# Servlet Filter Lifecycle

The Servlet Filter lifecycle consists of three main phases:

1. **Initialization:** The web container initializes the filter when the application starts up or the first request is made. The init(FilterConfig filterConfig) method is called once to perform any setup tasks. The FilterConfig object provides access to the filter's configuration parameters defined in the deployment descriptor (web.xml) or annotations.
2. **Request Processing:** For each request, the container invokes the doFilter(ServletRequest request, ServletResponse response, FilterChain chain) method. This method performs the core filtering logic. It can:
   * Modify the request or response objects.
   * Log information about the request.
   * Validate user input.
   * Check for user authentication or authorization.
   * Stop the request from proceeding further down the filter chain.
   * Pass the request to the next filter in the chain by calling chain.doFilter(request, response). If this method is not called, the request will not reach the target servlet.
3. **Destruction:** When the web application is shut down or the filter is no longer needed, the web container calls the destroy() method. This method is called once to perform any cleanup tasks, such as releasing resources or closing database connections.

**Detailed Execution Flow of the doFilter() Method**

The doFilter() method's execution can be thought of as having two parts, creating a "before and after" effect around the target servlet's execution.

1. **Pre-processing (Before the chain.doFilter() call):**
   * When a request arrives, the container invokes the doFilter() method of the *first* filter in the chain.
   * At this point, you can perform all your pre-processing logic. This is where you would:
     + **Log information**: Record the request URL, user agent, or timestamps.
     + **Authentication/Authorization**: Check if the user is logged in or has the necessary permissions. If not, you can redirect them to a login page or send an error response.
     + **Data Modification**: Read the request and modify it. For example, you can decode a request body, or add or remove request parameters. This is often done by wrapping the ServletRequest object.
     + **Compression/Encryption**: Intercept the input stream to decompress or decrypt the data before it reaches the servlet.
     + **Sanitization**: Clean user input to prevent XSS (Cross-Site Scripting) attacks.
   * After completing the pre-processing tasks, the filter **must** call chain.doFilter(request, response) to pass the request to the next filter in the chain, or to the target servlet if it's the last filter.
   * **Critical Point**: If chain.doFilter() is **not** called, the request processing stops, and the request will **never** reach the next filter or the target servlet. This is how a filter can block a request (e.g., if a user is not authenticated).
2. **Post-processing (After the chain.doFilter() call):**
   * Once chain.doFilter() is called, the request moves down the chain. The target servlet is executed, and its response is generated.
   * The control then flows back up the filter chain in reverse order.
   * Code placed **after** the chain.doFilter() call in a filter's doFilter() method will be executed as the response travels back to the client.
   * This is where you can perform post-processing tasks such as:
     + **Logging**: Log the response status code and execution time.
     + **Response Modification**: Add or modify response headers (e.g., adding Cache-Control headers).
     + **Content Manipulation**: Compress the response data, encrypt it, or modify the content (e.g., adding a footer to an HTML page).
     + **Resource Cleanup**: Perform any cleanup related to the request (e.g., closing a session-specific database connection).