**Servlets**

A server side component to handle HTTP request

To perform the request and response processing

Servlets are java classes that are loaded and executed by a servlet container that can run standalone or as a component of a web server or a J2EE server

By default the container does not initialize the servlets as soon as it starts up.

It initializes a servlet when it receives a request for the first time for that servlet.

This is called **lazy loading**.

The servlet deployment descriptor (web.xml) defines the <load-on-startup> element, which can be configured to make the servlet container load and initialize the servlet as soon as it starts up.

The process of loading a servlet before any request comes in is called pre-loading or pre-initializing a servlet.

Servlet Life Cycle

* Loading and instantiating
* Initialization
* Request Handling
* End of life

The servlet container reads each web.xml file, and **loads** the servlet classes identified in the deployment descriptor.

Then it **instantiates** each servlet by calling its no-argument constructor

After the servlet is loaded and instantiated, the servlet must be initialized.

This occurs when the container calls the init(ServletConfig) method.

The init() method

* Read initialization parameters or configuration data.
* Initialize external resources such as database connections.
* Perform other onetime activities.

The deployment descriptor can define parameters that apply to the servlet through the <init-param> element.

The servlet container reads these parameters from the web.xml file and stores them as name-value pairs in a ServletConfig object.

How to Make Your Servlets Thread-Safe

Use method variables for request-specific data. Whenever you need to access data from a request, that data should be stored in method variables, also known as local variables.

These are variables defined within the scope of a method. Each thread that enters a method gets its own copy of the method variables. Thus, no thread can change the member variables of any other thread.

How Not to Make Your Servlets Thread-Safe

1. Use SingleThreadModel interface

What SingleThreadModel interface does is signal to the servlet container that only a single thread should be allowed in the class at a time.

Using SingleThreadModel does not make your servlet threadsafe.

You would still need to synchronize access to these resources.

2. Synchronize service(), doPost(), or doGet()

**Session Management**

HTTP is a stateless protocol.

Each request and response stand alone.

Without session management, each time a client makes a request to a server, it’s a brand new user with a brand new request from the server’s point of view.

A **session** refers to the entire interaction between a client and a server from the time of the client’s first request, which generally begins the session, to the time the session is terminated.

The session could be terminated by the client’s request, or the server could automatically close it after a certain period of time.

A session can last for just a few minutes, or it could last days or weeks or months (if the application were willing to let a session last that long).

Session Tracking with Cookies

Session Tracking with URL Rewriting

**Filter**

A filter dynamically intercepts requests and responses to transform or use the information contained in the requests or responses but typically do not themselves create responses.

Filters can also be used to transform the response from the Servlet or JSP before sending it back to client.

public void init(FilterConfig filterConfig)

public void doFilter (ServletRequest, ServletResponse, FilterChain)

public void destroy()

Types

Authentication Filter

Logging and Auditing Filters