Jakarta EE 10 Tutorials

１．開発環境

１－１．ソフトウェア

ソフトウェア	バージョン	説明
Adoptium OpenJDK	17.0.11+9	
Eclipse IDE for Enterprise Java and Web Developers	2023-12	
WildFly	31.0.1.Final	WildFly Maven Plugin で使用
Apache Maven	3.9.6	

※ Eclipse を日本語化する場合

Pleiades 日本語化プラグインのサイトから Pleiades プラグイン単体をダウンロードします。

ダウンロードした zip を任意のディレクトリに展開し、zip 内の readme/readme_pleiades.txt の内容に従ってインストールした Eclipse を日本語化してください。

１－２．環境変数

環境変数	説明
JAVA_HOME	Adoptium OpenJDK 17.0.11+9 インストールディレクトリのパスを指定
MAVEN_HOME	Apache Maven 3.9.6 インストールディレクトリのパスを指定
Path	%Java_HOME%\bin、%MAVEN_HOME%\bin を追加

※ MAVEN_HOME は Maven 3.5 より不要となっていますが、説明を簡単にするために追加しています。

2．チュートリアル

2－1．RESTful Web Service

この項では Jakarta EE を使用して RESTful Web サービスを作成する方法を説明します。
作成するサービスは `http://127.0.0.1:8080/<プロジェクト名>/restfulservice/hello` で HTTP GET リクエストを受け入れ、次のような JSON ペイロードを応答するものとします。

```
{ "name": "world" }
```

作成するプロジェクトの最終的な構成は以下のようになります。（パッケージは任意）

```
.
│  pom.xml
│
└─src
    ├─main
    │  ├─java
    │  │  └─path
    │  │      └─to
    │  │          └─tutorial
    │  │              └─restfulwebservice
    │  │                      HelloApplication.java
    │  │                      HelloRecord.java
    │  │                      HelloResource.java
    │  │
    │  ├─resources
    │  └─webapp
    └─test
        ├─java
        └─resources
```

アプリケーションクラス

サービス名は `Application` クラスを継承したサブクラスで次のように宣言します。ここでは `restfulservice` という名前を指定しています。
配備するモジュールに `Application` のサブクラスが存在する場合、モジュール内のリソースクラスが検索され、Web リソースとして公開されます。

```
import jakarta.ws.rs.ApplicationPath;
import jakarta.ws.rs.core.Application;

@ApplicationPath("restfulservice")
public class HelloApplication extends Application {
}
```

リソースクラス

Jakarta EE アプリケーションでは、クライアントとの対話のターゲットを識別するリソースを公開するためのリソースクラスを作成します。`HelloResource.java` は以下のようになります。

```
package path.to.sample;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.QueryParam;
import jakarta.ws.rs.core.MediaType;

@Path("hello")
public class HelloResource {

    @GET
    @Produces({ MediaType.APPLICATION_JSON })
    public HelloRecord hello(@QueryParam("name") String _name) {
        if (null == _name || _name.trim().isEmpty()) {
            _name = "world";
        }

        return new HelloRecord(_name);
    }
}
```

提供するリソースとサービスは、グローバルアドレス空間を提供する URI によって識別されます。

`@Path` アノテーションは、ユーザが指定した URL とリクエストの処理を担当する Java クラスとの間の接続を確立します。

`@GET` アノテーションは、Jakarta REST によって定義された実行時アノテーションの一つであり、同様の名前の HTTP メソッドに対応し、上記のコードではユーザがリソースにアクセスするには `HTTP GET` メソッドが必要であることを示します。Jakarta REST では、一般的な HTTP メソッドである GET、POST、PUT、DELETE、及び HEAD の一連のリクエストメソッド指定子が用意されています。

`@Produces` アノテーションは、HTTP リクエストまたはレスポンスの MIME メディアタイプを指定することができる。上記のコードでは、JSON 形式の応答を返すための `application/json` を指定します。
戻り値の `HelloRecord` オブジェクトを JSON にシリアライズしてレスポンスが生成されます。

`@QueryParam` アノテーションは、リクエスト URI からクエリパラメータを抽出して引数に指定することができます。

レコードクラス

`hello(String)` メソッドは `HelloRecord` を返すように定義されています。`record` は Java 16 で追加された新しいレコードクラスです。

```
public record HelloRecord(String _name) {
}
```

レコードクラスを従来の POJO にする場合は以下のようになります。

```
public final class HelloRecord {
    private final String name;

    public HelloRecord(String _name_) {
        this.name = _name_;
    }

    public String name() {
        return this.name;
    }
}
```

