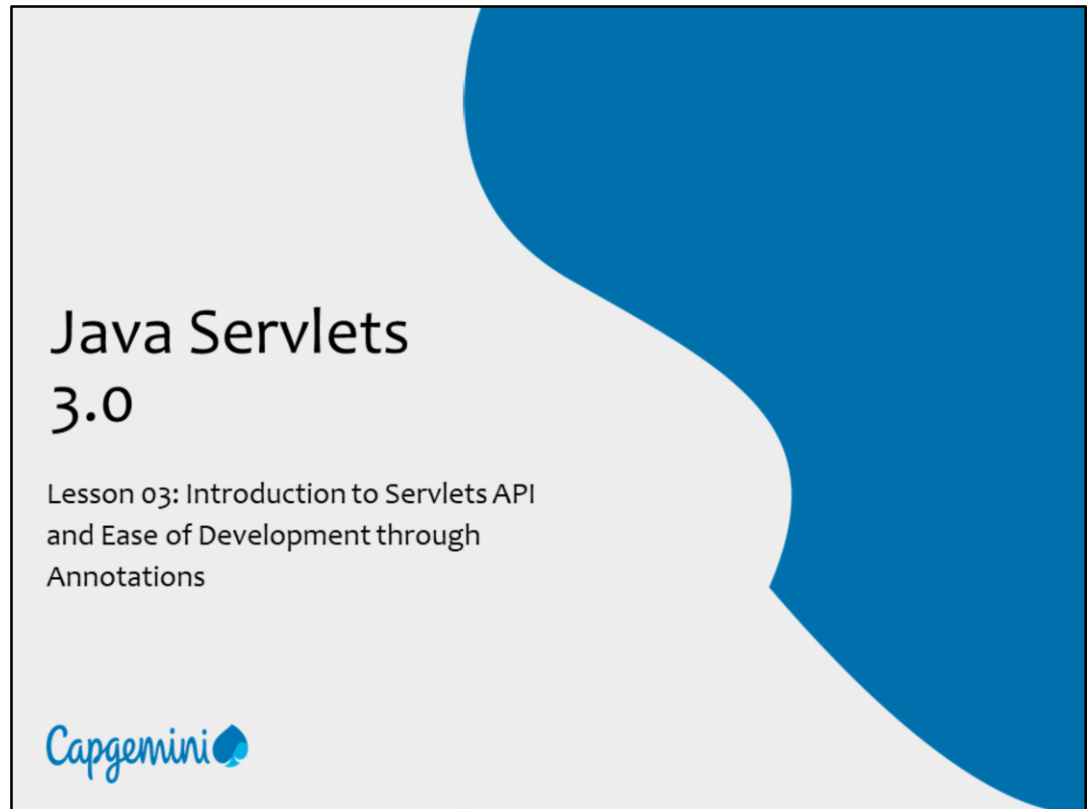


Instructor Notes:

Add instructor
notes here.



Instructor Notes:

Explain the
lesson coverage

Lesson Objectives

In this lesson, we will learn:

- Introduction to Servlet
- Role of Servlets in Web application design
- Advantages of Servlets
- Basic Servlet Architecture : Servlet Container
- Servlet Lifecycle
- Ease of Developing Servlets through Annotations
- Retrieving Information from HTML Page



Lesson Objectives:

This lesson introduces Servlets 3.0 and ease of development through Annotations ,
The lesson contents are:

- 3.1: Introduction to Servlet
- 3.2: Role of Servlets in Web Application Design
- 3.3: Advantages of Servlets
- 3.4: Basic Servlet Architecture: Servlet Container
- 3.5: Servlet Lifecycle
- 3.6: Ease of Developing Servlets through Annotations
- 3.7: Retrieving Information

Instructor Notes:

3.1: Introduction to Servlet What are Servlets?



Servlets are Java programs that extend the functionality of a Web server and capable of generating a dynamic response to a particular request using the HTTP Request / Response paradigm

Servlets are not tied to a specific client-server protocol but they are most commonly used with HTTP and the word "Servlet" is often used in the meaning of "HTTP Servlet"

- It is available and runs on all major web and application servers
- It is platform and server independent

Introduction to Servlet:

Java servlets are a key component of server-side Java Development. Servlets are modules of Java code that run in a server application to answer client requests. These are small, pluggable extensions to server that enhance the server's functionality. Servlets allow developers to extend and customize any Java-enabled server.

When servlets are used to generate dynamic content for a web page or otherwise extend the functionality of a web server, is like creating a web application!

Since servlets are written in the highly portable Java language and follow a standard framework, they provide a means to create sophisticated server extensions in a server and operating system independent way.

Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes. We shall see more on this in the coming sections.

Servlets first came on the scene around 1997. As of March 26 2010, the current version of the servlet specification is 3.0. The minimum platform requirement for Servlet 3.0 is JDK 1.6.

Instructor Notes:

Explain each of these points with an example.

