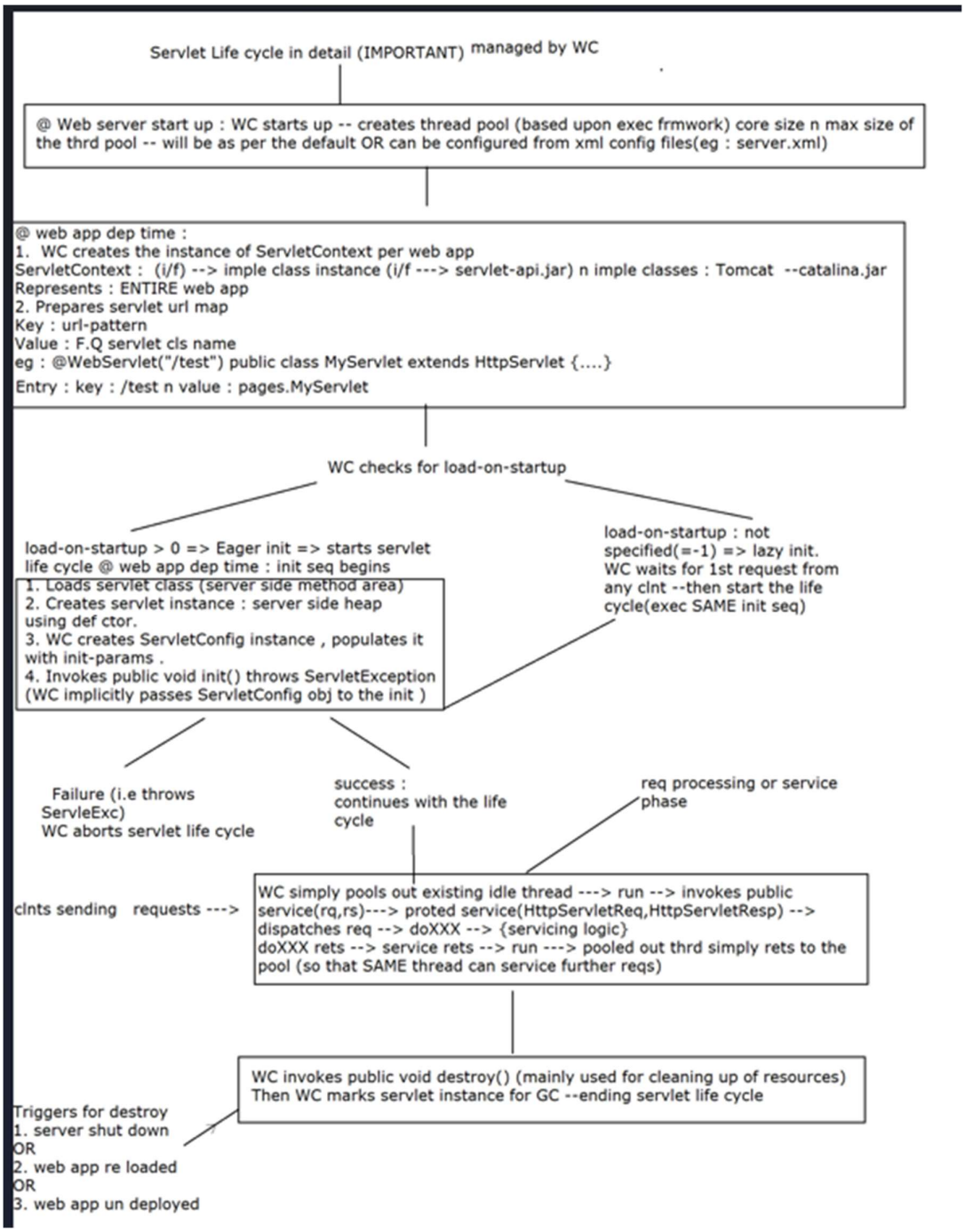
# Servlet Life Cycle - Study Notes




A **Servlet** is a Java class that handles HTTP requests in a web application. Its life cycle consists of various stages, including loading, instantiation, initialization, request handling, and destruction. The **Web Container** (part of the Web Server) is responsible for managing the entire life cycle of a servlet.

