**Servlets: Introduction to Servlets**

Simon Xu

---

**Contents**

- Servlet Overview
- First Servlet Program
- Servlet Life Cycle
- Browser/Servlet Communication
- Servlet Session
- Other advanced Topics

---

**What is a Servlet?**

- Java’s answer to the Common Gateway Interface (CGI).
- Applet: a java program that runs within the web browser.
- Servlet: a java program that runs within the web server.
  - Rapidly becoming the standard for building web applications.

---

**Life of a Servlet**

Regardless of the application, servlets usually carry out the following routine:

1. Read any data sent by the user
   - Capture data submitted by an HTML form.

2. Look up any HTTP information
   - Determine the browser version, host name of client, cookies, etc.

---

**Diagram**

- Web Browser
- Web Server
- Java Servlet
- Database
Life of a Servlet

3. Generate the Results
   - Connect to databases, connect to legacy applications,

4. Format the Results
   - Generate HTML on the fly

5. Set the Appropriate HTTP headers
   - Tell the browser the type of document being returned or set any cookies.

6. Send the document back to client

What can we build with Servlets?
- Search Engines
- Personalization Systems
- E-Commerce Applications
- Shopping Carts
- Product Catalogs
- Intranet Applications

Server Side Options
- There are many options for creating server side applications.
  - We will examine CGI briefly only.
  - This better enables us to understand servlets within the broader context of web development.
  - Also enables us to better understand the advantages and disadvantages of servlets.

Server Side Options
- Common Gateway Interface (CGI)
- Fast CGI
- Mod Perl
- Server Extensions
  - NSAPI
  - ISAPI
- ASP
- PHP
- Cold Fusion
Common Features

- All server side frameworks share a common set of features:
  - Read data submitted by the user
  - Generate HTML dynamically based on user input
  - Determine information about the client browser
  - Access Database systems
  - Exploit the HTTP protocol

Decision Points

- When evaluating which server side framework to use, you need to consider a number of critical factors:
  - Ease of development:
    - How easily can you build new applications?
  - Performance:
    - How fast can the framework respond to queries?
  - Scalability:
    - Can the framework scale to thousands, millions of users?
  - Security:
    - Are there any inherent security vulnerabilities?

CGI

- Represents one of the earliest, practical methods for generating web content.
- Primarily written in the Perl programming language.
- Unfortunately, traditional CGI programs suffer from scalability and performance problems.
- Let’s examine these two problems…

CGI Architecture

- For each browser request, the web server must spawn a new operating system process.
- Creating a new operating system process for each request takes time and memory.
- Hence, traditional CGI programs have inherent performance and scalability problems.
- Every other server architecture tries to address these problems.
Advantages of Servlets

- Servlets have six main advantages:
  - Efficient
  - Convenient
  - Powerful
  - Portable
  - Secure
  - Inexpensive

Advantage 1: Efficient

- For each browser request, the servlet spawns a light weight thread.
- This is faster and more efficient than spawning a new operating system process.
- Hence, servlets have better performance and better scalability than traditional CGI.

Servlets Architecture

- For each browser request, the web server only spawn a new thread, not a new operating system process.

Advantage 2: Convenient

- Servlets include built-in functionality for:
  - Reading HTML form data
  - Handling cookies
  - Tracking user sessions
  - Setting HTTP headers
  - Java is object oriented

Advantage 3: Powerful

- Servlets can talk directly to the web servers.
- Multiple servlets can share data:
  - Particularly important for maintaining database connections.
  - Includes powerful techniques for tracking user sessions.

Advantage 4: Portable

- One of the advantages of Java is its portability across different operating systems.
- Servlets have the same advantages.
- You can therefore write your servlets on Windows, then deploy them on UNIX.
- You can also run any of your servlets on any Java-enabled web server, with no code changes.
**Advantage 5: Secure**
- Traditional CGI programs have a number of known security vulnerabilities.
- Hence, you usually need to include a separate Perl/CGI module to supply the necessary security protection.
- Java has a number of built-in security layers.
- Hence, servlets are considered more secure than traditional CGI programs.

**Advantage 6: Inexpensive**
- You can download free servlet kits for development use.
- You can therefore get started for free!
- Nonetheless, production strength servlet web servers can get quite expensive.

