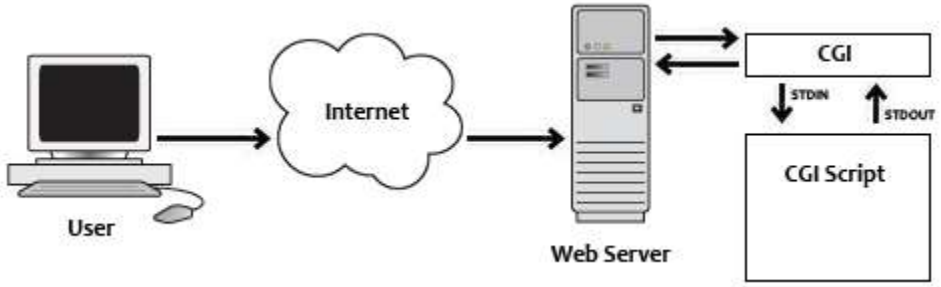
# Servlet

# CGI

- CGI (Common Gateway Interface) is used to provide dynamic content to the user.
- CGI is used to execute a program that resides in the server to process data or access databases to produce the relevant dynamic content.
- Programs that resides in server can be written in native operating system such as C++.

# Diagrammatic Representation



# Disadvantages Of CGI

- For each request CGI Server receives, It creates new Operating System Process.
- If the number of requests from the client increases then more time it will take to respond to the request.
- As programs executed by CGI Script are written in the native languages such as C, C++, perl which are platform independent.

