Spring Boot is an open source Java-based framework used to create a micro Service. It is developed by Pivotal Team and is used to build stand-alone and production ready spring applications.

What is Micro Service?

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.

Advantages

- Easy deployment
- Simple scalability
- Compatible with Containers
- Minimum configuration
- Lesser production time

What is Spring Boot?

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup.

Advantages

- Easy to understand and develop spring applications
- Increases productivity
- Reduces the development time

Why Spring Boot?

You can choose Spring Boot because of the features and benefits it offers as given here –

It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.

It provides a powerful batch processing and manages REST endpoints.

In Spring Boot, everything is auto configured; no manual configurations are needed.

It offers annotation-based spring application

Eases dependency management

It includes Embedded Servlet Container

Creating a Project

1) Spring Initializr

Spring Initializr is a web-based tool provided by the Pivotal Web Service. With the help of Spring Initializr, we can easily generate the structure of the Spring Boot Project. It offers extensible API for creating JVM-based projects.

It also provides various options for the project that are expressed in a metadata model. The metadata model allows us to configure the list of dependencies supported by JVM and platform versions, etc. It serves its metadata in a well-known that provides necessary assistance to third-party clients.

Spring Initializr Modules

initializr-actuator: It provides additional information and statistics on project generation. It is an optional module.

initializr-bom: In this module, BOM stands for Bill Of Materials. In Spring Boot, BOM is a special kind of POM that is used to control the versions of a project's dependencies. It provides a central place to define and update those versions. It provides flexibility to add a dependency in our module without worrying about the versions.

Outside the software world, the BOM is a list of parts, items, assemblies, and other materials required to create products. It explains what, how, and where to collect required materials.

initializr-docs: It provides documentation.

initializr-generator: It is a core project generation library.

initializr-generator-spring:

initializr-generator-test: It provides a test infrastructure for project generation.

initializr-metadata: It provides metadata infrastructure for various aspects of the projects.

initializr-service-example: It provides custom instances.

initializr-version-resolver: It is an optional module to extract version numbers from an arbitrary POM.

initializr-web: It provides web endpoints for third party clients.

Supported Interface

It supports **IDE STS, IntelliJ IDEA Ultimate, NetBeans, Eclipse**. You can download the plugin from https://github.com/AlexFalappa/nb-springboot. If you are using VSCode, download the plugin from https://github.com/microsoft/vscode-spring-initializr.

Use Custom Web UI http://start.spring.io or https://start-scs.cfapps.io.

It also supports the command-line with the **Spring Boot CLI** or **cURL or HTTPie**.

Generating a Project

Before creating a project, we must be friendly with UI. Spring Initializr UI has the following labels:

Project: It defines the kind of project. We can create either Maven Project or Gradle Project. We will create a Maven Project throughout the tutorial.

Language: Spring Initializr provides the choice among three languages Java, Kotlin, and Groovy. Java is by default selected.

Spring Boot: We can select the Spring Boot version.

Project Metadata: It contains information related to the project, such as Group, Artifact, etc. Group denotes the package name; Artifact denotes the Application name. The default Group name is com.example, and the default Artifact name is demo.

Dependencies: Dependencies are the collection of artifacts that we can add to our project.

There is another Options section that contains the following fields:

Name: It is the same as Artifact.

Description: In the description field, we can write a description of the project.

Package Name: It is also similar to the Group name.

Packaging: We can select the packing of the project. We can choose either Jar or War.

Java: We can select the JVM version which we want to use. We will use Java 8 version throughout the tutorial.

There is a **Generate** button. When we click on the button, it starts packing the project and downloads the **Jar** or **War** file, which you have selected.

Spring Tool Suite (STS) IDE

Spring Tool Suite is an IDE to develop Spring applications. It is an Eclipse-based development environment. It provides a ready-to-use environment to implement, run, deploy, and debug the application. It validates our application and provides quick fixes for the applications.

Download Spring Tool Suite from https://spring.io/tools3/sts/all.

Creating a Spring Boot Project Using STS

Example

Spring Boot Annotations

Spring Boot Annotations is a form of metadata that provides data about a program. In other words, annotations are used to provide supplemental information about a program. It is not a part of the application that we develop. It does not have a direct effect on the operation of the code they annotate. It does not change the action of the compiled program.

Core Spring Framework Annotations

@Required: It applies to the bean setter method.

```
public class Settet
{
private Integer cost;
@Required
public void setCost(Integer cost)
{  this.cost = cost;  }
public Integer getCost()
{  return cost;  }
}
```

@Autowired: Spring provides annotation-based auto-wiring by providing @Autowired annotation. It is used to autowire spring bean on setter methods, instance variable, and constructor. When we use @Autowired annotation, the spring container auto-wires the bean by matching data-type.

Example

```
@Component
public class Customer
{
private Person person;
@Autowired
public Customer(Person person)
{
this.person=person;
}
```

}

@Configuration: It is a class-level annotation. The class annotated with @Configuration used by Spring Containers as a source of bean definitions.

Example

```
@Configuration
public class Vehicle
{
@BeanVehicle engine()
{
return new Vehicle();
}
}
```

@ComponentScan: It is used when we want to scan a package for beans. It is used with the annotation @Configuration. We can also specify the base packages to scan for Spring Components.