**Review of Servlet Introduction**
- Servlets: a Java program that runs within the web server.
- The simple life of Servlets
- The applications using servlets
- Advantages of Servlets

**Servlet Template**
- First, let’s take a look at a generic servlet template.
- All your future templates will follow this general structure.
- The most important pieces are noted in Red.
Generic Template

- Import the Servlet API:
  
  ```java
  import javax.servlet.*;
  import javax.servlet.http.*;
  ```

- To create servlets, you must remember to always use these two import statements.

All your servlets must extend `HTTPServlet`.
`HTTPServlet` represents the base class for creating Servlets within the Servlet API.
Once you have extended `HTTPServlet`, you must override one or both:
- `doGet`: to capture HTTP Get Requests
- `doPost`: to capture HTTP Post Requests

**doGet  doPost**

- `doGet` and `doPost` methods each take two parameters:
  `HttpServletRequest`: encapsulates all information
  `HttpServletResponse`: encapsulate all information
  servlet response.

  * Form data, client host name, HTTP request headers.
  * HTTP Return status, outgoing cookies, HTML response.

  servlet to handle both GET or vice versa.

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTemplate extends HttpServlet {
  public void doGet(HttpServletRequest request,
   HttpServletResponse response)
   throws ServletException, IOException {
    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (e.g. data the user
    // entered and submitted).
    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).
    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }

  public void doPost(HttpServletRequest request,
   HttpServletResponse response)
   throws ServletException, IOException {
    // Use "request" to read incoming HTTP headers
    // (e.g. cookies) and HTML form data (e.g. data the user
    // entered and submitted).
    // Use "response" to specify the HTTP response status
    // code and headers (e.g. the content type, cookies).
    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
}
```
Getting an OutputStream

- The HttpServletResponse object has a getWriter() method.
- This method returns a java.io.PrintWriter object for writing data out to the Web Browser.

```java
PrintWriter out = response.getWriter();
```

Output Stream

- Once you have an OutputStream object, you just call the println() method to output to the browser.
- Anything you print will display directly within the web browser.
- As we will now see, you can also output any HTML tags.

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hello World");
    }
}
```

Generating HTML

- To generate HTML you need to add two steps:
  - Tell the browser that you are sending back HTML.
  - Modify the println() statements to return valid HTML.

Hello World!

- We are finally ready to see our first real servlet.
- This servlet outputs “Hello World!” as plain text, not HTML.
- Let’s take a look at the code, and then see the servlet in action.

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>
" + "<HEAD><TITLE>Hello World</TITLE></HEAD>
" + "<BODY>
" + "<H1>Hello World</H1>
" + "</BODY></HTML>");
    }
}
```

HelloWorld.java

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>
" + "<HEAD><TITLE>Hello World</TITLE></HEAD>
" + "<BODY>
" + "<H1>Hello World</H1>
" + "</BODY></HTML>");
    }
}
```
Generating HTML

- To return HTML, you must set the content MIME type to text/html:
  
  ```java
  response.setContentType("text/html");
  ```
- Remember that you must set the content type before you output any content.
- Once you have set the MIME type, you can return any HTML document you want.

How to compile your program?

- Hence, you will need to do development on your own machines.
- This is fairly straightforward, but it takes some time.

The Software

- To develop servlets, you will need three pieces of software:
  
  - Text Pad: a simple, text editor.
  - Java 2 Software Development Kit (JDK), Version 1.3
  - Java Servlet Development Kit (JSDK)
    
    > Contains the Servlet Runner for running Servlets on your own machine.
  - You may use other replacement

Servlet Runner

1. Starting Servlet Runner

   - Open an MS-DOS Window
   - Go to the JSDK2.1 root directory: `CD c:\jsdk2.1`
   - Run the `startserver` command: `startserver`
   - By default, Servlet Runner will run on Port 8080
   - Open your web browser and go to: `http://localhost:8080`
   - In your browser, you should see an index page of Sample servlets. Click any one of the "Execute" links to run the servlet.