- Example, for point #1 : based on book search, all books related to a particular category – say “Java” are returned in a neat tabular fashion
- Point # 2: For a registration form, user is sent an appropriate message like “successfully registered!” on successful registration.
- Point #3: Some kind of login servlet that authenticate user before providing access to any other resources on the web site
- Point# 4 : A search for a particular book, say servlets, will return a list of books from some database at server side...

3.2: Role of Servlets in Web Application Design

What Can Servlets Do?



Servlets can do the following functions:

- Dynamically build and return an HTML file based on nature of client request
- Process user input passed in an HTML form and return an appropriate response
- Provide user authentication and other security mechanisms
- Interact with server resources such as databases, other applications and network files to return useful information to the client
- Automatically attach web page design elements such as headers or footers, to all pages returned by server
- Forward requests from one server to another for load balancing purpose
- Manage state information on top of the stateless HTTP

Role of Servlets in Web Application Design:

Servlets can read explicit data sent in by the client in browser. This data can come from an HTML page into which user has entered data. Servlets can also read implicit request data which is sent by browser as part of request header.

Servlets can dynamically generate response and send content back to client. This process may involve talking to databases, executing another server-side component, and so on.

The response sent can be in form of pure HTML, plain text, XML, GIF images, or even as compressed data.

Instructor Notes:

Explain these points. Mention that Servlets being Java programs, all advantages of Java are applicable to Servlets too.

3.3: Advantages of Servlets

Advantages of Servlets

Servlets provide the following advantages:

- Crash Resistance
- Cross-Platform
- Cross-Server
- Durable
- Dynamically Loadable across the Network
- Extensible
- Multithreaded
- Protocol Independent
- Secure

Advantages of Servlets over CGI:

Servlets provide the following advantages:

Compiled: Servlets are compiled into Java byte-codes. This improves performance through compile-time code optimization. Server-side JIT compilers dramatically improve the performance of JVM.

Compilation also offers the advantages of strong error and type checking. Since many errors are flushed out during the compilation, servlets are more stable and easier to develop and debug.

Crash Resistance: The JVM does not allow servlets direct access to memory locations, thereby eliminating crashes that results from invalid memory accesses. In addition, before execution, the JVM verifies that compiled Java class files are valid and do not perform illegal operations. Finally, rather than crashing, the JVM will propagate an exception up the calling chain until it is caught. Thus, a poorly written or malicious servlet cannot crash the server.

Cross-Platform: Since servlets are written in Java, they enjoy the same cross platform support as any program. This “write once, run anywhere” capability allows servlets to be easily distributed throughout the enterprise without rewriting for each platform.

Cross-Server: Servlets can be run on virtually every popular web server that is in use today. More than a dozen software vendors currently provide native support for servlets within their products. For those servers that do not currently offer native servlet support, there are many third party add-ons that allow these servers to load and run servlets.

Instructor Notes:

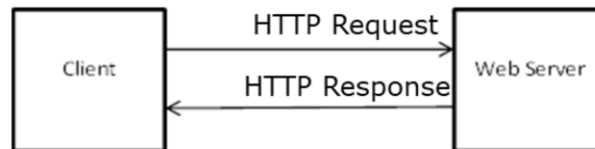
3.4: Basic Servlet Architecture : Servlet Container

Servlet Container



What is a Web Server?

- To know what is a Servlet container we need to know what is a Web Server first



A web server uses HTTP protocol to transfer data. In a simple situation, a user types in a URL (e.g. `www.servletdemos.com/static.html`) in browser (a client), and get a web page to read. So what the server does is sending a web page to the client. The transformation is in HTTP protocol which specifies the format of request and response message.

Since we using WildFly 8.x which is an Application server which will host the Servlet Container to handle the web components.

We could also use the Web server for the same purpose.

Since HTTP is a web oriented protocol, thus we have taken the term to be as "Web Server"

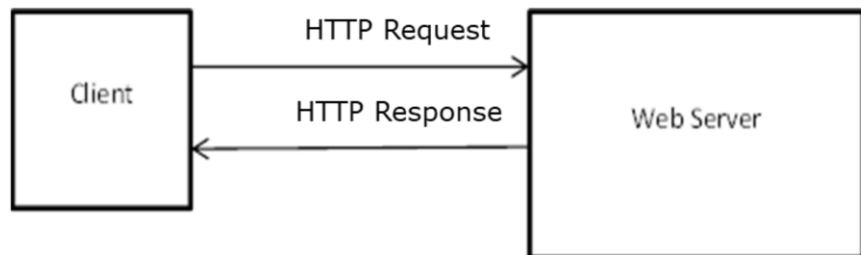
Instructor Notes:

3.4: Basic Servlet Architecture : Servlet Container

Servlet Container / Servlet Engine / Web Container

**What is a Servlet Container / Servlet Engine / Web Container**

- Client can request only static web page from Server. If client wants to read Web page based upon input (that requires processing), basic idea of servlet container is to dynamically generate the Web page on the server side
- Servlet container is essentially a part of a web server that interacts with the servlets. Servlet Container is container for servlets



When a request is sent, it is handled by the web server. If response is available within the web server it will be returned by the server to the client.

