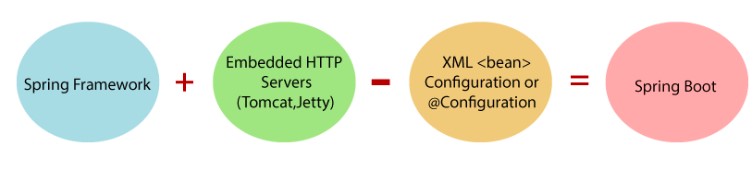
**Spring Boot**

1. **Introduction**

Spring boot is a Spring module which provides RAD (Rapid Application Development) feature to Spring framework. It is used to create stand alone spring based application that you can just run because it needs very little spring configuration.



We should use Spring boot framework because:

* The dependency injection approach is used in Spring boot
* It contains powerful database transaction management capabilities
* It simplifies integration with other Java framework like JPA/Hibernate ORM, Struts, etc.
* It reduces the cost and development time of the application.

Along with the Spring Boot framework, many other Spring sister project help to build applications addressing modern business needs. There are the following Spring sister projects are as follows:

* **Spring Data**: it simplifies data access from the relational and **NoSql** databases.
* **Spring Batch**: it provides powerful **batch** processing.
* **Spring Security**: it is a security framework that provides robust **security** to applications.
* **Spring Social:** it supports integration with **social networking** like LinkedIn.
* **Spring Integration:** it is an implementation of Enterprise Integration Patterns. It facilities integration with other **enterprise application** using lightweight messaging and declarative adapters.

1. **Advantage**

* Create stand-alone Spring applications that can be started using java –jar.
* Embed Tomcat, Jetty or Undertow directly. You don’t need to deploy WAR files.
* It provides opinionated starter POMs to simplify your Maven configuration.
* It automatically configure Spring whenever possible.
* There is no requirement for XML configuration.
* It offers CLI tool for developing and testing the Spring Boot application.
* It offers the number of plug-ins.
* It also minimizes writing multiple boilerplate codes (the code that has to be included in many places with little or no alteration), XML configuration, and annotations.
* It increase productivity and reduces development time.

1. **Limitations**

Spring boot can use dependencies that are not going to be used in the application. These dependencies increase the size of application.

1. **Goals of Spring Boot**

The main goal of Spring boot is to reduce development, unit test, and integration test time.

1. Provides Opinionated Development approach.
2. Avoids defining more Annotations Configuration.
3. Avoids writing lots of import statement.
4. Avoids XML configuration.

By providing or avoiding the above points, Spring Boot framework reduces **Development** **time**, **Developer** **effort**, and **increases** **productivity**.

1. **Spring Boot Features**

* **Web Development**

It is well-suited Spring module for web application development. We can easily create a self-contained HTTP application that uses embedded servers like Tomcat, Jetty, or Undertow. We can use the spring-boot-starter-web module to start and run the application quickly.

* **Spring Application**

The *SpringApplication* is a class that provides a convenient way to bootstrap a Spring application. It can be started from the main method. We can call the application just by calling a static run() method.

* **Application event and listeners**

Spring boot uses event to handle the variety of tasks. It allows us to create factories file that is used to add listeners. We can refer it to using the ***ApplicationListener*** key.

* **Admin Support**

Spring boot provides the facility to enable admin-related features for the application. It is used to access and manage application remotely. We can enable it in the Spring boot application by using **spring.application.admin.enabled** property.

* **Externalized Configuration**

Spring boot allows us to externalize our configuration so that we can work with the same application in different environments. The application uses YAML files to externalize configuration..

* **Properties Files**

Spring boot provides a rich set of **Application Properties**. So, we can use that in the properties file of our project. The properties file is used to set properties like **server-port =8082** and many others. It helps to organize application properties.

* **YAML support**

It provides a convenient way of specifying the hierarchical configuration. It is a superset of JSON. The SpringApplication class automatically supports YAML. It is an alternative of properties file.