The diagram outlines the process using a flow of arrows and components. Let's go step-by-step.

---

### 🔷 STEP 1: Web Server Starts Up

#### 🔸 What Happens?

- The **web server** (e.g., Apache Tomcat) **starts up**.

# Servlet Life Cycle - Study Notes

- It triggers the **Web Container (WC)**, which is a part of the server responsible for managing servlet execution.
- The **WC creates a thread pool**, which it uses to serve client requests.

🔸 **Thread Pool Details:**

- Based on the **execution framework**, a pool of threads is created.
- **Core size and max size** of the thread pool:
  - Can be set as **default**, or
  - Configured using XML configuration files like server.xml in Tomcat.

🔸 **Purpose:**

- These threads are reused for incoming requests instead of creating a new thread each time (improves performance and scalability).

---

🔷 **STEP 2: Web Application is Deployed**

🔸 **What Happens?**

- WC loads the **Web Application (WAR)** file.
- For each application, WC creates a single instance of **ServletContext**, which acts as a shared memory and configuration holder.

🔸 **Key Components:**

- **Web.xml** (Deployment Descriptor):
  - Contains servlet declarations and mappings.
  - Example mapping:

xml

Copy code

```
<servlet>
 <servlet-name>MyServlet</servlet-name>
 <servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>MyServlet</servlet-name>
 <url-pattern>/test</url-pattern>
</servlet-mapping>
```

- OR, annotations like:

java

Copy code

@WebServlet("/test")

- **Servlet class**:

java

Copy code

```
public class MyServlet extends HttpServlet {
  // service methods (doGet, doPost etc.)
}
```

🔸 **Servlet Mapping:**

- Mapping connects a URL (like /test) to a specific servlet class (MyServlet).

---

🔷 **STEP 3: WC Checks for load-on-startup**

This step determines **when the servlet is initialized**. Two cases are possible:

---

✅ **Case 1: load-on-startup = 0 OR Not Specified**

This is **lazy initialization**.

🔸 **What Happens?**

- Servlet is **not loaded** during deployment.
- Servlet is loaded and initialized **only when the first request** is made to the mapped URL.

🔸 **Process:**

1. WC loads the servlet class (Class.forName()).
2. WC creates an instance using the **no-arg constructor**.
3. WC creates a **ServletConfig** object.
4. WC **links ServletConfig with ServletContext**.
5. WC calls the **init() method** on the servlet instance.
    - Initializes resources.
    - Can throw ServletException if something goes wrong.
6. Servlet is now **ready to handle requests** using service().

# Servlet Life Cycle - Study Notes

🔸 **If Exception Occurs:**

- If init() throws ServletException, the servlet **fails to initialize**.
- WC **aborts the life cycle** of that servlet.

---

✅ **Case 2: load-on-startup = n (where n > 0)**

This is **eager initialization**.

🔸 **What Happens?**

- Servlet is **immediately loaded and initialized** during web app deployment.
- Even before any client request is made.

🔸 **Same Life Cycle:**

- Load class
- Instantiate it
- Create ServletConfig
- Call init()
- Ready to handle requests

---

🔷 **STEP 4: Request Handling using service() Method**

Once the servlet is initialized, it can process requests.

🔸 **When a Request Comes In:**

- WC **picks a thread** from the **thread pool**.
- Invokes the **service() method** of the servlet.
    - Internally calls doGet(), doPost(), etc., based on HTTP request type.
- All requests are handled using **the same servlet instance**, but **different threads**.

🔸 **Threading Behavior:**

- Multiple client requests are handled concurrently.
- **Same servlet instance**, **multiple threads**.
- That's why servlets must be **thread-safe**—avoid shared mutable state without synchronization.