If response is not available and requires some processing then request will be processed by the Web Container.

The web container will then process the request and handover the response back to web server. The web server will then handover response back to client.

The web container delegates the request processing to web components like Servlets and JSP.

For example: If we request for a page as `www.gmail.com` , then same page is returned as response for all users. **This is a static content**

Whereas after entering your login credentials respective inbox details per user is retrieved as a response. **This is dynamic content**

Instructor Notes:

3.4: Basic Servlet Architecture : Servlet Container

Servlet Container / Servlet Engine / Web Container**Advantages of Servlet Container**

- Providing communication support between Web Components and Web Server
- Life cycle support for Web Components
- Networking support
- Enabling Web Security
- Multi threading support for Web Components

Providing communication support between Web Components and Web Server: Servlet as a web component does not communicate with Web Server directly. All the communication happens via Web container

Life cycle support for Web Components: Life cycle of Web components (Servlets and JSP) is managed by the container

Networking support: Web components do not need to open network and socket connections all is taken care by Web server

Enabling Web Security: Access to Web components can be secured by granting privileges.

Multi Threading support: Web components are multi-thread, for every request there is one thread generated. Thus all threads are managed by container. We do not need to do explicit multi-threading.

Instructor Notes:

Explain the Servlet hierarchy.

Explain how a protocol independent servlet can extend the functionality of any type of server including Web server, mail server etc

3.4: Basic Servlet Architecture : Servlet API

Servlet API

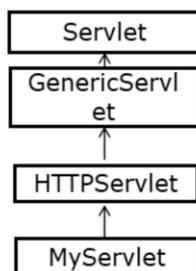


A servlet is an instance of a class which implements the `javax.servlet.Servlet` interface.

Servlet is web component inside Servlet Container

Most servlets extend one of the standard implementations of that interface, namely `javax.servlet.GenericServlet` or `javax.servlet.http.HttpServlet`.

In Servlet 3.0, there is `javax.servlet.annotation` package for creating servlets via annotations



Basic Servlet Architecture:

Servlet API:

Servlets use classes and interfaces from three packages: `javax.servlet`, `javax.servlet.http`.

A Servlet, in its most general form, is an instance of a class which implements the `javax.servlet.Servlet` interface. Most Servlets extend one of the standard implementations of that interface, namely `javax.servlet.GenericServlet` and `javax.servlet.http.HttpServlet`.

As of Servlet 3.0, make use of package `javax.servlet.annotation` to create Servlets. A protocol-independent servlet should subclass `GenericServlet`, while an HTTP servlet should subclass `HttpServlet`, which is itself a subclass of `GenericServlet`! (refer to the above figure).

Notice that the classes do not belong to the core Java API. They are extensions to the core API and hence `javax`!

Instructor Notes:

Explain the lifecycle methods. Concept of dispatching doXXX() methods which is done by server methods must be explained to participants in depth,

Thereby highlight the consequence of overriding service method

3.4: Basic Servlet Architecture : Servlet API

Servlet Interface Life Cycle Methods

init():

- It is executed once when the servlet is first loaded.

service():

- It is called in a new thread by server for each request.

destroy():

- It is called when server deletes servlet instance.

These lifecycle methods are implemented in `GenericServlet` class.

Basic Servlet Architecture:

Servlet Interface Life Cycle Methods:

The Servlet interface defines methods to initialize a servlet, to service requests, and to remove a servlet from the server. These are known as life-cycle methods and are called in the following sequence:

The `init()` method is guaranteed to be called only once during the Servlet's lifecycle. The Servlet performs one-time setup configurations in this method. It stores the `ServletConfig[*]` object so that it can be retrieved later by calling the Servlet's `getServletConfig()` method (This is handled by `GenericServlet`). The `ServletConfig` object contains Servlet parameters and a reference to the Servlet's `ServletContext[*]`.

`service()` method gets called every time a new request comes in. The method is called concurrently (that is, multiple threads may call this method at the same time) so it should be implemented in a thread-safe manner.

When servlet needs to be unloaded (for example: since a new version should be loaded or the server is shutting down), the `destroy()` method is called.

This method too is guaranteed to be called only once during the Servlet's lifecycle. All resources which were allocated in `init()` should be released in `destroy()`.

We shall be covering entire lifecycle of a typical servlet in the next section.

[*] : The `ServletConfig` and `ServletContext` interfaces are explained in the next slide.