* **Type-safe configuration**

The strong type-safe configuration is provided to govern and validate the configuration of the application. Application configuration is always a crucial task which should be type-safe. We can also use annotation provided by this library.

* **Logging**

Spring boot uses common logging for all internal logging. Logging dependencies are managed by default. We should not change logging dependencies if no customization is needed.

* **Security**

Spring boot applications are spring bases web applications. So, it is secure by default with basic authentication on all HTTP endpoints. A rich set of endpoints is available to develop a secure Spring Boot application.

1. **Spring Vs Spring Boot Vs Spring MVC**
2. **Spring Vs Spring Boot**

|  |  |
| --- | --- |
| **Spring** | **Spring Boot** |
| **Spring Framework** is widely used Java EE framework for building applications. | **Spring Boot** framework is widely used to develop **REST API**’s. |
| It aims to simplify Java EE development that makes developers more productive | It aims shorten the code length and provide the easiest way to develop **Web Application**. |
| The primary feature of the Spring Framework is **dependency injection**. | The primary feature of Spring Boot is **Autoconfiguration**. It automatically configures the classes based on the requirement. |
| It helps to make things simpler by allowing us to develop loosely coupled applications. | It helps to create a **stand-alone** application with less configuration. |
| The developers writes a lot of code (**boilerplate code**) to do minimal task. | It reduces **boilerplate** code. |
| To test Spring project, we need to set up the server explicity. | Spring Boot offers **embedded server** such as **Jetty** and **Tomcat**, etc. |
| It does not provide support for an in memory database. | It offers several plugins for working with embedded and **in memory** database such as **H2** |
| Developers manually define dependencies for the Spring project in **pom.xml**. | Spring boot comes with the concept of starter in pom.xml file that internally takes care of downloading the dependencies JAR’s based on Spring Boot requirements. |

1. **Spring Boot Vs Spring MVC**

|  |  |
| --- | --- |
| **Spring Boot** | **Spring MVC** |
| **Spring boot** is a module of Spring of packaging the Spring based application with sensible defaults. | **Spring MVC** is model view controller based web framework under the Spring framework. |
| It provides default configurations to build **Spring powered** framework. | It provides **ready to use** features for building a web application. |
| There is no need to build configuration manually. | It requires build configuration manually. |
| There is **no requirement** for a deployment descriptor. | A deployment descriptor is **required**. |
| It avoids boilerplate code and wraps dependencies together in single unit. | It specifies each dependency separately. |
| It **reduces** development time and increases productivity. | It takes **more** time to achieve the same. |

1. **Spring Boot Project Component**
2. **Architecture**
3. **Presentation Layer**

The presentation layer handles the HTTP request, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consist of views i.e., frontend part.

1. **Business Layer**

The business layer handles all the **business logic**. It consists of service classes and uses services provided by data access layers. It also performs **authorization** and **validation**.

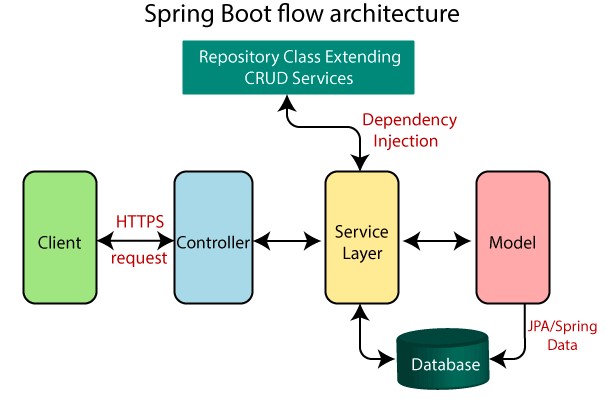
1. **Persistence Layer**

The persistence layer contains all the **storage logic** and translates business object from and to database rows.

1. **Database Layer**

In the database layer, **CRUD** (create, read, update, delete) operations are performed.