2. Stopping Servlet Runner

   - To stop Servlet Runner, run the `stopserver` command: `stopserver`

Review of First Servlet Program

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletTemplate extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<HTML>
      <HEAD><TITLE>Hello World</TITLE></HEAD>
      <BODY>
      <H1>Hello World</H1>
      </BODY></HTML>"};
}
```
**Contents**
- Servlet Overview
- First Servlet Program
- Servlet Life Cycle
- Servlet Communication
- Servlet Session
- Other advanced Topics

**Life of a Servlet**
- Birth: Create and initialize the servlet
  - Important method: `init()`
- Life: Handle 0 or more client requests
  - Important method: `service()`
- Death: Destroy the servlet
  - Important method: `destroy()`

**Birth: The init() method**
- The `init()` method is called when the servlet is first requested by a browser request.
- It is **not** called again for each request.
- Used for one-time initialization.
- No concurrency issues during servlet initialization.
- There are two versions of the `init()` method:
  - Version 1: takes no arguments
  - Version 2: takes a `servletConfig` object as an argument.

**Property Files**
- To understand the difference between the two `init()` options, you need to first understand property files.
- All Web Servers/Servlet Runners maintain a central properties file for storing constants.
- You can add your own properties here. For example:
  - Database settings, user names, passwords, URLs, etc.

**Servlet Runner**
- Our Servlet Runner maintains a properties file at:
  - `C:\jsdk2.1\examples\WEB-INF\servlets.properties`
- All initialization parameters go here.
- Remember that the location and format of the properties file is different for each web server.
- The next slide shows a sample `servlets.properties` file...

```plaintext
servlet.properties file
# $Id: servlets.properties,v 1.2 $
# Define servlets here
<servletname>.code=<servletclass>
<servletname>.initparams=<name=value>,<name=value>
snoop.code=SnoopServlet
snoop.initparams=initarg1=foo,initarg2=bar
```

**Example Initialization Parameters**
**Servlet.properties Rules**

- To add your own properties, you need to follow the `servlet.properties` rules.
- You first need to register your servlet within the property file:
  ```<servletname>.code=<servletclass>```

**Version 1: init() method**

- No parameters
- Used when the servlet does **not** need to read any property files.
- Here’s an example:
  ```
  public void init() throws ServletException {
      ...
  }
  ```

**ServletConfig Object**

- Provides access to the servlet properties file.
- Has a `getInitParameter()` method for retrieving specific properties.
- For example:
  ```
  String message = config.getParameter("message");
  String password = config.getInitParameter("password");
  ```

**Version 2: init() method**

- Used when the servlet needs to read from a property file.
- Here’s an example:
  ```
  public void init (ServletConfig config) throws ServletException {
      super.init (config);
      ...
  }
  ```

**Version 2: init() method Cont.**

- Let’s examine version 2 again:
  ```
  public void init (ServletConfig config) throws ServletException {
      super.init (config);
      ...
  }
  ```

- It is important to call `super.init()`. The `init()` method of the superclass registers the `ServletConfig` object so you can access it later.

**Servlet.properties Rules**

- You can then add your own properties:
  ```<servletname>.initparams=<name=value>,<name=value>```

- For example, the following registers the `Birth` servlet, and sets its password parameter to "bluemoon":
  ```
  Birth.code=Birth
  Birth.initparams=password=bluemoon
  ```
Example

- Let’s examine a simple example.
- This example uses the 2nd init() option.
- In this case, we have hard coded one parameter, and read one parameter from the properties file.
- Once initialized, this program echos out its initialization parameters.

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Birth extends HttpServlet {
    String projectCodeName;
    String password;
    // init() is called first
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        projectCodeName = new String("Xerces");
        password = config.getInitParameter("password");
    }
    // Handle an HTTP GET Request
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.println("Project Code Name:  "+projectCodeName);
        out.println("Password:  "+password);
        out.close();
    }
}

Password is retrieved from the properties file.
```

```properties
# Define servlets here
# <servletname >.code=<servletclass>
# <servletname >.initparams=<name=value>,<name=value>
birth.code=Birth
birth.initparams=password=bluemoon
```

Life of a Servlet---init()

- The first time a servlet is called, the Servlet is instantiated, and its init() method is called.
- Only one instance of the servlet is instantiated.
- This one instance handles all browser requests.