Instructor Notes:

Explain the lifecycle of processing a request

3.4: Basic Servlet Architecture : Request Processing

Steps to process a Request



Web server receives HTTP request

Web server forwards the request to servlet container

The servlet is dynamically retrieved and loaded into the address space of the container, if it is not in the container

The container invokes the `init()` method of the servlet for initialization(invoked once when the servlet is loaded first time)

The container invokes the `service()` method of the servlet to process the HTTP request, i.e., read data in the request and formulate a response. The servlet remains in the container's address space and can process other HTTP requests

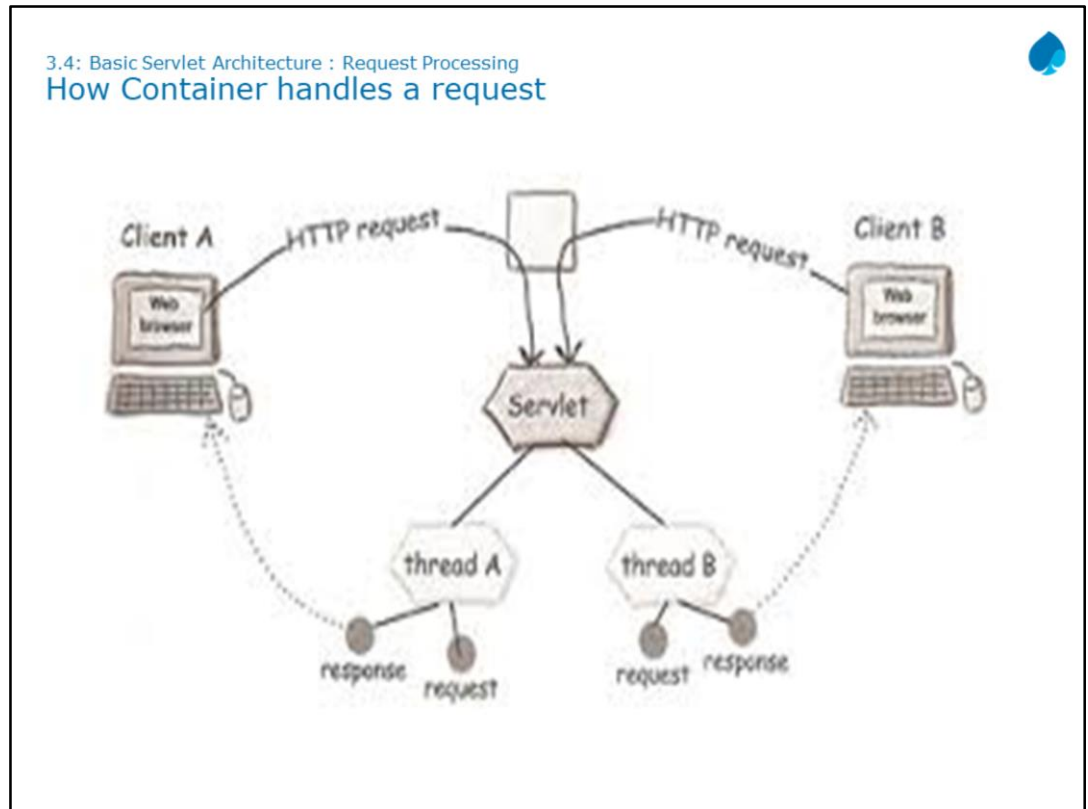
Web server return the dynamically generated results to the client

From the life cycle of a servlet object, we can see that servlet classes are loaded to container by class loader dynamically. Each request is in its own thread, and a servlet object can serve multiple threads at the same time. When it is no longer being used, it should be garbage collected by JVM.

Like any Java program, the servlet runs within a JVM. To handle the complexity of HTTP requests, the servlet container comes in. The servlet container is responsible for servlets' creation, execution and destruction.

Instructor Notes:

Explain the lifecycle of processing a request



Every request coming to the container from the client for a particular servlet will be treated as a separate thread.

Each thread will have its own copy of the request / response objects.

After the processing of a particular request is completed the threads would be garbage collected. But the servlet instance would still be there in memory.

Thus for processing multiple requests multiple threads are created.

Instructor Notes:

Explain the
GenericServlet

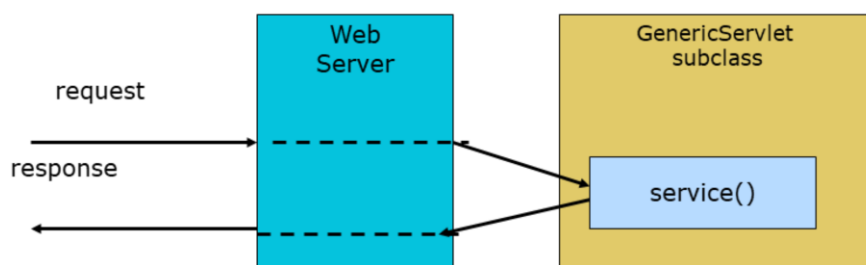
Explain the figure.

