Application frameworks

Introduction to Maven and Spring Boot

Overview

- Build Tools
- Maven
- Spring Framework
- Spring Boot

Source : https://www.overviewdocs.com/

Build Tools

- What ?
- Why is it used?
- Ant vs Maven

What ?

- A tool for setting up and automate the development process.
- Have no use once the application is built and deployed.
- There are different build tools that can be used with Java.
    - » Ant
    - » Maven
    - » Gradle

Why is it used ?

- **Developer Environments**: where you are interactively working and changing a codebase, several times in a single hour, and want it to execute many small actions (whether manually or via unit tests) in between modifying the code.
- **Continuous Integration**: where you are running a relatively large test suite on a codebase, perhaps once every several minute, with the code only changing between runs of the suite.
- **Production Deployment**: Where you want to set up the codebase once, stop changing it, and ship the finished product. Whether you are shipping it to web servers or shipping it on CDs to install on your customers' laptops.

Ant vs Maven

- Both are two popular build tools used by the Java community.
- Ant is the predecessor of Maven and is the foundation for maven.
- Maven is built on the concept of “**Convention over Configuration**”.
- In Ant, all the dependent libraries should be copied manually to the project location to setup the project
  - **Ivy** was another tool being introduced as a solution.
- Maven resolves dependencies recursively and download automatically from a remote location (A central repository of Java libraries).

Maven

- Why Maven
- Phases of a project
- Archetypes
- Using Archetypes
- Dependencies
- Plugins
- Multiple modules
- Repositories

Why Maven ?

- “Convention over Configuration”
- Provides help with managing
  - Builds
  - Documentation
  - Reporting
  - Dependencies
  - Releases
  - Distribution
- Builds can be automated. (Compiling, packaging, testing and deployments)

Why Maven ?

- Dependencies are downloaded from a central repository.
- Dependencies are copied to a local repository to be reused.
- Recursive (transitive) dependencies are resolved automatically.
- Contain pre-configured project presets called archetypes.
- Easier management in project modules and project interdependencies.

Phases of a Project

- Predefined phases.
  - **clean**: Cleanup the artifacts generated by previous builds.
  - **compile**: Compile the source code.
  - **package**: Create a distributable bundle.
  - **install**: Copy the package into the local repository.
  - **deploy**: Copy the final package to the release environment or publish to a remote repository.
  - **test**: Execute the test cases.

Archetypes

- A templating toolkit.
- An original pattern or model from which all other things of the same kind are made.
- Used to setup a project with sample code as quickly as possible.
- Can define custom archetypes including the things for your own requirements.
- Contain a huge repository of archetypes. Following are two common ones out of them.
    - » maven-archetype-quickstart
    - » maven-archetype-webapp

Using Archetypes

- Generating a sample project.

```
mvn archetype:generate -DgroupId=com.mycompany.app
-DartifactId=my-app
-DarchetypeArtifactId=maven-archetype-quickstart
```

  - **groupId**: Package name of the project
  - **artifactId**: Project name
  - **archetypeArtifactId**: name of the archetype to be used

Generated Directory Structure

```
my-app
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- com
    |           `-- mycompany
    |               `-- app
    |                   `-- App.java
    `-- test
        `-- java
            `-- com
                `-- mycompany
                    `-- app
                        `-- AppTest.java
```

POM (Project Object Model)

- **'pom.xml'** contains the Project Object Model for the project.
- POM contains every important piece of information about the project.

```
<groupId>com.mycompany.app</groupId>
<artifactId>my-app</artifactId>
<packaging>jar</packaging>
<version>1.0-SNAPSHOT</version>
<name>Maven Quick Start Archetype</name>
```

- Generate IDE specific project descriptor

```
mvn idea:idea
mvn eclipse:eclipse
```

