**HISTORY OF JAVA WEB SERVLET and JavaServer Pages (JSP)**

* Java is built naturally with internet functionality. In **June 1997**, Sun Microsystems announced the Java Servlet.
* Java Servlet is one of the fundamental building block in developing different components of a Java Web specifically mainstream server-side Java which runs in a single process by using grain threads.
* While in **1998,** Sun released the JavaServer Pages **(JSP).**
* JSP is used to easily code a dynamic content of the web’s HTML pages.
* Both Servlets and JSPs lets the programmers build or develop a portable, easy to maintain, modular and scalable Web applications.

**THE JAVA SERVLET**

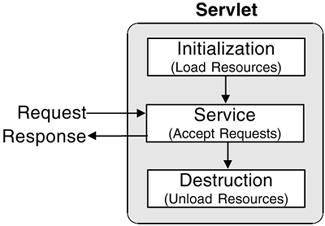
* In web applications, servlets were the first one which got the full access and power of Java. Wherein just like applets, it is completely and fully written in Java programming language.
* It is used for extending the server’s capability to host different applications that are accessed through a programming model, the request-response model. Which means it takes the request of the client and generates the response to the request.
* Java Servlet’s version 2.4 (current) which is included in the Java 2 Enterprise Edition (J2EE). Through the Tomcat project, it is available for free and also an open source.

**Features of Servlet 2.4:**

* Web Applications: In this section, servlet is always part of it (the web application), wherein it provides all the resources of a website.

**Servlet Container** – it manages all the Servlets in the basis of Web application.

*The web container facilitates the conversion to and from the HTTP request/response message to HTTP Servlet Request/HTTP Servlet Response*.

* Servlets and HTTP Servlets: With the support of HTTP, Servlet is used to provide a dynamic web pages and also plays an essential mechanism in Client/Server model.
* Filters: Used in authentication, logging and compression and it is an abstracted method that manipulates the request and response of the client before the request ends.
* Security: Servlets used the security that has been provided by JVM but also provides a way to control access of resources in the web application.
* Internalization: One of the best feature of a Servlet is that it can be develop content using variety of different languages by means of Servlet API.
* Servlet Life Cycle

1. *Instantiation:*

* The web container created the instance only once in the cycle.

1. *Initialization:* **init() method is invoked**

* This phase represents the creation of different resources to service requests. The init() method is invoked only once and before servicing of request takes place, the servlet invokes it first.

Syntax:

public void init(ServletConfig config) throws ServletException

1. *Request Handling:* **service() method is invoked**

* This phase represents all requests interactions invoked by each client.

Two parameters of service() method: Represents client’s request and Servlets’s client’s response.

1. javax.servlet.ServletRequest
2. javax.servlet.ServletResponse

Syntax:

public void service(ServletRequest request, ServletResponse response)

  throws ServletException, IOException

1. *Destruction:* **destroy() method is invoked**

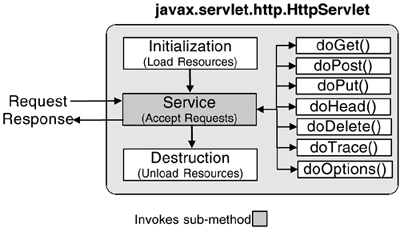
* Represents when the Servlet is removed in a container, simply, the destruction of a life cycle phase. The container calls the method destry() and terminate the resources that have been created.