Also explain about
ServletConfig and
ServletContext
interfaces

3.4: Basic Servlet Architecture : Hierarchy GenericServlet class

`javax.servlet.GenericServlet`

- It is protocol independent
- It makes writing servlets easier
 - It provides simple versions of `init()` and `destroy()` and of the methods in the `ServletConfig` interface.
 - It also implements the `log` method, declared in the `ServletContext` interface.
- To write a generic servlet, override the abstract `service()`.



Basic Servlet Architecture:

GenericServlet:

All the lifecycle methods are implemented in `GenericServlet` class. Thus if `GenericServlet` class is extended to create a servlet, all the methods may be overridden to provide implementations. Optionally, just override the `service()` method to handle the request in the manner required!

ServletContext interface :

It defines a set of methods that a servlet uses to communicate with its servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.

There is one context per “web application” per JVM.

ServletContext attributes can be used to share information among a group of servlets.

The ServletContext object is contained within the ServletConfig object, which the Web server provides the servlet when the servlet is initialized.

ServletConfig interface:

This is a servlet configuration object used by a servlet container used to pass information to a servlet during initialization.

It is implemented by `GenericServlet`.

All of its initialization parameters can be set in deployment descriptor. Or else they can be passed via annotations. The ServletConfig parameters are specified for a particular servlet and are unknown to other servlets.

ServletContext and ServletConfig will be discussed in later sessions.

Instructor Notes:

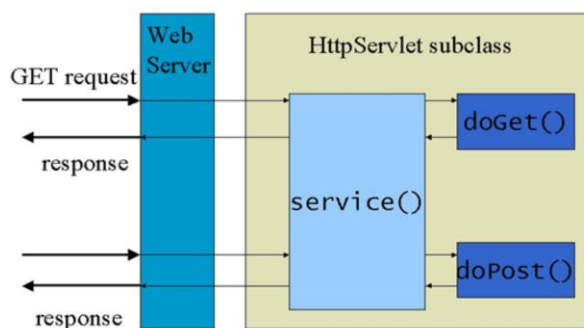
Explain the figure
and compare it vis-
à-vis generic servlet

3.4: Basic Servlet Architecture : Hierarchy

HttpServlet class

`javax.servlet.http.HttpServlet:`

- It has a built-in HTTP protocol support.
- Its subclass must override at least one of the following methods: `doGet()`, `doPost()`, `doHead()`, `doTrace()`, `doPut()`, `doDelete()`
- One must not override the `service()` method, since it dispatches a request to the different `doXXX()` methods for different HTTP requests.



Basic Servlet Architecture:

HttpServlet:

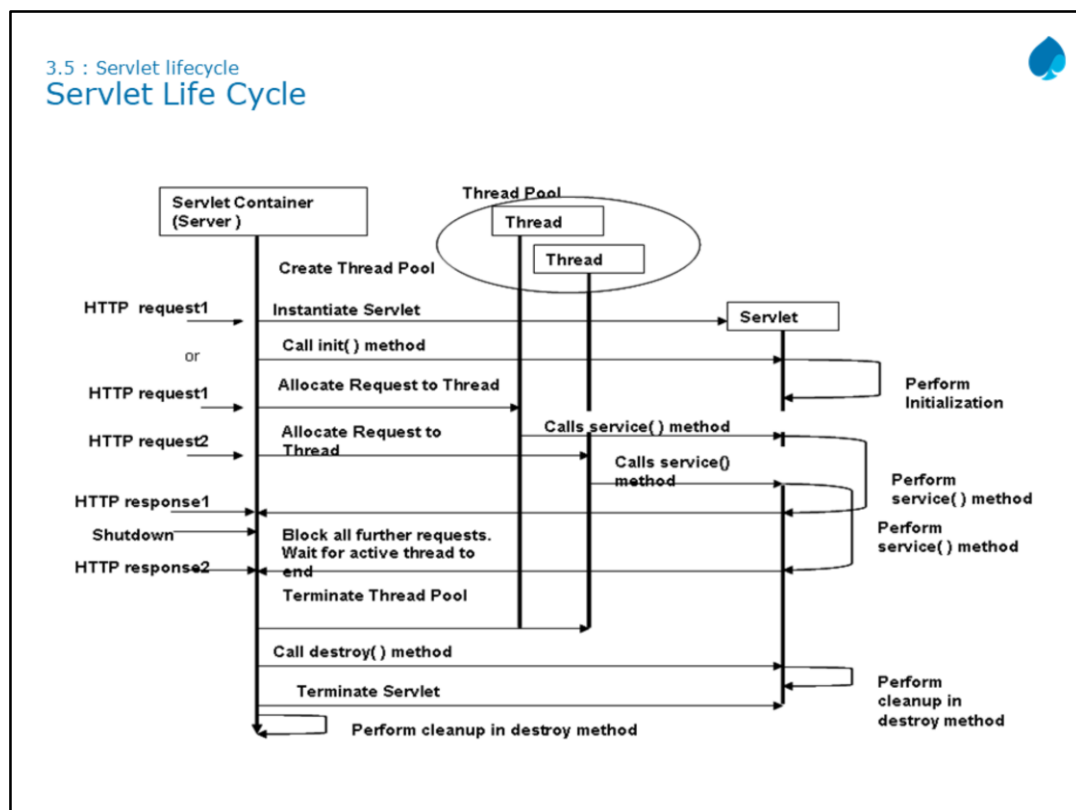
`init()` and `destroy()` can be overridden to manage resources that are held for the life of the servlet. Servlet can implement `getServletInfo()` to provide information about itself. The `service()` method of `HttpServlet` dispatches a request to different Java methods for different HTTP request methods. It recognizes the standard HTTP/1.1 methods like GET, HEAD, PUT, POST, DELETE, OPTIONS and TRACE. Other methods are answered with a Bad Request HTTP error.