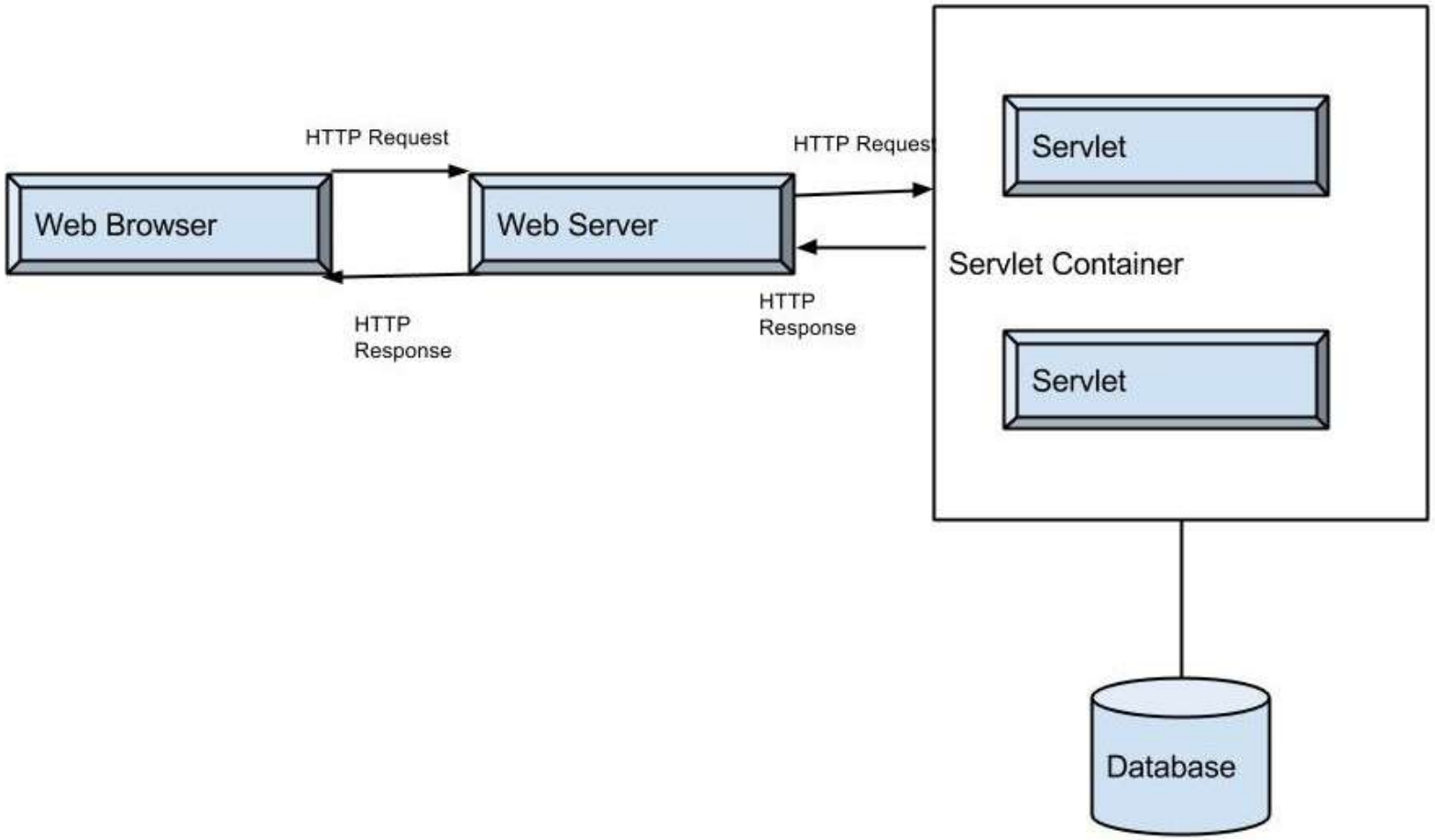
# Servlet API History

| SERVLET API VERSION | DATE | JAVA EE / JDK | FEATURES / CHANGES |
|---|---|---|---|
| Servlet 3.1 | May 2013 | JavaEE 7 | Non-blocking I/O, HTTP protocol upgrade mechanism |
| Servlet 3.0 | December 2009 | JavaEE 6, JavaSE 6 | Pluggability, Ease of development, Async Servlet, Security, File Uploading |
| Servlet 2.5 | September 2005 | JavaEE 5, JavaSE 5 | Requires JavaSE 5, supports annotation |
| Servlet 2.4 | November 2003 | J2EE 1.4, J2SE 1.3 | web.xml uses XML Schema |
| Servlet 2.3 | August 2001 | J2EE 1.3, J2SE 1.2 | Addition of `Filter` |
| Servlet 2.2 | August 1999 | J2EE 1.2, J2SE 1.2 | Becomes part of J2EE, introduced independent web applications in .war files |
| Servlet 2.1 | November 1998 | Unspecified | First official specification, added `RequestDispatcher`, `ServletContext` |
| Servlet 2.0 | | JDK 1.1 | Part of Java Servlet Development Kit 2.0 |
| Servlet 1.0 | June 1997 | | |

# What is Servlet ?

- Servlets are an important component of a J2EE application. Servlets along with JavaServer Pages (JSP) and EJB modules can be termed as server-side J2EE component types.
- Servlet is a Java Programming Language.
- Servlets are used to create web applications.
- Servlets are used to extend the applications hosted by web servers. Servlet runs in a J2EE application server.

# Servlet Architecture



# Process can be summarized

- A client sends a request to a Web Server, through a Web Browser.
- The Web Server searches for the required Servlet and initiates it.
- The Servlet then processes the client request and sends the response back to the server, which is then forwarded to the client.

# Advantages

- Servlets provide component based and platform-independent methods for building Web based applications.
- Each Request is run in a separate thread ,so servlet request processing is faster than CGI.
- Servlets overcomes the limitations of CGI program.
- Servlets run on Java Virtual Machine and in any platform and it is simple to write.
- Servlet are more powerful and the performance is better.

# Advantages(Contd...)

- Servlets are platform-independent.
- Servlet technology, in addition to improved performance, offers security, robustness, object orientation, and platform independence.
- As mentioned in the definition of servlets are fully integrated with the Java language and its standard APIs. Hence JDBC for Java database connectivity is also integrated in it.
- A servlet handles concurrent requests
- Handling HTTP requests and send text and data back to the client is made easy by servlet request and response objects.

# Disadvantages of a Servlet

- Servlets often contain both business logic and presentation logic so it makes application difficult to understand.
- You would need JRE to be installed to run a servlet program.

