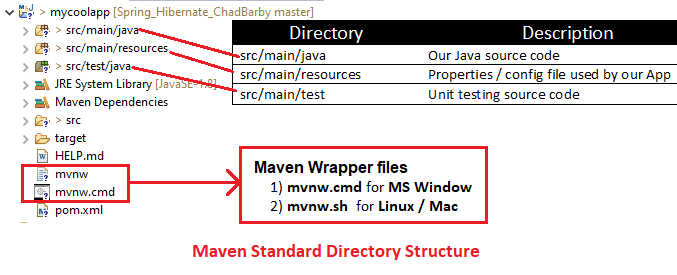
**Exploring the Spring Boot Project Structure**

**Spring Initializr**:

Spring Initializr created a Maven project for us. Now we explore the project structure.

**Maven Standard Directory Structure**:



|  |  |
| --- | --- |
| Directory | Description |
| src/main/java | Our Java source code |
| src/main/resources | Properties / config file used by our App |
| src/main/test | Unit testing source code |

**Maven Wrapper files**:

1. **mvnw** allows us to run a Maven project
   * No need to have Maven installed or present on our path
   * If correct version of Maven is NOT found in our computer
     + Automatically downloads correct version and runs Maven
2. Two files are provided
   * **mvnw.cmd** for **MS Windows**
   * **mvnw.sh** for **Linux/Mac**
3. If we already have Maven installed previously
   * Then we can ignore/delete the **mvnw** files
   * Just use Maven as normally
     + **mvn clean compile test**

**Maven POM file (pom.xml)**:

"**pom.xml**" includes info that we entered at Spring Initializr website

**pom.xml**:

<?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*

*https://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>2.1.8.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>com.ruhul.springboot.demo</groupId>

<artifactId>mycoolapp</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>mycoolapp</name>

<description>Demo project for Spring Boot</description>

<properties>

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

<maven-jar-plugin.version>3.1.1</maven-jar-plugin.version>

</properties>

<dependencies>

<!-- Spring Boot Starters -> A collection of Maven dependencies

(Compatible versions)

-->

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

1. Spring Boot Starters () A collection of Maven dependencies (Compatible versions)
2. "spring-boot-starter-web " collection of dependencies like
   1. Spring-web
   2. Spring-web-MVC
   3. Hibernate-validator
   4. Tomcat
   5. JSON
   6. etc.

This save's developer from having to list all of the individual dependencies. Also, makes sure we have compatible version.

**Spring Boot Maven plugin**:

Here in the "**pom.xml**" file we also have the "**spring-boot-maven-plugin**". Here this plugin is use to package an executable .jar or .war file that we can run from the command line.

* ./mvnw package
* ./mvnw spring-boot:run

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

If Maven is already installed in our computer then we can use the following command

* ./mvn package
* ./mvn spring-boot:run

**Java Source Code**:

Here in our "MycoolappApplication.java" class we have an imported class

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

This is a special annotation. This annotation basically enables Auto Configuration, Component Scanning and additional configuration of our application. This one annotation is actually composing of other annotations.

* @EnablesAutoConfiguration
* @ComponentScann
* @Configuration

**Annotations**:

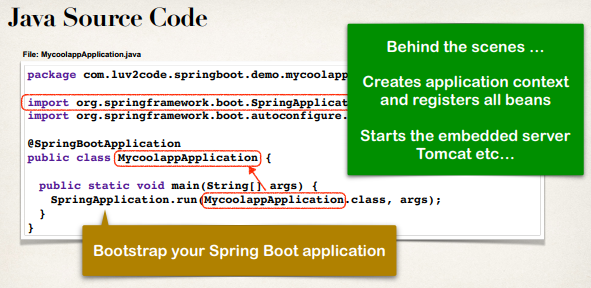
"**@SpringBootApplication**" is composed of the following annotations.

|  |  |
| --- | --- |
| Annotation | Description |
| @EnableAutoConfiguration | Enables Spring Boot's auto-configuration support |
| @ComponentScan | Enables component scanning of current package  Also recursively scans sub-packages |
| @Configuration | Able to register extra beans with @Bean  or import other configuration classes |

**SpringApplication.java class**:

In Spring boot, we have a special class name "**SpringApplication.class**". We use the class to Bootstrap our Spring boot application. Behind the scenes, Spring boot actually creates application context

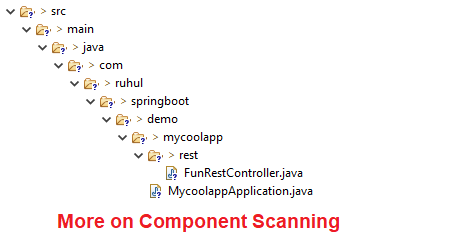
and registers all beans, all the component scanning and also started the embedded server like Tomcat, keep the server up and running and listening the given port that we can start accessing.



**More on Component Scanning**:

The base practice is to place our main application class in the root package above our other packages

* This implicitly defines a base search package
  + Allows us to leverage default component scanning
  + No need to explicitly reference the base package name



In our example here we have our main Spring boot application class "**MycoolappApplication.java**" and by using Spring boot it will automatically perform component scans sub-packages here. Includes "**rest**" sub-package any other sub-packages we create and give them any name.

* Default scanning is fine if everything is under
  + **com.ruhul.springboot.demo.mycoolapp**
* But what about the other packages?
  + **com.acmi.iot.utils**
  + **com.odduu.ruhul**

It does not follow the hole naming convention, that follow our main package

"**com.ruhul.springboot.demo.mycoolapp**". Here we have to explicitly list the base packages to scan.

**File: MycoolappApplication.class**:

**package** com.ruhul.springboot.demo.mycoolapp;

...

@SpringBootApplication(

scanBasePackages={" com.ruhul.springboot.demo.mycoolapp",

"com.acmi.iot.utils",

"com.odduu.ruhul"})

**public** **class** MycoolappApplication {

...

}

**Application Properties**:

By default, Spring Boot will load properties from: "**application.properties**". The application.properties file will be created by the Spring initializer when we create Spring boot application. This file is empty at the beginning. We can add Spring boot property here like

server.port = 9090

spring.application.name = demoservice

**Static Content**:

By default, Spring Boot will load static resources from "/static" directory. Some example of static resources is HTML files, CSS, JavaScript, images, etc. We simply place those in the static directory and Spring automatically lode those.

**Static Content**:

Do not use the "**src/main/webapp**" directory if our application is packaged as a JAR. Although this is a standard Maven directory, it works only with WAR packaging. It is silently ignored by most build tools if we generate a JAR.

**Templates**:

Spring Boot includes auto-configuration for following template engines.

* Thymeleaf
* FreeMarker
* Mustache

By default, Spring Boot will load templates from "/templates" directory.

**Unit Tests**:

Exploring the Spring Boot Project Structure