Do not override the `service()` method because it handles setup and dispatching to all the `doXXX()` methods. Since `service()` method dispatches to `doXXX()` methods, it passes its request and response objects to these methods.

An HTTP servlet generally overrides `doGet()` to handle GET requests and `doPost()` to handle POST type of requests.

Instructor Notes:

Explain the sequence diagram about the life cycle of Servlets

**Basic Servlet Architecture:****Servlet Life Cycle:**

The process by which a server invokes a servlet can be broken down into the nine steps described below and as shown in the figure in the above slide:

The server loads the servlet when the client first requests it or, if configured to do so, at server start-up. The servlet may be loaded from either a local or remote location using the standard Java class loading facility. This step is equivalent to the following code:

```
Class c= Class.forName("com.igate.MyServlet");
```

The server creates one or more instances of the servlet class. Depending on the implementation, the server may create a single instance that services all requests through multiple threads or create a pool of instances from which one is chosen to service each new request. This step is equivalent to the following Java code:

```
Servlet s= (Servlet) c.newInstance();
```

The server constructs a `ServletConfig` object that provides initialization information to the servlet.

The server calls the servlet's `init(ServletConfig cfg)` method. The `init()` method is guaranteed to finish execution prior to the servlet processing the first request. If the server created multiple servlets instances (step 2), then the `init` method is called one time for each instance.

Instructor Notes:

3.6: Ease of developing Servlets through Annotations Annotations and their Need



Annotations can be described as metadata. These are metadata for the code written; and do not contain any business logic

They specify a standard way of defining metadata in code
Annotations are tightly coupled with the code

Application development becomes easy due to annotations

Why Were Annotations Introduced?

Prior to annotation (and even after) XML were extensively used for metadata and somehow a particular set of Application Developers and Architects thought XML maintenance was getting troublesome. They wanted something which could be coupled closely with code instead of XML which is very loosely coupled (in some cases almost separate) from code. Interesting point is XML configurations were introduced to separate configuration from code.

Suppose, you want to set some application wide constants/parameters. In this scenario, XML would be a better choice because this is not related with any specific piece of code. If you want to expose some method as a service, annotation would be a better choice as it needs to be tightly coupled with that method and developer of the method must be aware of this.


Another important factor is that annotation defines a standard way of defining metadata in code. Prior to annotations people also used their own ways to define metadata. Some examples are – using marker interfaces, comments, transient keywords etc. Each developer decided his own way to decide metadata, but annotation standardized things.

These days most frameworks use combination of both XML and Annotations to leverage positive aspects of both.

Instructor Notes:

3.6: Ease of developing Servlets through Annotations

Differences between Servlets 2.0 and Servlets 3.0



Servlet 2.0	Servlets 3.0
Creation of Servlets and other components via XML mappings	Creation of Servlets and other components via annotations
No such feature	Pluggability support for 3 rd party frameworks
No such feature	Asynchronous processing of Servlets

Pluggability Support for 3rd Party frameworks and Asynchronous processing of Servlets - details of which are shared in the Appendix.

Creation of Servlets and other components via XML mappings – details of which are shared in the Appendix.

In the subsequent slides we would be seeing creation of Servlets and other components via annotations.

Instructor Notes:

Mention about the @WebServlet annotation, and its various attributes. Also specify about importance of loadOnStartup element.

3.6: Ease of developing Servlets through Annotations

URL Mapping of Servlets and Annotations



@WebServlet Annotation:

javax.servlet.annotation.WebServlet is a class-level annotation that affirms the annotated class as a servlet and holds metadata about the declared servlet

The urlMappings attribute is a mandatory attribute of @WebServlet that specifies the URL pattern that invokes this servlet

Here is the much simplified version written to the Servlet 3.0 API.