Dependency Management

- Another feature of the POM

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
        <type>jar</type>
        <optional>true</optional>
    </dependency>
</dependencies>
```

Dependency Management

–type

Packaging type of the project (**jar, war, ear**)

–version

Version of the project.

“SNAPSHOT”: A beta version in changing phase

Downloaded when a maven phase is executed in a timely manner

–optional

Marks as optional when the project itself is a dependency.

Dependency Management

–scope

Classpath of the task (compiling, runtime or test execution)

Different scopes are as following.

“compile”

Default scope.

Dependencies are available in all classpaths.

Propagated to dependent projects.

“test”

Only available for test compilation and execution.

These are not transitive

Dependency Management

- –scope
  - “provided”
    - Expect the JDK or the container to provide at runtime.
  - “runtime”
    - Not needed for compilation but for runtime and test execution.
  - “system”
    - Similar to provided but need to specify the JAR file.
- –systemPath
  - Used when dependency scope is `system`.
  - Example: `${java.home}/lib`

Plugins

- Plugins are used to customise the build for a project.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.3</version>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Maven Jar Plugin

- Used to set the manifest for the JAR file.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <configuration>
                <archive>
                    <manifest>
                        <addClasspath>true</addClasspath>
                        <mainClass>com.mycompany.app.App</mainClass>
                    </manifest>
                </archive>
            </configuration>
        </plugin>
    </plugins>
</build>
```

Modules

- Can contain a project with multiple modules.
- Dependency modules are always build before dependent modules.

```
<modules>
    <module>app</module>
    <module>util</module>
</modules>
```

- Have to mention the parent information in the sub module.

```
<parent>
    <groupId>com.xyz</groupId>
    <artifactId>multi-app</artifactId>
    <version>1.0</version>
</parent>
```

Adding a Property file (resource) to Project

- All project resources are maintained in separate resource directories inside main.

```
src -> main -> resources

src -> test -> resources
```

- Most of the times, property files which contain application configuration are stored in these locations.
- Can also be explicitly mentioned also in the `<resources>` section of the POM.

Maven Repositories

- Collections of artifacts which adhere to the Maven repository directory layout.

```
<repositories>
    <repository>
        <id>codehausSnapshots</id>
        <name>Codehaus Snapshots</name>
        <url>http://snapshots.maven.codehaus.org/maven2</url>
        <layout>default</layout>
    </repository>
</repositories>
```

- Maven first search the local repo for artifacts.
- If not found move the defined remote alternatives.
- Default central repository lives in:

  https://repo.maven.apache.org/maven2/

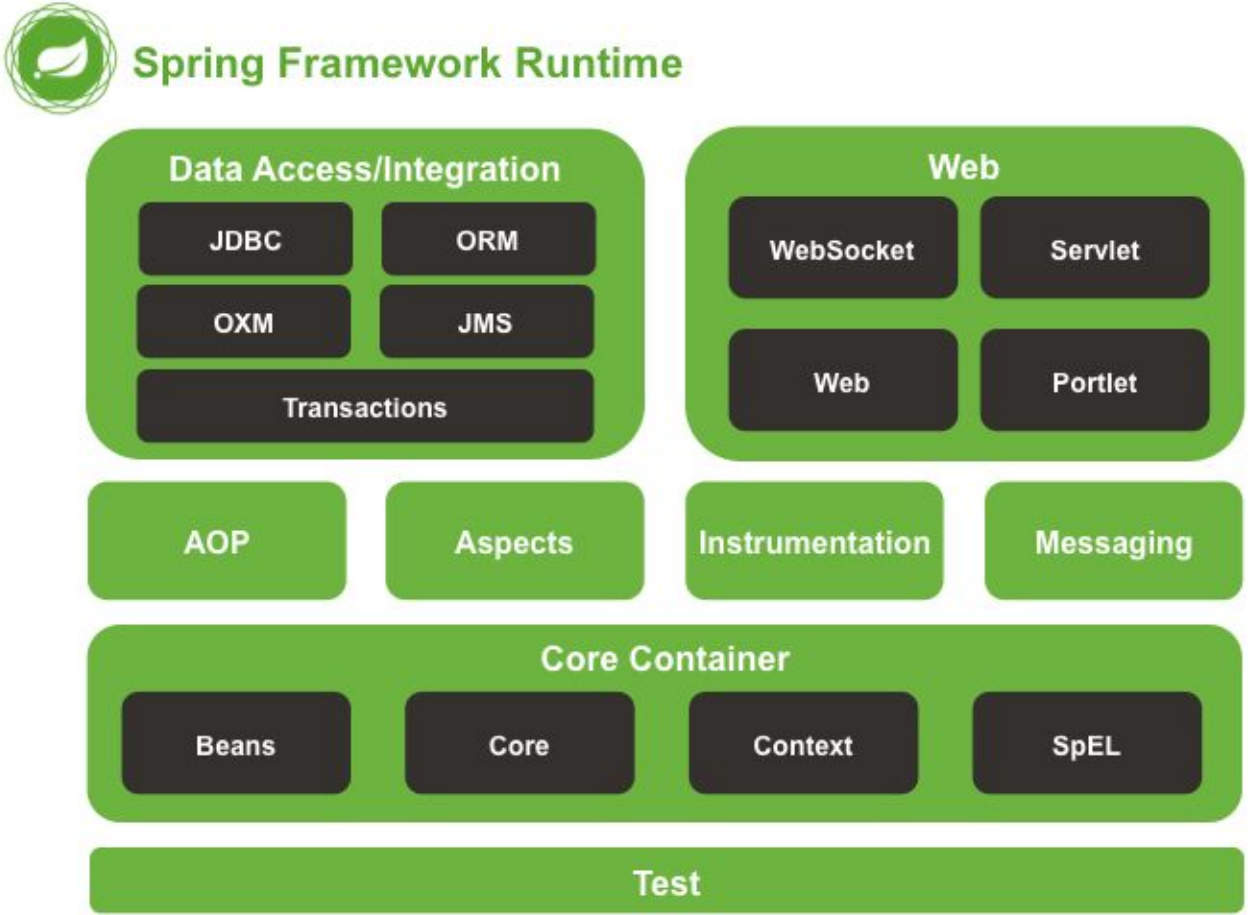
Spring Framework

- What ?
- Dependency Injection (DI)
- IoC - Inversion of Control

What ?

- A Java platform that provides comprehensive infrastructure support for developing Java applications.
- Handles the infrastructure so you can focus on the application.
- Enables you to build applications from "plain old Java objects" (POJOs)
- Apply enterprise services non-invasively to POJOs
- Doesn’t require EJB container product like the application servers.
- Was mostly famous for Dependency Injection (**DI**) with the favour of Inversion of Control (**IoC**)

Overview


Spring Framework Runtime
Data Access/Integration
JDBC
ORM
OXM
JMS
Transactions
Web
WebSocket
Servlet
Web
Portlet
AOP
Aspects
Instrumentation
Messaging
Core Container
Beans
Core
Context
SpEL
Test

Dependency Injection (DI)

- Inject the dependencies into classes and remove the logic of creating and linking them out of the class.
- Gain low coupling by removing the couplings between entities.
- If Class B is a dependency for class A, framework will create the instance of class B and inject it into class A.
- Bundled in **Spring IoC** module.

Inversion of Control (IoC)

- DI is a specific type of IoC.
- Little bit of homework ?
- Small conceptual example.

```
public class TextEditor {
    private SpellChecker checker;
    public TextEditor() {
        this.checker = new SpellChecker();
    }
}
```

```
public class TextEditor {
    private ISpellChecker checker;
    public TextEditor(ISpellChecker checker) {
        this.checker = checker;
    }
}
```

Using DI

- Adding maven dependency.

```
<dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.0.0.RELEASE</version>
</dependency>
```

- Two ways to inject dependencies.
  - Setter Based
  - Constructor Based
- But it works for Fields also. (?)

Using DI

- Setter Based

```
@Autowired
public void setService(MessageService svc){
        this.service=svc;
}
```

- Constructor Based

```
@Autowired
public MyApplication(MessageService svc){
        this.service=svc;
}
```

- Fields

```
@Autowired
private MessageService service;
```

Few Mostly Used Annotations

- @Component

  Notify Spring that this class is a component of the application.

- @Autowired

  Let Spring know that autowiring (injections) are required.

- @Service

  Tells Spring the class is a component in Service layer.

- @Repository

  Tells Spring the class is a component in Repository (DAO) layer.