Spring Boot Flow Architecture:



1. Now we have validator classes, view classes, and utility classes.
2. Spring boot uses all modules of Spring like Spring MVC, Spring Data, etc. The architecture of Spring Boot is the same as the architecture of Spring MVC, except one thing: there is no need for **DAO** and **DAOImpl** classes in Spring Boot.
3. Creates a data access layer and performs CRUD operation.
4. The client makes the HTTP requests (PUT or GET).
5. The request goes to the controller, and the controller maps the request and handles it. After that, it calls the service logic if required.
6. In the service layer, all the business logic performs, it performs the logic on the data that is mapped to JPA with model classes.
7. A JSP page is returned to the user if no error occurred.
8. **Annotations**
9. ***@EnableAutoConfiguration***

This annotation is usually placed on the main application class. The @EnableAutoConfiguration annotation implicity defines a base “search package”. This annotation tells Spring Boot to start adding beans based on classpath setting, other beans, and various property settings.

1. ***@SpringBootApplication***

This annotation is used on the application class while setting up a Spring Boot project. The class that is annotated with the @SpringBootApplication must be kept in the base package. The @SpringBootApplication is a convenient. The @SpringBootApplication that adds all the following:

1. *@Configuration*
2. *@EnableAutoConfiguration*
3. *@ComponentScan*
4. **Dependency Management**

Spring boot manages dependencies and configuration automatically. Each release of Spring Boot provides a list of dependencies that it supports.

1. Advantages
2. It provides the centralization of dependency information by specifying the Spring Boot version in one place. It helps when we switch from one version to another.
3. It avoids mismatch of different versions of Spring Boot libraries.
4. We only need to write a library name with specifying the version. It is helpful in multi module projects.
5. Maven Dependency Management System

The maven project inherits the following features from spring-boot-starter-parent:

1. The default java compiler version
2. UTF-8 source encoding
3. It inherits a **Dependency Section** from spring-boot-dependency-pom.
4. Dependencies, inherited from the spring-boot-dependency-pom
5. Sensible **resource** **filtering**
6. Sensible **plugin** **configuration**
7. Inheriting starter parent

The following **spring-boot-starter-parent** inherits automatically when we configure the project.

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.2.2.BUILD-SNAPSHOT</version>

<relativePath/>

</parent>

1. Changing java version

We can also change the Java version by using <java.version> tag.

<properties>

<java.version>1.8</java.version>

</properties>

1. Adding Spring BootMaven Plugin

We can also add Maven plugin in our pom.xml file. It wraps the project into an executable **jar** file.

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

1. Spring Boot without parent POM

If we don’t want to use **spring-boot-starter-parent** dependency, but still want to take the advantage of the dependency management, we can use <**scope**> tag, as follows:

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.2.2.RELEASE</version>

<type>pom</type>

<scope>**import**</scope>

</dependency>

</dependencies>

</dependencyManagement>

1. **Application Properties**

Spring boot framework comes with a built in mechanism for application configuration using a file called **application.properties**. It is located inside the **src/main/resources** folder, as shown in the following figure. The application.properties file allows us to run an application in a **different environment**. In short, we can use the application properties file to:

1. Configure Spring Boot framework
2. Define our application custom configuration properties
3. YAML Properties File

Spring boot provides another file to configure the properties is called **yml** file. The Yaml file works because the **Snake YAML** jar is present in the classpath. Example of application .yml:

spring:

application:

name: demoApplication

server:

port: 8081

1. Spring Boot Property Categories

There are **sixteen** categories of Spring Boot property are as follows:

1. Core Properties
2. Cache Properties
3. Mail Properties
4. JSON Properties
5. Data Properties
6. Transaction Properties
7. Data Migration Properties
8. Integration Properties
9. Web Properties
10. Templating Properties
11. Server Properties
12. Security Properties
13. RSocket Properties
14. Actuator Properties
15. DevTools Properties
16. Testing Properties
17. Application Properties Table