---

🔷 **STEP 5: Servlet Destruction (destroy())**

This is the final stage in the servlet's life cycle.

# Servlet Life Cycle - Study Notes

🔸 **When is destroy() called?**

- When:
  - Web application is **undeployed**, OR
  - **Server shuts down**, OR
  - **Reloading** the application.

🔸 **What Happens?**

- WC **calls destroy()** on the servlet instance.
- Used to **release resources** like:
  - Database connections,
  - File handles,
  - Thread pools,
  - Any background tasks.

🔸 **After that:**

- Servlet is **marked for Garbage Collection (GC)**.
- The life cycle of the servlet **ends**.

---

🔷 **Quick Recap: Life Cycle Methods in HttpServlet Class**

| Method | Purpose | Called By |
|---|---|---|
| init() | Initializes the servlet (called once) | WC |
| service() | Handles every request | WC |
| destroy() | Cleans up resources (called once) | WC |

---

🔷 **ServletContext vs ServletConfig**

| Feature | ServletContext | ServletConfig |
|---|---|---|
| Scope | Application-wide | Per servlet |
| Created By | Web Container | Web Container |
| Purpose | Shared config and attributes | Servlet-specific init params |
| Lifespan | From deployment to undeployment | From servlet creation to destroy |

---

# Servlet Life Cycle - Study Notes

🔷 **Summary: Lifecycle Events in Order**

1. **Web server starts** → WC initialized.
2. **Thread pool created**.
3. **Web app is deployed** → ServletContext created.
4. **Servlet class mapped via web.xml or annotation**.
5. Servlet is **initialized** (init()):
    - **Eagerly** if load-on-startup > 0
    - **Lazily** if load-on-startup = 0 or absent
6. WC handles **requests** via service() → doGet() / doPost().
7. On shutdown, WC calls **destroy()**.
8. Servlet is marked for **garbage collection**.

---

📝 **Final Note:**

The diagram and this explanation together form a **complete picture** of how servlets operate within a Java EE or Jakarta EE web application. Understanding this life cycle is crucial for:

- Writing efficient, scalable servlets.
- Managing resources properly.
- Debugging deployment and request issues.

## 1. What is a Servlet?

A servlet is a Java program that runs on a web server and handles client requests by generating dynamic web content. It extends the capabilities of servers.

## 2. Servlet Life Cycle Stages

The life cycle of a servlet is managed by the Web Container and consists of:

- Loading and Instantiation
- Initialization using init()
- Request Handling using service()
- Destruction using destroy()

# Servlet Life Cycle - Study Notes

## 3. init() Method

Called once when the servlet is first loaded. Used to initialize resources like DB connections. Executed before any request is processed.

## 4. service() Method

Called for each request. It dispatches to doGet(), doPost(), etc. depending on the HTTP method. Runs on separate threads, hence servlets must be thread-safe.

## 5. destroy() Method

Called when the servlet is taken out of service (server shutdown or undeployment). Used to release any resources held.

## 6. Lazy vs Eager Initialization

- Lazy (default): Servlet initialized upon first request.
- Eager: Specified using load-on-startup in web.xml or @WebServlet. Servlet loaded on deployment.

## 7. ServletConfig vs ServletContext

ServletConfig: Specific to a servlet, contains init parameters.
ServletContext: Shared across servlets, contains application-wide settings and resources.

## 8. Thread Pool in Servlet Container

Web Container creates a thread pool on startup. Threads from this pool handle incoming servlet requests to

improve performance.

## 9. Example URL Mapping

@WebServlet("/example") maps a servlet to handle requests on '/example' path. Alternatively defined in web.xml.

## 10. Summary

1. Web server starts -> WC initialized
2. Thread pool created
3. Servlet loaded -> init()
4. Requests handled by service() -> doGet()/doPost()
5. Servlet destroyed via destroy() when app stops