Example

```
@ComponentScan(basePackages = "com.sys")
@Configuration
public class ScanComponent
{
// ...
}
```

@Bean: It is a method-level annotation. It is an alternative of XML <bean> tag. It tells the method to produce a bean to be managed by Spring Container.

Example

```
@Bean
public BeanExample beanExample()
{
return new BeanExample ();
}
```

Spring Framework Stereotype Annotations

@Component: It is a class-level annotation. It is used to mark a Java class as a bean. A Java class annotated with @Component is found during the classpath. The Spring Framework pick it up and configure it in the application context as a Spring Bean.

Example

```
@Component
public class Student
{ ..... }
```

@Controller: The @Controller is a class-level annotation.It is a specialization of @Component. It marks a class as a web request handler. It is often used to serve web pages. By default, it returns a string that indicates which route to redirect. It is mostly used with **@RequestMapping annotation.**

```
@Controller
@RequestMapping("books")
public class BooksController
{
@RequestMapping(value = "/{name}", method = RequestMethod.GET)
public Employee getBooksByName()
{
return booksTemplate;
}
}
```

@Service: It is also used at class level. It tells the Spring that class contains the business logic.

```
package com.javatpoint;
@Service
public class TestService
{
public void service1()
{
//business code
}
}
```

@Repository: It is a class-level annotation. The repository is a DAOs (Data Access Object) that access the database directly. The repository does all the operations related to the database.

```
package com.sys;
@Repository
public class TestRepository
{  public void delete()
{
//persistence code
} }
```

Spring Boot Annotations

@EnableAutoConfiguration: It auto-configures the bean that is present in the classpath and configures it to run the methods. The use of this annotation is reduced in Spring Boot 1.2.0 release because developers provided an alternative of the annotation, i.e. **@SpringBootApplication.**

@SpringBootApplication: It is a combination of three annotations **@EnableAutoConfiguration, @ComponentScan**, and **@Configuration**.

Spring MVC and REST Annotations

@RequestMapping: It is used to map the web requests. It has many optional elements like consumes, header, method, name, params, path, produces, and value. We use it with the class as well as the method.

@GetMapping: It maps the HTTP GET requests on the specific handler method. It is used to create a web service endpoint that fetches It is used instead of using: **@RequestMapping(method = RequestMethod.GET)**

@PostMapping: It maps the HTTP POST requests on the specific handler method. It is used to create a web service endpoint that creates It is used instead of using: **@RequestMapping(method = RequestMethod.POST)**

@PutMapping: It maps the HTTP PUT requests on the specific handler method. It is used to create a web service endpoint that creates or updates It is used instead of using: **@RequestMapping(method = RequestMethod.PUT)**

@DeleteMapping: It maps the HTTP DELETE requests on the specific handler method. It is used to create a web service endpoint that deletes a resource. It is used instead of using: **@RequestMapping(method = RequestMethod.DELETE)**

@PatchMapping: It maps the HTTP PATCH requests on the specific handler method. It is used instead of using: **@RequestMapping(method = RequestMethod.PATCH)**

@RequestBody: It is used to bind HTTP request with an object in a method parameter. Internally it uses HTTP MessageConverters to convert the body of the request. When we annotate a method parameter with **@RequestBody**, the Spring framework binds the incoming HTTP request body to that parameter.

@ResponseBody: It binds the method return value to the response body. It tells the Spring Boot Framework to serialize a return an object into JSON and XML format.

@PathVariable: It is used to extract the values from the URI. It is most suitable for the RESTful web service, where the URL contains a path variable. We can define multiple **@PathVariable** in a method.

@RequestParam: It is used to extract the query parameters form the URL. It is also known as a query parameter. It is most suitable for web applications. It can specify default values if the query parameter is not present in the URL.

@RequestHeader: It is used to get the details about the HTTP request headers. We use this annotation as a method parameter. The optional elements of the annotation are name, required, value, defaultValue. For each detail in the header, we should specify separate annotations. We can use it multiple time in a method

@RestController: It can be considered as a combination of **@Controller** and **@ResponseBody** annotations. The **@RestController** annotation is itself annotated with the **@ResponseBody** annotation. It eliminates the need for annotating each method with **@ResponseBody**.

@RequestAttribute: It binds a method parameter to request attribute. It provides convenient access to the request attributes from a controller method. With the help of @RequestAttribute annotation, we can access objects that are populated on the server-side.

