Jetty

Jetty is a highly scalable and versatile web server and servlet container that can be used to serve Java web applications. It is often used for testing and development purposes, but it can also be deployed in production environments.

Here are some common use cases and features of Jetty:

1. **Web Server**: Jetty can serve static content and dynamic web applications using servlets and JSPs.

2. **Servlet Container**: It implements the Java Servlet API, allowing you to deploy Java web applications that conform to the servlet specification.

3. **Embedded Server**: Jetty can be embedded within Java applications, allowing you to run web applications as part of your application stack without the need for a separate web server.

4. **HTTP/2 and WebSocket Support**: Jetty supports modern protocols like HTTP/2 and WebSocket, enabling real-time communication between clients and servers.

5. **Asynchronous I/O**: Jetty's architecture is designed for high concurrency and performance, making it suitable for handling a large number of simultaneous connections.

6. **Modular and Lightweight**: Jetty's architecture is modular and lightweight, allowing you to include only the components you need for your application.

7. **Extensible**: Jetty provides a rich set of APIs and extension points, allowing you to customize and extend its functionality as needed.

Overall, Jetty is a powerful and flexible web server and servlet container that is widely used in the Java ecosystem for building and deploying web applications.

Sure, here's a basic example of how you can use Jetty as an embedded server in a Java application:

```java
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.servlet.ServletContextHandler;
import org.eclipse.jetty.servlet.ServletHolder;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
```

```
public class EmbeddedJettyExample {

    public static void main(String[] args) throws Exception {
        // Create a new Jetty server instance
        Server server = new Server(8080);

        // Create a servlet context handler
        ServletContextHandler contextHandler = new
ServletContextHandler(ServletContextHandler.SESSIONS);
        contextHandler.setContextPath("/");

        // Add a servlet to the context handler
        contextHandler.addServlet(new ServletHolder(new HelloWorldServlet()), "/
hello");

        // Attach the context handler to the server
        server.setHandler(contextHandler);

        // Start the server
        server.start();
        System.out.println("Server started on port 8080");

        // Wait for the server to finish
        server.join();
    }

    // Define a simple servlet
    public static class HelloWorldServlet extends HttpServlet {
        @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws IOException {
            resp.setContentType("text/plain");
            resp.getWriter().println("Hello, Jetty!");
        }
    }
}
```

In this example:

1. We create a new Jetty `Server` instance and specify the port number (8080) on which the server will listen for incoming requests.

2. We create a `ServletContextHandler` to manage servlets and other servlet-related resources.

3. We define a simple servlet (`HelloWorldServlet`) that responds with a plain text "Hello, Jetty!" message when accessed.

4. We add the servlet to the `ServletContextHandler` and specify the URL mapping ("/hello") for accessing the servlet.

5. We attach the `ServletContextHandler` to the Jetty server.

6. We start the server using `server.start()` and wait for it to finish using `server.join()`.

When you run this application, you can access the servlet at `http://localhost:8080/hello`, and it will respond with the "Hello, Jetty!" message.

To use the embedded Jetty server instead of Tomcat, you need to exclude the spring-boot-starter-tomcat from spring-boot-starter-web and include the spring-boot-starter-jetty dependency. Also, you will need to exclude the default added spring-boot-starter-tomcat dependency.

https://www.jrebel.com/blog/jetty-server