CODEGYM

Raising the bar

Packaging & Dependences Management for Java Web Applications

1. Đóng gói ứng dụng Java

Mục tiêu

- Trình bày được khái niệm đóng gói
- Trình bày được khái niệm thư viện phụ thuộc
- Trình bày được các định dạng đóng gói ứng dụng Java
- Trình bày được các công cụ đóng gói ứng dụng Java phổ biến
- Biết cách sử dụng Maven để quản lý thư viện phụ thuộc
- Biết cách sử dụng Maven để đóng gói ứng dụng Java
- Biết cách sử dụng Gradle để quản lý thư viện phụ thuộc
- Biết cách sử dụng Gradle để đóng gói ứng dụng Java

Gói phần mềm chuyển giao được

- Trong đa số trường hợp, không thể chuyển giao mã nguồn cho khách hàng
- Hệ thống của khách hàng (môi trường production) là môi trường thực thi, không phải môi trường phát triển
- Môi trường production không có những công cụ đặc thù của môi trường phát triển như *compiler/debugger/testing tool*...
- Gói phần mềm chuyển giao tới khách hàng phải ở trạng thái sẵn sàng để hoạt động

Những tài nguyên cần thiết để ứng dụng có thể hoạt động

JRE cần có những tài nguyên sau để có thể khởi động và duy trì thực thi ứng dụng:

- Tập tất cả các class cần thiết, nằm dưới dạng file *.class* (byte-code)
- Tập tất cả các tài nguyên khác: *cấu hình, message bundle, HTML, CSS, static files, ...*

Những file .class cần thiết

- Java API và các thư viện builtin, sẵn có trong JRE. Không cần thiết phải bổ sung vào gói chuyển giao.
- Thư viện phụ thuộc build sẵn. Có thể cần bổ sung vào gói chuyển giao trong trường hợp môi trường production không có sẵn.
- Mã byte-code của bản thân chương trình, có được sau quá trình compile. Bao gồm byte-code của chương trình chính – luôn cần bổ sung vào gói chuyển giao, và test cases – không bổ sung vào gói chuyển giao.

Gói phần mềm Java

Gói phần mềm Java có thể được đóng gói và chuyển giao tới môi trường thực thi (hệ thống) dưới hai hình thức:

- Một gói độc lập (**standalone**), được thực thi bởi JRE của hệ thống.
- Một gói tự túc (**self-contained**): bao gồm standalone application, cộng thêm một phiên bản JRE riêng, có khả năng tự khởi động được.

Lưu ý: còn có định dạng *Java Web Start application*, và *Java embered* không được kể tới ở đây.

2. Công cụ để đóng gói

Thư viện phụ thuộc (Dependence)

- Là các cấu phần chương trình tới từ bên ngoài mà đảm nhiệm một nhiệm vụ cụ thể. Ví dụ: :JDBC, Thymeleaf, Spring...

Công cụ quản lý thư viện phụ thuộc (Dependence Management tool)

- Là những phần mềm giúp tích hợp thư viện phụ thuộc vào bản thân chương trình đang được phát triển. Ví dụ *Composer*, *NPM*, *Ant*, *Maven*, *Gradle*...
- Sử dụng những file cấu hình (chẳng hạn *composer.json, package.json, pom.xml, build.gradle...*) để làm rõ:
  - Những phụ thuộc nào cần tìm về
  - Phiên bản nào cần tìm về
  - Tìm các phụ thuộc về từ repository nào

Dependence Repository

- Kho chứa các thư viện phụ thuộc của các Công cụ Quản lý Thư viện phụ thuộc.
- NPM Repository, Maven Central, Packagist... là các ví dụ.

Tại sao cần dùng Công cụ Quản lý Thư viện phụ thuộc

- Đảm bảo môi trường phát triển (Dev Environment) và môi trường thực thi (Production Environment) sử dụng cùng một phiên bản của Thư viện phụ thuộc.
- Giúp việc quản lý (bổ sung, thay thế, cập nhật) các thư viện phụ thuộc trở nên dễ dàng hơn

Building và Build tool

- **Building** là quá trình đóng gói tất cả các tài nguyên cần thiết thành một chương trình hoạt động được.
- **Build tool** là những chương trìnhg có thể tự động hóa tiến trình build. Ví dụ: Intellij builder, Ant, Maven, Gradle, Gulp, Webpack...
- Các nhiệm vụ thường gặp của build tool:
  - Compile chương trình
  - Thực thi các kiểm thử
  - Đóng gói chương trình cùng các thư viện phụ thuộc
  - Deploy
  - Một số build tool đảm nhiệm thêm cả chức năng của một Trình quản lý thư viện phụ thuộc