# Servlet vs CGI

| Servlet | CGI (Common Gateway Interface) |
|---|---|
| Servlets are portable | CGI is not portable. |
| In Servlets each request is handled by lightweight Java Thread | IN CGI each request is handled by heavy weight OS process |
| In Servlets, Data sharing is possible | In CGI, data sharing is not available. |
| Servlets can link directly to the Web server | CGI cannot directly link to Web server. |
| Session tracking and caching of previous computations can be performed | Session tracking and caching of previous computations cannot be performed |
| Automatic parsing and decoding of HTML form data can be performed. | Automatic parsing and decoding of HTML form data cannot be performed. |
| Servlets can read and Set HTTP Headers | CGI cannot read and Set HTTP Headers |
| Servlets can handle cookies | CGI cannot handle cookies |
| Servlets can track sessions | CGI cannot track sessions |
| Servlets is inexpensive than CGI | CGI is more expensive than Servlets |

# Servlet API

- **Servlet API** is supported by all Servlet containers, such as Tomcat and Weblogic, etc. The Application Programming Interface (API) contains interface and classes to write a servlet program. The servlet API contains two packages as listed below:
- **javax.servlet**
- **javax.servlet.http**

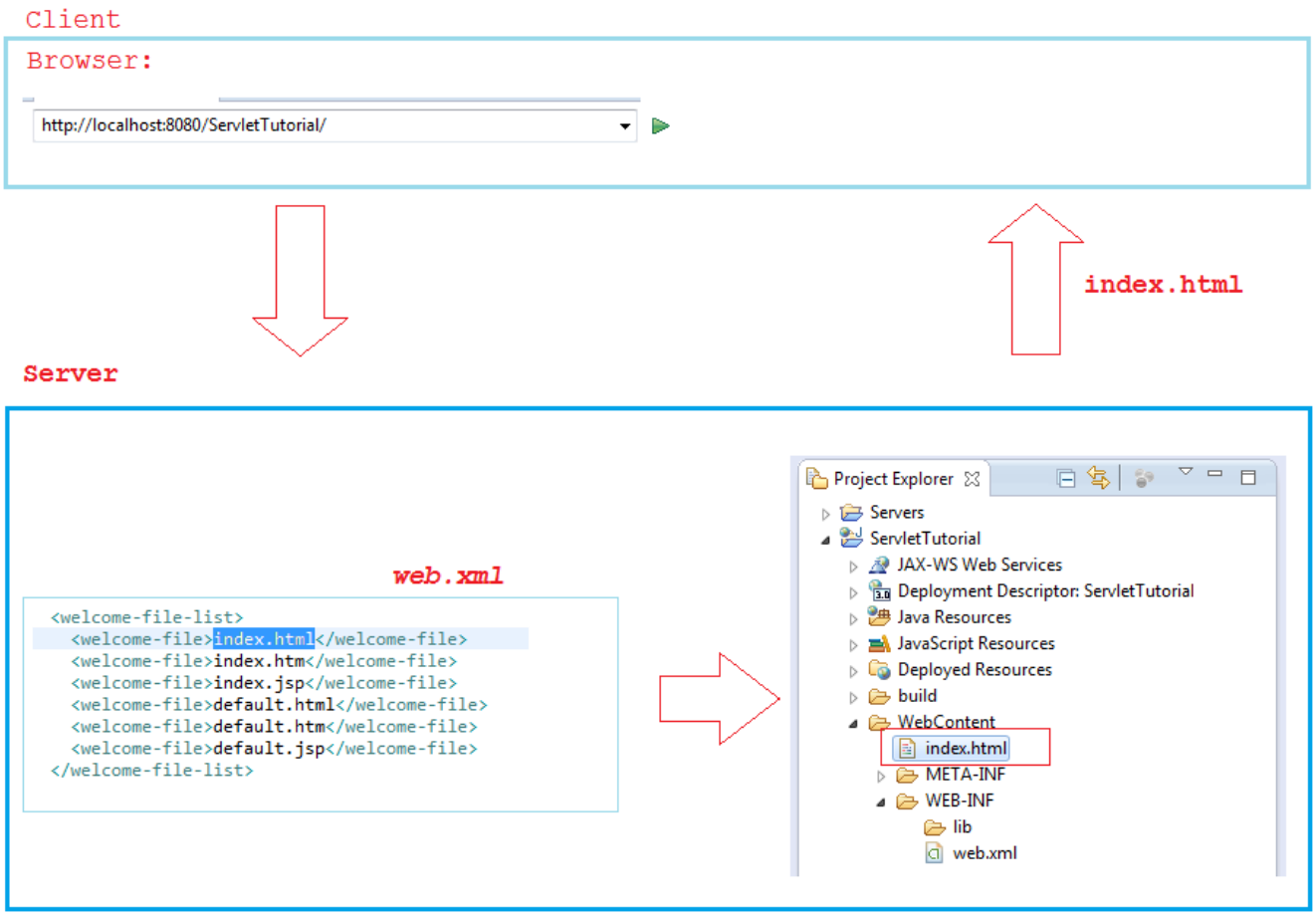
# Example(ServletDemo.java)

```java
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ServletDemo extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("  <head>");
        out.println("    <title>SimpleServlet</title>");
        out.println("  </head>");
        out.println("  <body>");
        out.println("    Hello, World");
        out.println("  </body>");
        out.println("</html>");
    }
}
```