Spring Boot Dependency Management

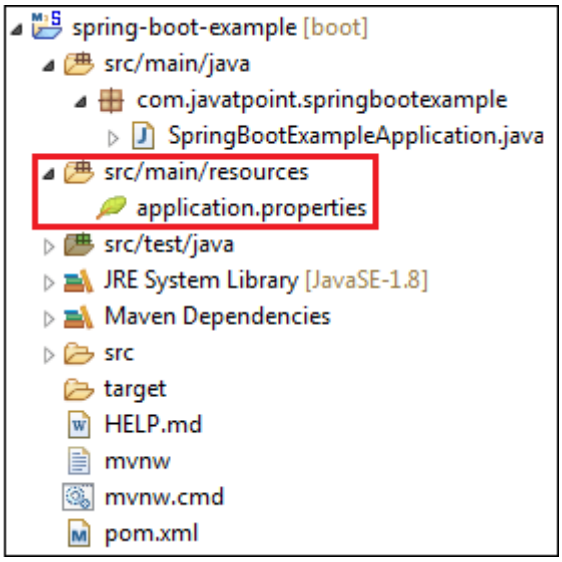
Spring Boot manages dependencies and configuration automatically. Each release of Spring Boot provides a list of dependencies that it supports. The list of dependencies is available as a part of the Bills of Materials (spring-boot-dependencies) that can be used with Maven. So, we need not to specify the version of the dependencies in our configuration. Spring Boot manages itself. Spring Boot upgrades all dependencies automatically in a consistent way when we update the Spring Boot version.

Advantages of Dependency Management

- 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.
- It avoids mismatch of different versions of Spring Boot libraries.
- We only need to write a library name with specifying the version. It is helpful in multi-module projects.

Spring Boot 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.



Spring Boot provides various properties that can be configured in the application.properties file. The properties have default values. We can set a property(s) for the Spring Boot application. Spring Boot also allows us to define our own property if required.

The application.properties file allows us to run an application in a different environment. In short, we can use the application.properties file to:

Configure the Spring Boot framework

define our application custom configuration properties

Example of application.properties

#configuring application name

spring.application.name = demoApplication

```
#configuring port
server.port = 8081
```

In the above example, we have configured the application name and port. The port 8081 denotes that the application runs on port 8081.

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. Instead of using the application.properties file, we can also use the application.yml file, but the Yml file should be present in the classpath.

Example of application.yml

```
spring:
application:
name: demoApplication
server:
port: 8081
```

In the above example, we have configured the application name and port. The port 8081 denotes that the application runs on port 8081.

Spring Boot Property Categories

There are sixteen categories of Spring Boot Property are as follows:

- ✓ Core Properties
- ✓ Cache Properties
- ✓ Mail Properties
- ✓ JSON Properties
- ✓ Data Properties
- ✓ Transaction Properties
- ✓ Data Migration Properties
- ✓ Integration Properties
- ✓ Web Properties
- ✓ Templating Properties
- ✓ Server Properties
- ✓ Security Properties
- ✓ RSocket Properties
- ✓ Actuator Properties
- ✓ DevTools Properties
- ✓ Testing Properties

Application Properties Table

Property	Default Values	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	Configures 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 location.
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.application.name		It is used to set the application name.
spring.application.admin.enabled	false	It is used to enable admin features for the application.
spring.config.location		It is used to config the file locations.
spring.config.name	application	It is used to set config the file name.

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 user 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.port	8080	It is used to set HTTP port.
server.server-header		It is used for the Server response header (no header is sent if empty)
server.servlet-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's Facebook App ID.
spring.social.linkedin.app-id		It is used to set application's LinkedIn App ID.
spring.social.twitter.app-id		It is used to set application's 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 registrations.
spring.security.oauth2.client.provider.*		OAuth provider details.

Spring Boot 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.

In the Spring Boot Framework, all the starters follow a similar naming pattern: spring-boot-starter-*, where * denotes a particular type of application. For example, if we want to use Spring and JPA for database access, we need to include the spring-boot-starter-data-jpa dependency in our pom.xml file of the project.

Third-Party Starters

We can also include third party starters in our project. But we do not use spring-boot-starter for including third party dependency. The spring-boot-starter is reserved for official Spring Boot artifacts. The third-party starter starts with the name of the project. For example, the third-party project name is abc, then the dependency name will be abc-spring-boot-starter.

Spring Boot Actuator

Spring Boot Actuator is a sub-project of the Spring Boot Framework. It includes a number of additional features that help us to monitor and manage the Spring Boot application. It contains the actuator endpoints (the place where the resources live). We can use HTTP and JMX endpoints to manage and monitor the Spring Boot application. If we want to get production-ready features in an application, we should use the Spring Boot actuator.

Spring Boot Actuator Features

- ✓ Endpoints
- ✓ Metrics
- ✓ Audit

Endpoint: The actuator endpoints allows us to monitor and interact with the application. Spring Boot provides a number of built-in endpoints. We can also create our own endpoint. We can enable and disable each endpoint individually. Most of the application choose HTTP, where the Id of the endpoint, along with the prefix of /actuator, is mapped to a URL.

For example, the /health endpoint provides the basic health information of an application. The actuator, by default, mapped it to /actuator/health.

Metrics: Spring Boot Actuator provides dimensional metrics by integrating with the micrometer. The micrometer is integrated into Spring Boot. It is the instrumentation library powering the delivery of application metrics from Spring. It provides vendor-neutral interfaces for timers, gauges, counters, distribution summaries, and long task timers with a dimensional data model.

Audit: Spring Boot provides a flexible audit framework that publishes events to an AuditEventRepository. It automatically publishes the authentication events if spring-security is in execution.

Spring Boot Data Base

Sping boot JPA