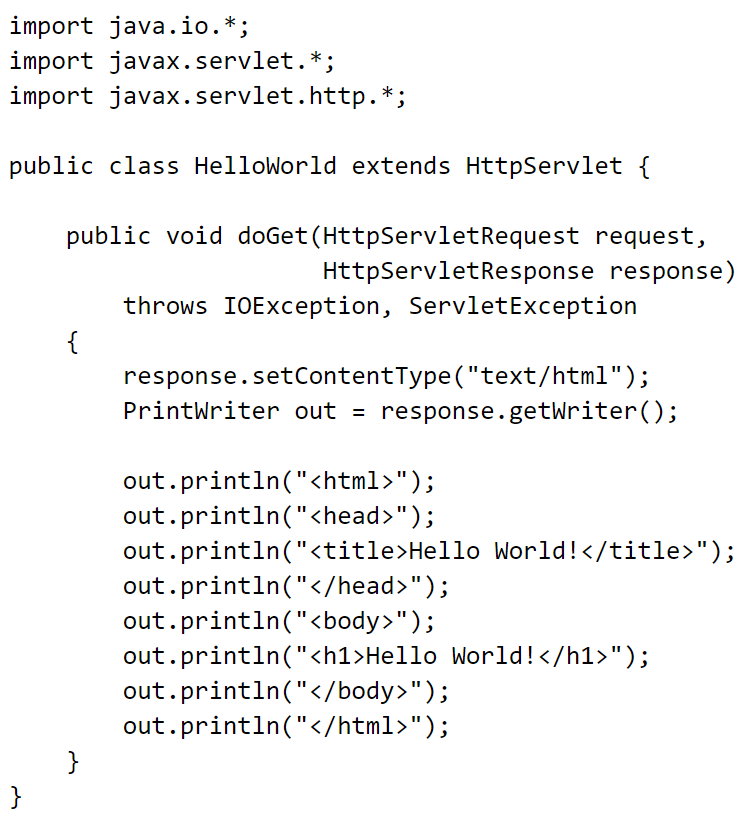
Syntax:

public void destroy()

* The HttpServlet object also uses the Servlet Life Cycle yet there are some modifications when it comes to HTTP protocol.
* During the phase of the service request, one of the seven helper methods is called which is appropriate to the type of HTTP request, named as follows:

1. doGet()
2. doPost()
3. doPut()
4. doHead()
5. doOptions()
6. doDelete()
7. doTrace()

The HttpServlet Life Cycle

* The figure below is a sample code for HttpServlet which will generate a simple HTML page.
* When deploying a servlet, it is not the case that it is written fully in java. For the client to access the Servlet, a unique URL is declared within the Web Application Deployment Descriptor which is the web.xml.
* The web.xml relies on the following two new elements:

1. servlet element – loaded by the web application to define the a Servlet.
2. Sevlet-mapping element – map the Servlet in a URL with the following servlet specification:
3. **An exact pattern to match.** The pattern must start with a /, but can contain anything afterwards. This type of pattern is used for a one-to-one mapping of a request to a specific Servlet.
4. **An extension match, \*.extension.** In this case all URLs ending with the given extension are forwarded to the specified Servlet. This is commonly used in Servlet frameworks and can force many requests to go to the same Servlet[15](javascript:popUp('/content/images/chap2_0321136497/elementLinks/ch02fn15.html')).
5. **A path mapping.** Path mappings must start with a / and end with a /\*. In between anything can appear. Path mappings are usually used for forwarding all requests that fall in a certain directory to a specific Servlet.
6. **Default Servlet, /.** A default Servlet mapping is used to define a Servlet for forwarding requests when no path information is given. This is analogous to a directory listing.

## **Servlet Configuration**

* Configuration information for a Servlet may consist of a string or a set of string values included in the Servlet's web.xml declaration. This functionality allows a Servlet to have initial parameters specified outside of the compiled code and changed without needing to recompile the Servlet.
* Each servlet has an object associated with it called the ServletConfig[19](javascript:popUp('/content/images/chap2_0321136497/elementLinks/ch02fn19.html')). This object is created by the container and implements the javax.servlet.ServletConfig interface.
* It is the ServletConfig that contains the initialization parameters. A reference to this object can be retrieved by calling the getServletConfig() method. The ServletConfig object provides the following methods for accessing initial parameters:
* getInitParameter(String name)- returns a String object that contains the value of the named initialization parameter or null if the parameter does not exist.
* getInitParameterNames() - returns the names of the Servlet's initialization parameters as an Enumeration of String objects or an empty Enumeration if the Servlet has no initialization parameters.

**Limitations of Configuration: web.xml Additions**

Initial parameters are a good method of providing simple one-string values that Servlets can use to configure themselves. This approach is simple and effective, but is a limited method of configuring a Servlet. For more complex Servlets it is not uncommon to see a completely separate configuration file created to accompany web.xml. When developing Servlets, keep in mind that nothing stops you from doing this. If the parameter name and parameter values mappings are not adequate, do not use them! It is perfectly OK to create a custom configuration file and package it in a WAR with the rest of a Web Application.

