Servlet Configuration and Deployment

Java Servlets are server-side components that provide a powerful way to extend the functionality of a web server. They are used to handle HTTP requests and generate dynamic web content. Servlet configuration and deployment are essential aspects of using Java Servlets. I'll explain these concepts and provide two code examples to illustrate them.

Servlet Configuration:

Servlet configuration typically involves setting up the following:

1. **Servlet Mapping**: You need to map a URL pattern to a specific servlet in the `web.xml` (web deployment descriptor) or through annotations in modern servlet containers. This tells the web server which servlet should handle requests for a particular URL.

2. **Servlet Initialization Parameters**: You can configure initialization parameters for a servlet. These parameters are set in the `web.xml` or through annotations, and they allow you to customize the behavior of your servlet.

3. **Servlet Filters**: Servlet filters are components that can be configured to intercept and process requests and responses before they reach a servlet. You can configure filters in the `web.xml` to perform tasks like authentication, logging, and more.

Servlet Deployment:

Servlet deployment involves deploying your servlet in a web container or application server. Here are the steps for deploying a servlet:

1. **Compile Your Servlet**: You write your servlet code and compile it using a Java compiler. The resulting `.class` files should be placed in the correct directory structure.

2. **Create a Web Application**: You create a directory structure for your web application, including the `WEB-INF` directory. Inside `WEB-INF`, you place the `web.xml` file.

3. **Configure Servlet**: You configure your servlet in the `web.xml` file by specifying the servlet class, mapping it to a URL pattern, and setting any initialization parameters.

4. **Deploy to a Servlet Container**: You deploy your web application (WAR file) to a servlet container or application server. The container is responsible for managing the servlet's lifecycle and handling requests.

Now, let's provide two code examples: one for servlet configuration in `web.xml` and another using servlet annotations for configuration.

Example 1: Servlet Configuration in `web.xml`

```xml
<!-- web.xml -->
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>com.example.HelloServlet</servlet-class>
    <init-param>
        <param-name>greeting</param-name>
        <param-value>Hello, World!</param-value>
    </init-param>
</servlet>

<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

```java
// HelloServlet.java
package com.example;

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

public class HelloServlet extends HttpServlet {
    private String greeting;

    public void init() {
        greeting = getInitParameter("greeting");
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.getWriter().write(greeting);
    }
}
```

In this example, we configure the `HelloServlet` using `web.xml`. The servlet is mapped to the URL pattern `/hello`, and it takes an initialization parameter `greeting`.

Example 2: Servlet Configuration with Annotations

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

@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {
    private String greeting = "Hello, World!";

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.getWriter().write(greeting);
    }
}
```

In this example, we configure the `HelloServlet` using annotations. We specify the servlet's name and URL pattern directly in the code.

Both examples achieve the same goal, but the second example uses modern servlet annotation-based configuration, which is more concise and often preferred in contemporary Java web applications.