# Example(Web.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
    <display-name>SampleServlet</display-name>
    <servlet>
        <servlet-name>ServletDemo</servlet-name>
        <servlet-class>ServletDemo</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ServletDemo</servlet-name>
        <url-pattern>/ServletDemo</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
</web-app>
```

# Example Step- by-Step



# Generic Servlet and Http Servlet

# Generic Servlet

- **GenericServlet** is an abstract class defined in the Servlet API. For implementing class we need to implement all the methods in *javax.servlet.Servlet* Interface. Most of the Servlet objects do not need initializing and destroying operations but still we need to implement all the methods in *javax.servlet.Servlet* interface. To solve this problem, Servlet API provides *javax.servlet.GenericServlet* which implements **ServletConfig**interface and provides the implementation for the methods in this interface except the service method.

# Methods of GenericServlet class

- **public void init ():** This method is invoked by init (ServletConfig) method of GenericServlet. It is used to provide custom initializations without disturbing the initializations performed by GenericServlet in the init (ServletConfig) method.
- **public void init (ServletConfig) :** It is used to initialize the servlet. It indicates Servlet instance in being placed into the service.
- **public abstract void service (ServletRequest request, ServletResponse response)** : It is used to process user request. It is called by servlet container to intimate servlet about client request. It carries out single request from the client. ServletRequest object is used to collect data which is available with client requested data. ServletResponse object is used to generate the output content.
- **public void destroy ():** It indicates servlet instance is being taken out of service. It is used to clean up any resources that servlet might have initialized.
- **public ServletConfig getServletConfig ():** It returns ServletConfig interface reference. It is used to get configuration information from web.xml file.

# Methods of GenericServlet class

- **public ServletContext getServletContext ():** It returns ServletContext object reference. It is used to get configuration information from web.xml file. It is also used to set, get or remove attribute from web.xml file. If information is changed then there is no need to modify the servlet. So it is easy to maintain.
- **public String getInitParameter (String name):** It returns the Servlet initialization parameter of the given name and if requested parameter is not available then it returns null.
- **public Enumeration String getInitParameternames ():** It returns names of all Servlet initialization parameters defined in web.xml file.
- **public String getServletInfo ():** It returns information about the respective servlet. For e.g. version, copyright.
- **public String getServletName ():** It returns the Servlet instance name defined in Deployment Descriptor (web.xml).
- **public void log (String msg):** It writes class name of servlet and specified message to a servlet log file.
- **public void log (String msg, Throwable t):** It writes explanatory message and stack trace for a given Throwable exception to the servlet log file.

# Methods of HttpServlet class

- **public void service (ServletRequest request, ServletResponse response):** This method is used to process the user request. For each request the web container will issue unique request and response to the service method.
- **public void service (HttpServletRequest request, HttpServletResponse response):**This method receives requests from the service () method. It dispatches requests depending on the request type.
- **protected void doGet (HttpServletRequest request, HttpServletResponse response):** By using doGet () method we can send specific amount of data. If we use doGet () method data is shown in address bar. We must override doGet () method depending on type of request.
- **protected void doPost (HttpServletRequest request, HttpServletResponse response):** We can send large amount of data by using doPost () method. By using this method data is not viewable in address bar. When we want to send secure data like passwords and other things doPost () method is used. We must override doPost () method depending on type of request.

# Methods of HttpServlet class

- **protected void doDelete (HttpServletRequest request, HttpServletResponse response):** It is used to delete files, web pages or documents from the server. If requests are formatted incorrectly then it will return HTTP “Bad Request” error.
- **protected void doPut (HttpServletRequest request, HttpServletResponse response):** This method is used to put files, web pages or documents in the server means for uploading files on the server. If requests are formatted incorrectly then it will return HTTP “Bad Request” error.
- **protected void doTrace (HttpServletRequest request, HttpServletResponse response):** This method is used for logging and debugging purpose. It can be used for testing the requested message. There is no need to override this method.

# Methods of HttpServlet class

- **protected void doOptions (HttpServletRequest request, HttpServletResponse response):** This method handles OPTIONS request. There is no need to override this method. It determines which HTTP method supported by server and returns correct header.
- **protected long getLastModified (HttpServletRequest request, HttpServletResponse response):** It returns the time when request was last modified. This method should override GET request to return modification time of object.
- **protected void doHead (HttpServletRequest request, HttpServletResponse response):** This method request header part of the GET request without the GET response body. It receives request from service method and handles the request. If HEAD requests are formatted incorrectly then it will return HTTP “Bad Request” error.

# Example for Servlet

# Web.xml

- **Web.xml** defines mapping between URL paths and servlets that handle requests with those paths.
- The **web.xml** file provides configuration and deployment deployment information for the Web components that comprise a Web application.
- The ***web.xml***descriptor files represents the core of the java application. The web.xml file is located in **WEB-INF** directory of web application.

# Example