Client/Server Servlet Programming

A Servlet request and response is represented by the javax.servlet.ServletRequestand javax.servlet.ServletResponse objects, or a corresponding subclass of them. For HTTP Servlets the corresponding classes are HttpServletRequest and HttpServletResponse. These two objects were quickly introduced with the HelloWorld Servlet example, but the example was primarily focused on showing how a Servlet is deployed for use. Coding and deploying are the fundamental parts of Servlet development. Deployment was explained first because it is the exact same process for any given Servlet. Once explained it is a fairly safe assumption that you can repeat the process or simply copy and edit what already exists. Servlet code varies greatly depending on what the Servlet are designed to do.

**HttpServletRequest and HttpServletResponse**

The Servlet API makes manipulating an HTTP request and response pair relatively simple through use of the HttpServletRequest and HttpServletResponse objects. Both of these objects encapsulate a lot of functionality. Do not worry if it seems like this section is skimming through these two objects. Detailing all of the methods and members would be both tedious and confusing without understanding the rest of the Servlet API, but API discussion has to start somewhere and these two objects are arguably the most important. In this section discussion will only focus on a few of the most commonly used methods of each object. Later chapters of the book cover the other methods in full and in the context of which they are best used.

**HttpServletResponse**

The first and perhaps most important functionality to discuss is how to send information back to a client. As its name implies, the HttpServletResponse object is responsible for this functionality. By itself the HttpServletResponse object only produces an empty HTTPresponse. Sending back custom content requires using either the getWriter() or getOutputStream() method to obtain an output stream for writing content. These two methods return suitable objects for sending either text or binary content to a client, respectively. Only one of the two methods may be used with a given HttpServletResponse object. Attempting to call both methods causes an exception to be thrown.

**Table 1-1. HTML Markup from HelloWorld Servlet**

| **Generated Markup** | **HelloWorld.java** |
| --- | --- |
| <html> | out.println("<html>"); |
| <head> | out.println("<head>"); |
| <title>Hello World!</title> | out.println("<title>Hello World!</title>"); |
| </head> | out.println("</head>"); |
| <body> | out.println("</head>"); |
| <h1>Hello World!</h1> | out.println("<h1>Hello World!</h1>"); |
| </body> | out.println("</body>"); |
| </html> | out.println("</html>"); |

#### Response Headers

The HttpServletResponse object includes the following methods for manipulating *HTTP*response headers:

* **addHeader(java.lang.String name, java.lang.String value)**: The addHeader()method adds a response header with the given name and value. This method allows response headers to have multiple values.
* **containsHeader(java.lang.String name)**: The containsHeader() method returns a boolean indicating whether the named response header has already been set.
* **setHeader(java.lang.String name, java.lang.String value)**: The setHeader()method sets a response header with the given name and value. If the header had already been set, the new value overwrites the previous one. The containsHeader() method can be used to test for the presence of a header before setting its value.
* **setIntHeader(java.lang.String name, int value)**: The setIntHeader() sets a response header with the given name and integer value. If the header had already been set, the new value overwrites the previous one. The containsHeader() method can be used to test for the presence of a header before setting its value.
* **setDateHeader(java.lang.String name, long date)**: The setDateHeader()sets a response header with the given name and date value. The date is specified in terms of milliseconds since the epoch. If the header had already been set, the new value overwrites the previous one. The containsHeader()method can be used to test for the presence of a header before setting its value.
* **addIntHeader(java.lang.String name, int value)**: The addIntHeader()method adds a response header with the given name and integer value. This method allows response headers to have multiple values.
* **addDateHeader(java.lang.String name, long date)**: The addDateHeader()method adds a response header with the given name and date value. The date is specified in terms of milliseconds since the epoch[22](javascript:popUp('/content/images/chap2_0321136497/elementLinks/ch02fn22.html')). This method doesn't override previous response headers and allows response headers to have multiple values.