As MyServlet is annotated as a servlet using the @WebServlet annotation, it gets initialized during the startup of the web container. Note that the deployment descriptor is optional in this case.

```
@WebServlet(name = "Basic ", urlPatterns={"/MyApp"}, loadOnStartup=1)
public class MyServlet extends HttpServlet{
    public void doGet(HttpServletRequest req, HttpServletResponse res){      ....  }}
```

When a request arrives, the container matches the URL in the request with the servlet's URL Patterns and if the URL pattern matches, the corresponding servlet will be invoked to serve the request. All other attributes of this annotation are optional, with reasonable defaults.

The **name attribute** specifies the name of servlet. It could be different from Servlet class, URL Pattern. Rather is just a dummy name given to servlets. This attribute is not mandatory.

It also has **loadOnStartup** attribute, this is used to specify - If you want to do something on the startup of the servlet (i.e. the element loadOnStartup with value greater or equal to zero, 0), you need put the code in a init method or in the constructor of the servlet: As we know the init() method is the first life cycle method of servlet.

loadOnStartup also specifies that it will be the first servlet to execute before the client request comes for the servlet and the init() method would run.

For comparison, a code snippet for writing a Java servlet using the old Servlet 2.5 API is shown below. In Servlet 2.5, the web container will initialize the servlet only if you configure the details of the servlet in the deployment descriptor

```
public class MyServlet extends HttpServlet {
    public void doGet (HttpServletRequest req, HttpServletResponse res)
    {
        ....
    }
}
```

Deployment descriptor (web.xml)

```
<web-app>
    <servlet>
        <servlet-name>MyServlet</servlet-name>
        <servlet-class>samples.MyServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>MyServlet</servlet-name>
        <url-pattern>/MyApp</url-pattern>
    </servlet-mapping>
</web-app>
```

Instructor Notes:

Explain what is deploying a servlet in a container

3.6: Ease of developing Servlets through Annotations
Writing First Servlet

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;
@WebServlet("/HelloWorld")
public class HelloWorld extends HttpServlet {
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("<BIG>Hello World</BIG>");
        out.println("</BODY></HTML>");
    }
}

```

Callouts:

- Servlets are not part of standard SDK, they are part of j2ee
- Use of @WebServlet annotation
- Servlets normally extend HttpServlet
- Setting the response type
- Getting PrintWriter object to send response to client
- Output to the client

Elements of a Web Application:

The above servlet prints a simple hello world.

This code snippet is using @WebServlet annotation.

Here the URL pattern mapping is embedded within the same annotation. The class also extends HttpServlet and overrides the doGet(request, response) method.

Make use of the annotation @WebServlet above the class name which will create the servlet – HelloWorld inside the container. The servlet will now be referred by the name "HelloWorld". The "/HelloWorld" is the URL Pattern of servlet, that would be used to invoke servlet from web browser.

URL pattern could be any logical name example: "/Hello". It need not be necessarily same name of Servlet. A servlet could have multiple URL patterns.

The doGet() method is overridden to service all incoming requests. Line 8 uses HttpServletResponse object to set the content type of the response. Line 9 uses the response object to retrieve a PrintWriter object to send responses. The remaining lines dynamically generate the HTTP response back to the browser client!

Following are minimum steps needed to create any servlet:

Write servlet

Import the necessary Java packages.

Inherit from GenericServlet or the HTTP convenience class HttpServlet.

Override the service method or the doXXX() methods.

Save the file with a .java filename extension.

2. Deploy the servlet in a container. The process for deploying (installing) a servlet into a webserver (container) varies from webserver to webserver.

Test the servlet by invoking the servlet from a JDK1.8-compatible browser. The URL typically will be of the form:

http://host_name:port/servletcontext/servlet/servlet_class_name

For example: http://localhost:9090/WildFlyServletsTestDemo/HelloWorld,

where WildFlyServletsTestDemo is the context root and HelloWorld is the name of the servlet.

If the server is installed on the same machine as the browser, then the server name can also be referred to as localhost.

Instructor Notes:

3.7: Retrieving information from HTML Page

Retrieving Information



Many a times, information about the environment in which a web application is running needs to be known.

Sometimes information about the server that is executing servlets or which client is sending the request needs to be known.

Sometimes information regarding the requests that the application is handling is required to be known

We shall now see a number of methods that provide this information to servlets

In the Servlet Hierarchy, there is a Listener associated with different objects like ServletContext, HttpSession etc; that would be invoked when these objects are created.

A Listener is a java class that would implement the ServletContextListener interface and would implement the lifecycle methods of ServletContext:

`contextInitialized(ServletContextEvent se)`

`contextDestroyed(ServletContextEvent se)`

We are using the annotation `@WebListener` above the Listener class name that would instruct the container to create the ServletContextListener and invoke it when a ServletContext object is created by Container. Data or configuration information that needs to be shared across multiple servlets could be set up in the ServletContext here. ServletContext is obtained via the `ServletContextEvent.getServletContext()` method.

Instructor Notes:

The last method (`getAttribute()`) may not be understood at this point. Inform participants that this method will be covered in detail later


3.7: Retrieving information from HTML Page

Getting Information about the Server

There are four methods that a servlet can use to learn about its server:

These are as follows:

- `public String ServletRequest.getServerName()`
- `public String ServletRequest.getServerPort()`
- `public String ServletContext.getServerInfo()`
- `public String ServletRequest.getAttribute(String name)`