Ant, Maven, Gradle

- **Apache Ant**: một công cụ viết bằng Java, được xây dựng với mục đích làm một build tool uyển chuyển cho các ứng dụng Java. Sử dụng định dạng XML cho tập tin cấu hình.
- **Apache Maven**: một công cụ quản lý thư viện phụ thuộc kiêm build tool cho các ứng dụng Java. Khả năng quản lý tốt hơn so với Ant, nhưng kém uyển chuyển hơn. Sử dụng định dạng XML cho tập tin cấu hình.
- **Gradle**: một công cụ quản lý thư viện phụ thuộc kiêm build tool cho các ứng dụng Java. Được xây dựng trên concept của Ant và Maven. Sử dụng ngôn ngữ Groovy cho tập tin cấu hình.

Các thông tin trong tập tin cấu hình Gradle

- Các plugin được áp dụng
- Các Dependency Repository
- Các Dependency
- Các Task

Cấu hình Gradle: Plugins

- Gradle sử dụng thiết kế “mọi chức năng đều là plugin”.
- Mục plugin khai báo các chức năng (có sẵn hoặc tới từ dependence) muốn sử dụng

```
plugins {
    id 'java'
    id 'war'
    id 'jacoco'
    id "org.flywaydb.flyway" version
'5.1.4'
}
```

Cấu hình Gradle: Repository

Có thể sử dụng Maven Central hoặc repository khác

```
repositories {
    mavenCentral()
    maven { url 'https://jitpack.io' }
}
```

Cấu hình Gradle: Dependencies

Mỗi dependency đi kèm với vị trí, phiên bản, và scope khả dụng.

```
dependencies {
    // Lombok's Buider
    compile 'org.projectlombok:lombok:1.18.10'
    // Logging
    compile 'org.slf4j:slf4j-api:1.7.25'
    compile 'ch.qos.logback:logback-classic:1.2.3'
    // DB Migration
    providedCompile 'org.flywaydb:flyway-core:5.1.4'
}
```

Các Dependency Scope quan trọng

- **Compile**: dependency sẽ được đóng gói cùng với chương trình. Dependency cũng sẽ khả dụng tại mọi ngữ cảnh thực thi dự án: compile, test, execution...
- **Provided**: ngầm hiểu rằng dependency sẽ tồn tại tại ngữ cảnh thực thi chương trình trong tương lai, do đó không cần đóng gói cùng chương trình. Nhưng dependency vẫn khả dụng tại ngữ cảnh compile, test.
- **Test**: dependency không được sử dụng trong ngữ cảnh thực thi chương trình trong tương lai, do đó không cần đóng gói cùng chương trình. Dependency chỉ khả dụng trong ngữ cảnh test và compile test classes.

Cấu hình Gradle: Task

Là các đơn vị công việc có thể phải thực thi trong tiến trình build. Chẳng hạn như compile, tạo document, chạy kiểm thử, ...

```
jacocoTestReport {
    reports {
        xml.enabled false
        csv.enabled false
        html.destination file("${buildDir}/jacocoHtml")
    }
}
```

3. Sản phẩm của tác vụ đóng gói

.JAR file

- Một định dạng tập tin phổ biến của gói phần mềm/thư viện Java
- Là một tập tin nén chứa các file *.class*, *resource* và *metadata*
- Được tạo ra bởi công cụ **"jar"** trong bộ công cụ JDK, hoặc bằng công cụ đóng gói

```
META-INF/
   |--MANIFEST.MF
com/
   |--apwj/
        |--MyApplication.class
```

.WAR file

- Định dạng phổ biến để đóng gói Java Web Application
- Chứa các tài nguyên web, cùng với các class của application được dùng như các thư viện cho *servlet container*
- Được gọi thực thi bởi *servlet container*

```
META-INF/
    MANIFEST.MF
WEB-INF/
    web.xml
    jsp/
        helloWorld.jsp
    classes/
        static/
        templates/
        application.properties
    lib/
        // *.jar files as libs
```

Tổng kết

- Đóng gói và quản lý phụ thuộc là tiến trình quan trọng xuyên suốt hoạt động phát triển phần mềm.
- Có nhiều công cụ đóng gói, phổ biến trong hệ sinh thái Java là Ant, Maven, Gradle
- Gradle sử dụng tập tin cấu hình là build.gradle, được viết bằng ngôn ngữ Groovy
- Tập tin JAR và WAR là hai định dạng đóng gói Java Application phổ biến. JAR dùng để đóng gói nhiều loại thư viện/ứng dụng. WAR dùng để đóng gói ứng dụng Web.

Raising the bar