The following tables provide a list of common Spring Boot properties:

|  |  |  |
| --- | --- | --- |
| **Property** | **Default** | **Description** |
| Debug | False | It enables debug logs |
| spring.application.name |  | It is used to set the application name |
| spring.application.admin.enabled | False | It is used to enable admin features of the application |
| spring.config.name | Application | It is used to set config file name |
| spring.config.location |  | It is used to config the file name |
| server.port | 8080 | Configure the HTTP server port. |
| server.servlet.context-path |  | It configures the context path of the application |
| logging.file.path |  | It configures the location of the log file |
| spring.banner.charset | UTF-8 | Banner file encoding |
| spring.banner.location | classpath:banner.txt | It is used to set banner file loc |
| logging.file |  | It is used to set log file name. for example, data.log. |
| spring.application.index |  | It is used to set application index |
| spring.mail.default-encoding | UTF-8 | It is used to set default MimeMessage encoding. |
| spring.mail.host |  | It is used to set SMTP server host. For example, smtp.example.com |
| spring.mail.password |  | It is used to set login password of the SMTP server |
| spring.mail.port |  | It is used to set SMTP server port. |
| spring.mail.test-connection | False | It is used to test that the mail server is available on startup. |
| spring.mail.username |  | It is used to set login username of the SMTP server |
| spring.main.sources |  | It is used to set sources for the application. |
| server.address |  | It is used to set network address to which the server should bind to. |
| server.connection-timeout |  | It is used to set time in milliseconds that connectors will wait for another HTTP request before closing the connection. |
| server.context-path |  | It is used to set context path of the application. |
| server.server-header |  | It is used for the Server response header (no header is sent if empty) |
| server.server-path | / | It is used to set path of the main dispatcher servlet |
| server.ssl.enabled |  | It is used to enable SSL support |
| spring.http.multipart.enabled | True | It is used to enable support of multi-part uploads |
| spring.servlet.multipart.max-file-size | 1MB | It is used to set max file size |
| spring.mvc.async.request-timeout |  | It is used to set time in milliseconds. |
| spring.mvc-date-format |  | It is used to set date format, for example dd/MM/yyyy |
| spring.mvc.locale |  | It is used to set locale for the application |
| spring.social.facebook.app-id |  | It is used to set application Facebook App Id. |
| spring.social.linkedin.app-id |  | It is used to set application LinkedIn App Id. |
| spring.social.twitter.app-id |  | It is used to set application Twitter App Id. |
| security.basic.authorize-mode | role | It is used to set security authorize mode to apply |
| security.basic.enabled | true | It is used to enable basic authentication |
| spring.test.database.replace | any | Type of existing DataSource to replace. |
| spring.test.mockmvc.print | default | MVC print option |
| spring.freemaker.content-type | text/html | Content type value |
| server.server-header |  | Value to use for the server response header |
| spring.security.filter.dispatcher-type | async,error,request | Security filter chain dispatcher types. |
| spring.security.filter.order | -100 | Security filter chain order |
| spring.security.oauth2.client.registration.\* |  | OAuth client registration |
| spring.security.oauth2.client.provider.\* |  | OAuth client provider |

1. **Starters**

**Spring boot** provides a number of starters that allow us to add jars in the classpath. Spring boot built-in **starters** make development easier and rapid. **Spring Boot Starters** are the **dependency descriptors**.

1. **Starter Parent**
2. **Starter Web**
3. **Starter Data JPA**
4. **Starter Actuator**
5. **Starter Test**
6. **DevTools**
7. **Multi Module Project**
8. **Packaging**
9. **Auto configuration**
10. **Micro Services**

Micro service is an architecture that allows the developers to develop and deploy services independently. Each service running has its own process and this achieves the lightweight model to support business applications. Micro services offers the following advantages to its developers:

* Easy development
* Simple scalability
* Compatible with containers
* Minimum configuration
* Lesser production time