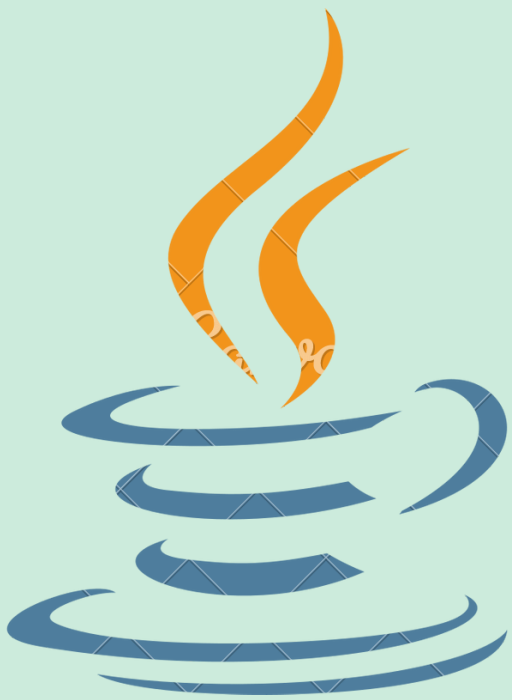
@techwithvishalraj



Java

pom.xml in a maven based Spring Project

Follow

Share

@techwithvishalraj

What is pom.xml in a maven based Spring Project?

- The full form of POM is Project Object Model.
- It is a fundamental unit of work in Maven.
- pom is a XML file that contains information about the project and configuration details used by Maven to build the project.
- When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.
- Configurations specified in the POM are the project dependencies, the plugins or goals, the build profiles.
- The pom.xml is the recipe that will be used to build your project.
- To learn in details, please refer maven site: https://maven.apache.org/pom.html

pom.xml

@techwithvishalraj

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
    XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.5.4</version>
    </parent>
    <groupId>com.example</groupId>
    <artifactId>my-spring-boot-app</artifactId>
    <version>1.0.0</version>
    <name>spring-boot-multi-module-project</name>
    <description>Demo project for Spring Boot</description>
    <packaging>pom</packaging>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <spring.version>3.3.9</spring.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <!-- Add any additional dependencies your project needs here -->
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
    <modules>
        <module>module1</module>
    </modules>
</project>
```

Let’s, go through each element.....

@techwithvishalraj

<project></project>

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">

         .....

</project>
```

- project is root element of the pom.xml.
- It describes basic schema settings such as apache schema and w3.org specification.
- xmlns:xsi = http://www.w3.org/2001/XMLSchema&#8221 indicates that the elements and data types used in the schema come from the http://www.w3.org/2001/XMLSchema namespace. It also specifies that the elements and data types that come from the http://www.w3.org/2001/XMLSchema namespace should be prefixed with xsi:
- xmlns specifies the default namespace declaration. This declaration tells the schema-validator that all the elements used in this XML document are declared in the http://maven.apache.org/POM/4.0.0&#8221 namespace
- The Java XML parser that spring uses will read the schemaLocation values and try to load them from the internet, in order to validate the XML file. Spring, in turn, intercepts those load requests and serves up versions from inside its own JAR files.
- If you omit the schemaLocation, then the XML parser won’t know where to get the schema in order to validate the config.

@techwithvishalraj

Application description tags

```
<groupId>com.example</groupId>
<artifactId>my-spring-boot-app</artifactId>
<version>1.0.0</version>
<name>spring-boot-multi-module-project</name>
<description>Demo project for Spring Boot</description>
```

- groupId: This is generally unique for a project. For example, all core Maven artifacts do should live under the groupId – org.apache.maven.
- artifactId: The artifactId is generally the name that the project is known by.
- version: defines the version of the project.
- name: Name of the app.
- description: description of the app.