- @Controller

  Tells Spring the class is a Controller in MVC architecture

More Theories
But
Not Today

Spring Boot

- Why ?
- What it brings ?
- Starters
- How To Use It ?
- Small microservice

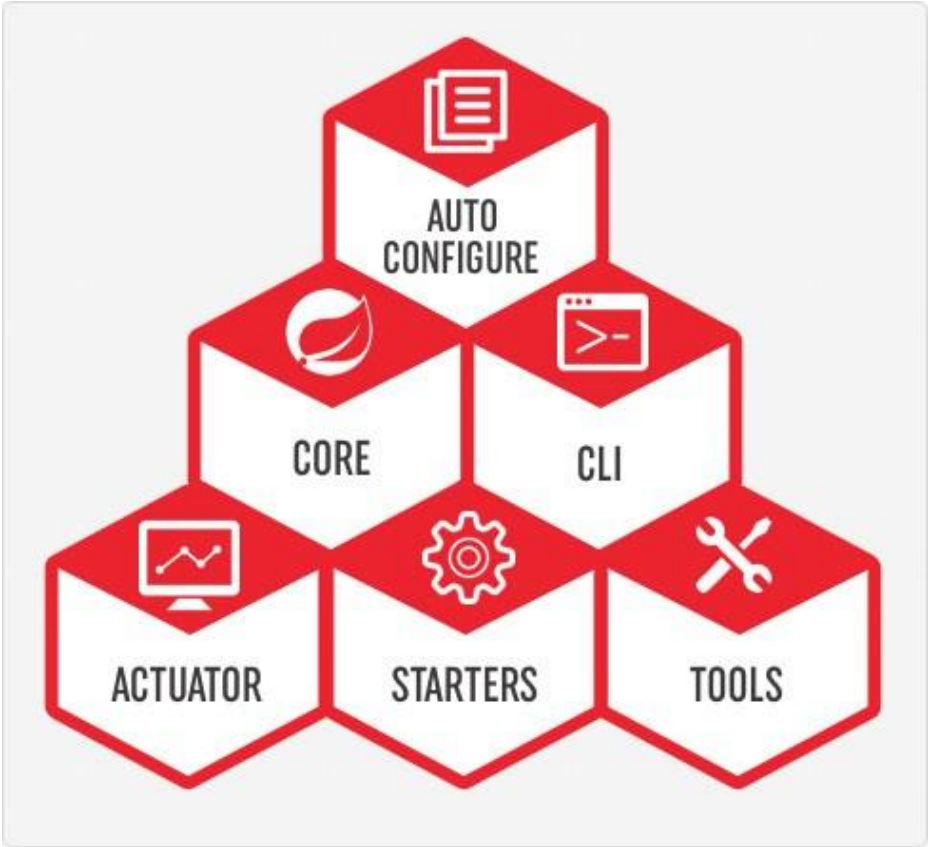
Why ?

- Spring is a great framework.
- But lot of configurations and boilerplate code are harder to manage and bootstrapping a project involves lot of time.
- Integration of frameworks or libraries are harder.
- No clear support for microservices.

What it brings ?

- Convention over Configuration.
- Standardization for Microservices.
- Integrated (embedded) server for development.
- Support for 3rd party libraries.
- Easier integration with other frameworks.

What it brings ?



Spring Boot Starters

- Incorporates many starters packages (in Maven & Gradle).
- Easy to include them in project as dependencies through Maven or Gradle.
- Starter packages comes with full integrations with the tools mentioned in each of the specific package.
- All the starters can be found in spring boot website.

```
spring-boot-starter-web
spring-boot-starter-data-mongodb
spring-boot-starter-hateoas
spring-boot-starter-test
```

How to use it ?

- Set the parent of the project to **`spring-boot-starter-parent`**.

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.2.RELEASE</version>
</parent>
```

- Using a starter

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

How to use it ?

- Bootstrapping the application.

```
@SpringBootApplication
public class Application {
    public static void main(String[] args ){
        SpringApplication.run(Application.class, args);
    }
}
```

How to use it ?

- Maven plugin for packaging support.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

How to use it ?

It’s only coding left

Lab Sessions

- All sessions are continuous
- Push them to a git repository
  (GitHub/Bitbucket)
- Use the same repository all the time

Any Questions?