Utiliser Jetty pour intégrer un serveur web

Pour utiliser `org.eclipse.jetty` dans un projet Java, vous souhaitez généralement tirer parti des bibliothèques Jetty pour intégrer un serveur web léger ou un conteneur de servlets dans votre application. Jetty, maintenu sous l'égide de la Fondation Eclipse, est un projet open-source qui vous permet de servir du contenu web, de gérer les requêtes HTTP et de prendre en charge des fonctionnalités telles que les servlets et les applications web. Voici un guide étape par étape pour vous aider à démarrer avec l'utilisation de `org.eclipse.jetty` pour configurer un serveur web de base dans une application Java en utilisant Maven comme outil de construction.

Étape 1 : Ajouter les dépendances Jetty à votre projet

Pour utiliser Jetty, vous devez inclure ses bibliothèques dans votre projet. Si vous utilisez Maven, ajoutez les dépendances nécessaires à votre fichier `pom.xml`. Pour une configuration de serveur web de base avec prise en charge des servlets, incluez `jetty-server` et `jetty-servlet`. Voici un exemple :

```
<project>
    <dependencies>
        <!-- Jetty Server -->
        <dependency>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-server</artifactId>
            <version>9.4.31.v20200723</version>
        </dependency>
        <!-- Jetty Servlet Support -->
        <dependency>
            <groupId>org.eclipse.jetty</groupId>
            <artifactId>jetty-servlet</artifactId>
            <version>9.4.31.v20200723</version>
        </dependency>
        <!-- Servlet API (optionnel, souvent inclus transitivement) -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.1.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
</project>
```

- `jetty-server` : Fournit la fonctionnalité de serveur de base pour gérer les requêtes HTTP.
- `jetty-servlet` : Ajoute la prise en charge des servlets, vous permettant d'utiliser l'API Java Servlet.
- `javax.servlet-api` : Définit les interfaces des servlets ; incluse explicitement ici pour plus de clarté, bien qu'elle soit souvent apportée transitivement par `jetty-servlet`.

Note : Vérifiez le Maven Central Repository pour la dernière version de Jetty et mettez à jour la balise `<version>` en conséquence.

Étape 2 : Créer un serveur Jetty simple

Vous pouvez utiliser Jetty de manière programmatique en créant une instance de la classe `Server` du package `org.eclipse.jetty.server`. Voici deux approches courantes : utiliser un gestionnaire de base pour des réponses statiques ou utiliser un servlet pour du contenu dynamique.

Option 1 : Utiliser un gestionnaire de base Cet exemple configure un serveur qui répond "Hello, World!"à toutes les requêtes.

```
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.server.Request;
import org.eclipse.jetty.server.handler.AbstractHandler;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class SimpleJettyServer {
    public static void main(String[] args) throws Exception {
        // Créer une instance de serveur écoutant sur le port 8080
        Server server = new Server(8080);

        // Définir un gestionnaire pour traiter les requêtes
        server.setHandler(new AbstractHandler() {
            @Override
            public void handle(String target, Request baseRequest, HttpServletRequest request, HttpServletResp
                    throws IOException {
                // Définir les propriétés de la réponse
                response.setContentType("text/plain");
                response.setStatus(HttpServletResponse.SC_OK);
                response.getWriter().println("Hello, World!");
```

```
                // Marquer la requête comme traitée
                baseRequest.setHandled(true);
            }
        });

        // Démarrer le serveur
        server.start();
        server.join(); // Garde le serveur en cours d'exécution jusqu'à ce qu'il soit arrêté
    }
}
```

- `Server server = new Server(8080)` : Crée un serveur Jetty sur le port 8080.
- `AbstractHandler` : Un gestionnaire simple où vous définissez comment traiter les requêtes dans la méthode `handle`.
- `baseRequest.setHandled(true)` : Indique que ce gestionnaire a traité la requête, empêchant tout traitement supplémentaire.

Exécutez ce code, puis ouvrez un navigateur et accédez à `http://localhost:8080/` pour voir “Hello, World!” .

Option 2 : Utiliser un servlet Pour des applications plus complexes, vous pouvez utiliser des servlets avec un `ServletContextHandler`.

```
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 ServletJettyServer {
    public static void main(String[] args) throws Exception {
        // Créer une instance de serveur sur le port 8080
        Server server = new Server(8080);

        // Créer un gestionnaire de contexte de servlet
        ServletContextHandler context = new ServletContextHandler();
        context.setContextPath("/"); // Chemin de contexte racine
```

```
        // Définir le gestionnaire pour le serveur
        server.setHandler(context);

        // Ajouter un servlet pour traiter les requêtes
        context.addServlet(new ServletHolder(new HelloServlet()), "/*");

        // Démarrer le serveur
        server.start();
        server.join();
    }
}

// Définir un servlet simple
class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        response.setContentType("text/plain");
        response.setStatus(HttpServletResponse.SC_OK);
        response.getWriter().println("Hello from Jetty Servlet!");
    }
}
```

- `ServletContextHandler` : Gère les contextes et les mappages de servlets.
- `context.addServlet` : Enregistre le `HelloServlet` pour traiter toutes les requêtes (`/*`).
- `doGet` : Définit le comportement du servlet pour les requêtes GET.

Exécutez ce code et visitez `http://localhost:8080/` pour voir “Hello from Jetty Servlet!”.

Étape 3 : Compiler et exécuter

1. Assurez-vous que votre projet est configuré avec Maven (par exemple, exécutez `mvn compile` pour télécharger les dépendances).
2. Compilez et exécutez la classe Java (par exemple, `java SimpleJettyServer` ou utilisez la fonction d’exécution de votre IDE).
3. Testez le serveur en accédant à `http://localhost:8080/` dans un navigateur ou en utilisant un outil comme `curl`.

Cas d'utilisation supplémentaires

- **Déployer un fichier WAR** : Utilisez `WebAppContext` pour déployer un fichier d'archive d'application web (WAR) ou un répertoire :

```
import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.webapp.WebAppContext;

public class WarJettyServer {
    public static void main(String[] args) throws Exception {
        Server server = new Server(8080);
        WebAppContext webapp = new WebAppContext();
        webapp.setContextPath("/");
        webapp.setWar("path/to/your.war"); // Chemin vers votre fichier WAR
        server.setHandler(webapp);
        server.start();
        server.join();
    }
}
```

  Ajoutez la dépendance `jetty-webapp` à votre `pom.xml` pour cette fonctionnalité.

- **Configurer Jetty** : Ajustez les paramètres comme le port, les pools de threads ou HTTPS de manière programmatique ou via des fichiers de configuration XML (voir la documentation Jetty pour les configurations avancées).

Points clés

- **Imports** : Utilisez les classes de `org.eclipse.jetty.server`, `org.eclipse.jetty.servlet`, etc., en les important selon les besoins.
- **Flexibilité** : Jetty peut servir du contenu statique, gérer des servlets, déployer des fichiers WAR ou même prendre en charge les WebSockets et HTTP/2 avec des modules supplémentaires.
- **Documentation** : Pour plus de détails, consultez la documentation officielle de Jetty.

En suivant ces étapes, vous pouvez utiliser efficacement `org.eclipse.jetty` pour intégrer un serveur web dans votre application Java, adapté à vos besoins spécifiques !