```
<web-app>
<servlet>
<servlet-name>HelloWorld</servlet-name>
<servlet-class>servletexample.createHelloWorlExample</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HelloWorld</servlet-name>
<url-pattern>/HelloworldExample/*</url-pattern>
</servlet-mapping>
</web-app>
```

# Elements in Web.xml

- **<web-aap>:** This element represents whole application of web.xml file
- **<servlet>**: This is the sub element of and represents the servlet
- **<servlet-name>**: This is the sub element of servlet and used to represents the name of servlet
- **<servlet-class>**: This is the sub element of servlet and used to represents the class of servlet
- **<servlet-mapping>**: This is the sub element of and used to map the servlet
- **<url-pattern>**: This is the sub element of servlet-mapping and used to client side to invoke the servlet

# Web.xml Tags

- Welcome-file-list tag; This tag is used to specify the default page of web application if none is specified.
- <welcome-file-list>
- <welcome-file>index.jsp</welcome-file>
- </welcome-file-list>
- In the above example index.jsp used as web page for web application

# Web.xml Tags

- Error page tag: This tag is used to specify the error occurred in the while weblogic server is responding ti a HTTP request, returns an HTML page that displays either the HTTP error code.
- <error-page>
- <error-code>105</error-code>
- <location>/jsp/error/PageNotFound.jsp</location>
- </error page>

# Web.xml Tags

- Session config tag: This tag is used to specify the session configuration parameter
- Example:
- <session-config>
- <session-timeout>
- 15
- </session-timeout>
- </session-config>
- In the above example session-config tag contains another tag session-timeout which specifies the http session timeout. The time specify in minutes.

# Advantages of web.xml file

- The first benefit of the xml is we can write it in our won markup language. There is a no restricted to limited sets of tags. By defining our own tag we can create a markup language in terms of specific problem.
- Searching the data is easy and efficient.
- Completely compatible with Java™ and 100% portable.

# Simple Hello World