プロジェクトの設定

Maven を使用して CLI からプロジェクトを実行する方法を説明します。
このチュートリアルでは `WildFly` を使用しますが、Jakarta EE と互換性のある他のランタイムはこちらのサイトで確認できます。
実行のためには、`pom.xml` ファイルに依存関係とプラグインを追加する必要があります。

```
...
    <packaging>war</packaging>
...
    <dependencies>
        <dependency>
            <groupId>jakarta.platform</groupId>
            <artifactId>jakarta.jakartaee-web-api</artifactId>
            <version>10.0.0</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
...
    <build>
        <finalName>${project.artifactId}</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.13.0</version>
            </plugin>
            <plugin>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.4.0</version>
                <configuration>
                    <failOnMissingWebXml>false</failOnMissingWebXml>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.wildfly.plugins</groupId>
                <artifactId>wildfly-maven-plugin</artifactId>
                <version>4.2.2.Final</version>
                <configuration>
                    <version>31.0.1.Final</version>
                    <server-config>standalone-full.xml</server-config>
                </configuration>
            </plugin>
        </plugins>
    </build>
...
```

wildfly-maven-plugin は、Jakarta EE アプリケーションのデプロイ、再デプロイ、アンデプロイ、または実行に使用されます。

プロジェクトの実行

WildFly のローカルインスタンスを実行する方法はいくつかありますが、通常の実行は wildfly-maven-plugin の Run Examplesを、開発時の実行は Dev Examples を参照してください。
このチュートリアルでは下記の Maven ゴールを使用します。

```
mvn clean package wildfly:run
```

上記の Maven ゴールはアプリをビルドし、Wildfly に配置します。Wildfly がインストールされていない場合は、`version` で指定したバージョンの Wildfly を自動的にダウンロードして実行し、war ファイルがデプロイされます。`version` を省略した場合は最新の安定板がダウンロードされます。

動作確認

WildFly が正常に起動した場合、サービスが実行されているので下記の URL にアクセスするとレスポンスが返されます。

```
http://127.0.0.1:8080/restful-web-service/restfulservice/hello
または
http://127.0.0.1:8080/restful-web-service/restfulservice/hello?name=XYZ
```

コマンドラインから以下のように確認することもできます。

```
curl -v http://127.0.0.1:8080/restful-web-service/restfulservice/hello
*   Trying 127.0.0.1:8080...
* Connected to 127.0.0.1 (127.0.0.1) port 8080
> GET /restful-web-service/restfulservice/hello HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/8.4.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Connection: keep-alive
< Content-Type: application/json
< Content-Length: 16
< Date: Thu, 02 May 2024 06:53:53 GMT
<
{"name":"world"}* Connection #0 to host 127.0.0.1 left intact
```

URL の構造は以下のようになっています。

```
http://<hostname>:<port>/<context-root>/<REST-config>/<resource-config>
```

URL の各パターンは下記の通りです。

URL パターン	説明
hostname	WildFly が実行されているサーバのホスト名または IP アドレス
port	WildFly の HTTP 受信ポート。デフォルトは 8080
context-root	アプリケーションに割り当てられるコンテキストルート。デフォルトは WAR ファイル名（拡張子除く）
REST-config	`@ApplicationPath` アノテーションに指定した値。未指定の場合は単に省略される
resource-config	`@Path` アノテーションに指定した値

２－２．Servlet Application

この項では、Jakarta Servlet を使用して単純な Servlet アプリケーションを構築する方法を追って説明します。

サーブレットの例

Jakarta EE でサーブレットを作成する手順について説明します。
最初に `HttpServlet` を拡張する新しいクラスを作成します。次に、サーブレットで処理する HTTP メソッド（GET リクエストを処理する場合は `doGet`、POST リクエストを処理する場合は `doPost`）をオーバーライドします。
例えば、GET リクエストで単純なテキストを取得するサーブレットを作成するには、次のように `HttpServlet` を拡張して `doGet` メソッドをオーバーライドします。

```
import java.io.IOException;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class HelloWorldServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest _req, HttpServletResponse _res) throws
ServletException, IOException {
        _res.getWriter().println("Hello, World!");
    }
}
```

`@WebServlet` アノテーションは、サーブレットクラスと URL マッピングを定義するために使用されます。クラス定義に `@WebServlet("hello")` がある場合、このサーブレットがアプリケーションのコンテキストルート内の `/hello` パスを対象とする HTTP リクエストに応答することを示します。

HTML フォーム

サーブレットにリクエストを送信するための `form` 要素を含む HTML ファイル `coffee_preferences.html` を `WEB-INF` ディレクトリに配置します。

```
<!DOCTYPE html>
<html>
  <head>
     <title>Coffee Preferences</title>
  </head>
  <body>
    <form action="storePreferences" method="post">
       <p>Select your coffee preferences:</p>
       <input type="checkbox" name="coffeeType" value="Black"> Black<br>
       <input type="checkbox" name="coffeeType" value="Latte"> Latte<br>
       <input type="checkbox" name="coffeeType" value="Cold Brew"> Cold Brew<br>
       <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Jakarta EE における `WEB-INF` ディレクトリは、クライアントが直接アクセスすることができないリソースを配置する安全なディレクトリとして機能します。ファイルを `WEB-INF` ディレクトリに配置すると、クライアントのブラウザからの直接 URL 指定ではアクセスできなくなります。これにより、サーブレットを介してのみアクセスできるリソースに追加のセキュリティ層が提供されます。

設定を保存するサーブレット

`coffee_preferences.html` で選択された内容をセッションに保存するサーブレットを作成します。