```
req.getServerName(): localhost
req.getServerPort(): 9090
getServletContext().getServerInfo(): WildFly 8.1.0.Final - 1.0.15.Final
getServerInfo() name: WildFly 8.1.0.Final - 1.0.15.Final
getServletContext().getAttribute("attribute"): null
```

Retrieving Information:**Getting Information about the Server:**

A servlet can find out much about the server in which it is executing. It can learn the hostname, listening port, and server software, among other things. A servlet can display this information to a client, use it to customize its behavior based on a particular server package, or even use it to explicitly restrict the machine on which the servlet will run.

`getServerName()` method returns the host name of the server to which the request was sent.

`getServerPort()` method returns the port number to which the request was sent

`getServerInfo()` method returns the name and version of the server software as: JBossWeb/2.0.1.GA.

`getAttribute()` returns the value of the named server attribute as an Object or null if the attribute does not exist. The attributes are server dependent. We shall see this in detail in the session on inter-servlet communication.

Instructor Notes:

Refer to
demos for
these
methods

3.7: Retrieving information from HTML Page

Getting Info about client m/c and user



Getting Information about client machine:

- `public String ServletRequest.getRemoteAddr()`: It retrieves the IP address of the client machine.
- `public String ServletRequest.getRemoteHost()`: It retrieves the hostname of the client machine.

Getting Information about User:

- `public String HttpServletRequest.getRemoteUser()`: It returns the login of the user, if the user has been authenticated, or null if not.

Retrieving Information:

Getting Info About client m/c and user:

A servlet has the ability to find out about the client machine and for pages requiring authentication, about the actual user. This information can be used for logging access data, associating information with individual users, or restricting access to certain clients.

A servlet can use `getRemoteAddr()` and `getRemoteHost()` to retrieve the IP address and hostname of the client machine, respectively.

The information comes from the socket that connects the server to the client, so the remote address and hostname may be that of a proxy server. An example remote address might be "192.26.80.1320" while an example of remote host might be "dist.engr.com".

The method `getRemoteUser()` of the `HttpServletRequest` gives the username of the client. With the remote user's name, a servlet can save information about each client. Over the long term, it can remember each individual's preferences. For the short term, it can remember the series of pages viewed by the client and use them to add a sense of state to a stateless HTTP protocol. A simple servlet that uses `getRemoteUser()` can greet its clients by name and remember when each user last logged in.

Instructor Notes:

Lab on all the three forms of Scripting elements.

Lab: JSP Scripting Elements



Lab 1.1



Instructor Notes:

Summary



In this lesson, we have learnt:

- Role of Servlets in web application design
- Basic Servlet Architecture
- Servlet Lifecycle
- Elements of a Web Application
- Developing Servlets
- Initializing Servlets
- Getting Information about the Server, Client and User



Summary:

This lesson introduced us to servlets, web applications. We have seen how to create simple servlets, install on server and execute it. We learnt about the servlet lifecycle and the Servlet API. We also saw how to initialize servlets using `init()` method and how to find information about the server, the client making the request and the user.

Instructor Notes:

Answers for the
Review Questions:

Answer 1: service

Answer 2:
GenericServlet

Review Questions



Question 1: Which Method in a HttpServlet is not recommended to be overridden?

- Option 1: init
- Option 2: destroy
- Option 3: service
- Option 4: doGet
- Option 5: doPost



Question 2: What is the superclass of HttpServlet?

- Option 1: GenericServlet
- Option 2: Servlet
- Option 3: SuperHttpServlet

Instructor Notes:

Answers for the
Review
Questions:

Answer 3:
Server-side
components

Answer 4: init,
doGet and
service

Review Questions



Question 3: Servlets are:

- Option 1: Server-side components
- Option 2: Client-side components
- Option 3: Neither Client-side or Server-side components
- Option 4: Data tier components



Question 4: When a servlet receives an HTTP GET Request for the first time, which of these methods will be called? (Assume the servlet is not preloaded)

- Option 1: init
- Option 2: doPost
- Option 3: processRequest
- Option 4: doGet
- Option 5: service

Instructor Notes:

Review Questions



Question 5: What is the significance of using annotation `@WebServlet("/HelloWorld")` above public class HelloWorld extends HttpServlet `{}`

- Option 1: Instruct container to create Servlet Hello with URL Pattern `"/HelloWorld"`
- Option 2: Instruct container to create Servlet HelloWorld with URL Pattern `"/Hello"`
- Option 3: Instruct container to create Servlet HelloWorld with URL Pattern `"/HelloWorld"`
- Option 4: No significanc