Life of a Servlet

- Birth: Create and initialize the servlet
  - Important method: init()
- Life: Handle 0 or more client requests
  - Important method: service()
- Death: Destroy the servlet
  - Important method: destroy()

Service() Method

- Each time the server receives a request for a servlet, the server spawns a new thread and calls the servlet’s service() method.
Let’s Prove it…

- To prove that only one instance of a servlet is created, let’s create a simple example.
- The Counter Servlet keeps track of the number of times it has been accessed.
- This example maintains a single instance variable, called count.
- Each time the servlet is called, the count variable is incremented.
- If the Server created a new instance of the Servlet for each request, count would always be 0!

```java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Counter extends HttpServlet {
    // Create an instance variable
    int count = 0;

    // Handle an HTTP GET Request
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        count++;
        out.println("Since loading, this servlet has "+ count + " times.");
        out.close();
    }
}
```

The Service Method

- By default the service() method checks the HTTP Header.
- Based on the header, service calls either doPost() or doGet().
- doPost and doGet is where you put the majority of your code.
- If your servlets needs to handle both get and post identically, have your doPost() method call doGet() or vice versa.

Thread Synchronization

- By default, multiple threads are accessing the same servlet at the same time.
- You therefore need to be careful to synchronize access to shared data.
- For example, what happens if two browsers request a stock trade for the same account at the same time.
- Synchronization is, however a large topic in itself, and I will skip it here.
- Nonetheless, there is an option called the SingleThreadModel…

SingleThreadModel Interface

- To prevent multi-threaded access, you can have your servlet implement the SingleThreadModel:
  ```java
  public class FormServlet extends HttpServlet implements SingleThreadModel {
      ...
  }
  ```
- This will guarantee that your servlet will only process one browser request at a time.
- It therefore addresses most synchronization issues.
- Unfortunately, this can result in severe slowing of performance.

Life of a Servlet

- Birth: Create and initialize the servlet
  ```java
  public void init() {
      ...
  }
  ```
- Life: Handle 0 or more client requests
  ```java
  public void service() {
      ...
  }
  ```
- Death: Destroy the servlet
  ```java
  public void destroy() {
      ...
  }
  ```
Before a server shuts down, it will call the servlet’s destroy() method. You can handle any servlet clean up here. For example:
- Updating log files.
- Closing database connections.
- Closing any socket connections.

Example: Death.java

This next example illustrates the use of the destroy() method.
- While alive, the servlet will say “I am alive!”.
- When the server is stopped, the destroy() method is called, and the servlet records its time of death in a “rip.txt” text file.

```java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Death extends HttpServlet {
    // Handle an HTTP GET Request
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException, ServletException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        out.println("I am alive!");
        out.close();
    }

    // This method is called when one stops the Java Web Server
    public void destroy() {
        try {
            FileWriter fileWriter = new FileWriter("rip.txt");
            Date now = new Date();
            String rip = "I was destroyed at: " + now.toString();
            fileWriter.write(rip);
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Example rip.txt file

I was destroyed at: Thu Aug 24 11:10:58 CDT 2000

Putting it all together
Now that we know all about the birth, life and death of a servlet, let’s put this knowledge together to create a persistent counter.

The `Counter.java` example we covered earlier has a big problem:
- When you restart the web server, counting starts all over at 0.
- It does not retain any persistent memory.

To create a persistent record, we can store the count value within a “counter.txt” file.

- `init()`: Upon start-up, read in the current counter value from counter.txt.
- `destroy()`: Upon destruction, write out the new counter value to counter.txt.

```java
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class CounterPersist extends HttpServlet {
    String fileName = "counter.txt";
    int count;
    public void init () {
        try {
            FileReader fileReader = new FileReader (fileName);
            BufferedReader bufferedReader = new BufferedReader (fileReader);
            String initial = bufferedReader.readLine();
            count = Integer.parseInt (initial);
        } catch (FileNotFoundException e) { count = 0; }
        catch (IOException e) { count = 0; }
        catch (NumberFormatException e) { count = 0; }
    }
    public void destroy() {
        try {
            FileWriter fileWriter = new FileWriter (fileName);
            String countStr = Integer.toString (count);
            fileWriter.write (countStr);
            fileWriter.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // Handle an HTTP GET Request
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        response.setContentType("text/plain");
        PrintWriter out = response.getWriter();
        count++;
        out.println("Since loading, this servlet has 
+ been accessed " + count + " times.");
        out.close();
    }
}
```

At Start-up, load the counter from file.
In the event of any exception, initialize count to 0.

When destroy() is called, store new counter variable back to counter.txt.

Review of Servlet Lifecycle
- Loading servlets using class loader
- Birth: Create and initialize the servlet, by overriding `init()`
- Life: Handle 0 or more client requests by invocation of method `service()`
- Death: Destroy the servlet by calling `destroy()` method
Questions?