```
import java.io.IOException;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

@WebServlet("/storePreferences")
public class StorePreferencesServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest _req, HttpServletResponse _res) throws
ServletException, IOException {
        _req.getRequestDispatcher("/WEB-
INF/coffee_preferences.html").forward(_req, _res);
    }

    @Override
    protected void doPost(HttpServletRequest _req, HttpServletResponse _res)
throws ServletException, IOException {
        final String[] coffeeTypes = _req.getParameterValues("coffeeType");
        final HttpSession session = _req.getSession();
        session.setAttribute("userCoffeeTypes", coffeeTypes);
        _res.sendRedirect("coffeeDashboard");
    }
}
```

このサーブレットは、`@WegServlet` アノテーションを介して `/storePreferences` URL にマッピングされます。doGet メソッドは、リクエストを `/WEB-INF/coffee_preferences.html` にある HTML ページに転送します。この HTML には `form` 要素が含まれています。doPost メソッドは、送信されたフォームからユーザが選択した設定を取得し、属性名 `userCoffeeTypes` として HTTP セッション内に文字列配列として保存されます。設定が保存されると、ユーザは `coffeeDashboard` サーブレットにリダイレクトされます。

パーソナライズされたコーヒーの推奨事項を生成するサーブレット

次に、保存されたコーヒーの好みを読み取り、推奨されるコーヒーのリストを動的に生成する別のサーブレットを作成します。

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.HashMap;
import java.util.Map;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

@WebServlet("/coffeeDashboard")
public class CoffeeDashboardServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private static final Map<String, String> COFFEE_DESCRIPTIONS = new HashMap<>
();
    static {
        COFFEE_DESCRIPTIONS.put("Black", """
                    <p>Black coffee has a robust flavor, perfect for those who
prefer a coffee with some bite.</p>
                    <p>Try brewing methods like French Press or Aeropress for an
enjoyable black coffee experience.</p>
                """);
        COFFEE_DESCRIPTIONS.put("Latte", """
                    <p>A latte is a creamy delight, suitable for people who enjoy
a smoother and less harsh flavor.</p>
                    <p>Experimenting with various syrups and sweeteners can
elevate your latte experience.</p>
                """);
        COFFEE_DESCRIPTIONS.put("Cold Brew",
                """
                            <p>Cold brew coffee tends to be smoother and less
acidic. It's perfect for those hot summer days.</p>
                            <p>Try brewing a batch in the fridge overnight for a
refreshing morning pick-me-up.</p>
                        """);
    }

    @Override
    protected void doGet(HttpServletRequest _req, HttpServletResponse _res) throws
ServletException, IOException {
        final HttpSession session = _req.getSession();
        final String[] coffeeTypes = (String[])
session.getAttribute("userCoffeeTypes");
```

```
        if (null == coffeeTypes || 0 == coffeeTypes.length) {
            this.handleNoCoffeeTypes(_res);
            return;
        }

        final PrintWriter out = _res.getWriter();
        out.println("""
                    <html>
                    <body>
                    <h1>Your Personalized Coffee Dashboard</h1>
                """);

        for (final String coffeeType : coffeeTypes) {
            final String additionalInfo = COFFEE_DESCRIPTIONS.get(coffeeType);
            out.println("""
                        <h2>Recommended %s</h2>
                        <p>Here are some %s blends you might enjoy.</p>
                        %s
                    """.formatted(coffeeType, coffeeType, additionalInfo));
        }

        out.println("""
                    </body>
                </html>
                """);
    }

    private void handleNoCoffeeTypes(HttpServletResponse _res) throws IOException
{
        final PrintWriter out = _res.getWriter();
        out.println("""
                <html>
                    <body>
                        <h1>No Coffee Types Found</h1>
                        <p>Please select at least one type of coffee.</p>
                    </body>
                </html>
                """);
    }
}
```

`CoffeeDashboardServlet` クラスは、ユーザのコーヒーの好みに基づいてパーソナライズされたコーヒーダッシュボードを生成します。このコードでは `COFFEE_DESCRIPTIONS` という静的 `Map` を使用して、コーヒーの説明を保存しています。

サーブレットは `doGet` メソッドをオーバーライドして HTTP GET リクエストを処理します。このメソッド内では、まず HTTP セッションに保存されているユーザのコーヒーの好みを取得します。設定が見つからない場合は、`handleNoCoffeeTypes` メソッドを呼び出してデフォルトのメッセージが表示されます。それ以外の場合は、選択したコーヒーの種類を反復処理し、`COFFEE_DESCRIPTIONS` マップから対応する説明を取得します。 最後に、この情報をパーソナライズされたダッシュボードに表示するための HTML コンテンツを生成します。

動作確認

このチュートリアルでは下記の Maven ゴールを使用します。

```
mvn clean package wildfly:run
```

WildFly が正常に起動した場合、サービスが実行されているので下記の URL にアクセスするとレスポンスが返されます。

```
http://127.0.0.1:8080/servlet-application/storePreferences
```