Properties

```
<properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <spring.version>3.3.9</spring.version>
</properties>
```

- It is used to define and set project-specific properties or variables that can be referenced throughout the POM file and its associated build process. These properties can help make the POM file more maintainable and flexible by centralizing configuration values and reducing redundancy.
- <spring.version> is project-specific properties. These properties can be referenced elsewhere in the pom.xml file, such as in the <dependencies> section, where you can specify the versions of Spring you want to use, e.g. <version>${spring.version}</version>

@techwithvishalraj

POM Relationships

- dependencies
- inheritance
- aggregation

Dependency Management

```
<dependencyManagement>
        <dependencies>
            <!-- Define managed dependencies here -->
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
                <version>4.12</version>
                <classifier>sources</classifier>
                <type>jar</type>
                <scope>compile</scope>
                <optional>true</optional>
                <systemPath>/path/to/example-library.jar</systemPath>
                <exclusions>
                    <exclusion>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-tomcat</artifactId>
                    </exclusion>
                </exclusions>
            </dependency>
            <!-- Add more managed dependencies as needed -->
        </dependencies>
</dependencyManagement
```

- Dependency Management is used by a POM to help manage dependency information across all of its children. it allows to consolidate and centralize the management of dependency versions without adding dependencies which are inherited by all children.
- Another extremely important use case of Dependency Management is the control of versions of artifacts used in transitive dependencies.
- Maven downloads and links the dependencies on compilation, as well as on other goals that require them.
- Inside <dependency> – you define what all modules and jars you will require in your project framework.
- Each dependency is simply an another maven project which is defined by the groupId, artifactId, version etc.
- Apart from above three there are several more tags used which are as follows:
  - classifier: The classifier tag allows you to distinguish between different versions or variants of the same artifact. It is often used when a single project produces multiple artifacts with different characteristics (e.g., source JAR, test JAR, or different distributions). The classifier is used to specify which variant of the artifact you want to depend on.
  - type: Corresponds to the chosen dependency type. This defaults to jar.
  - scope: This element refers to the classpath of the task at hand (compiling and runtime, testing, etc.) as well as how to limit the transitivity of a dependency. There are five scopes available: compile, provided, runtime, test , system. compile is default.
  - systemPath: The systemPath tag allows you to specify the path to a JAR file on your local file system that should be used as a dependency.
  - optional: Marks a dependency optional when this project itself is a dependency.
  - exclusions: Exclusions contain one or more exclusion elements, each containing a groupId and artifactId denoting a dependency to exclude. exclusions actively remove artifacts from the dependency tree. if you don’t want the particular dependency you can declared the groupID & artifactId in this tag. also if you want to change the version of the dependency then also you can exclude particular dependency and add dependency with updated version.

@techwithvishalraj

Inheritence in maven

```
...
    <packaging>pom</packaging>
...
```

- The packaging type required to be pom for parent and aggregation (multi-module) projects.
- When no packaging is declared, Maven assumes the packaging is the default: jar. but you can change it war or any other types such as pom, jar, maven-plugin, ejb, war, ear, rar.

Aggregation or Multi Module

```
...
    <modules>
        <module>my-project</module>
        <module>another-project</module>
        <module>third-project/pom-example.xml</module>
    </modules>
...
```

- A project with modules is known as a multi-module, or aggregator project. Modules are projects that this POM lists, and are executed as a group.

Build

@techwithvishalraj

- **<build>:** it is used to configure various aspects of the build process for your project. It allows you to specify plugins and their configurations that are used to build and package your application.
- **<plugins>:** This is the container for specifying Maven plugins that should be used during the build process.
- **<resource>:** This element represents a specific resource directory or location that you want to include in your project's build output.
- **<directory>:** Specifies the directory containing the resources. In this example, src/main/resources is the default directory for placing application resources in a Maven project.
- **<includes>**: You can specify one or more <include> elements to filter which files should be included as resources. The <include> elements use Ant-style patterns to match files. In this example, **/*.properties and **/*.xml include all .properties and .xml files recursively within the specified directory.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>2.6</version>
            <configuration>
                <classifier>test</classifier>
            </configuration>
        </plugin>
     </plugins>
    <resources>
        <resource>
            <directory>src/main/resources</directory>
                <includes>
                    <include>**/*.properties</include>
                </includes>
        </resource>
    </resources>
</build>
```

@techwithvishalraj

Thank you!

vishal-bramhankar

techwithvishalraj

Vishall0317