXXX() methods have all the default behaviors of returning a HTTP 405 Method Not Implemented error. If those methods were all abstract you would be forced to override them all, even though your business requirements don't need it all. It would only result in boilerplate code and unspecified behaviour.

Let's look at the Hierarchy of Servlet APIS

