Spring boot

Gets hands-on with spring boot

* Why spring boot?
* Terminologies

1. Spring initializer
2. Auto configuration
3. Starter project
4. Actuator
5. Developer Tools

Lets create new spring-boot project

1. start.spring.io (keep this URL in browser)
2. Select

Project – maven project

Language – java

Spring boot – 2.6.1(select latest version should not have snapshot and m1/m2/m3)

Project metadata

Group—give package name (com.sau)

Project—give class name (learn-spring-boot)

Below this all by default as given

1. On right side -> add dependency

Search web -> select spring web -> (below) click on generate

1. Download -> extract -> copy somewhere
2. In Eclipse -> file -> Import ->(type there)existing maven -> next -> browse ->folder(where spring file has extracted) -> pom.xml(select) ->finish

Build a hello world REST-API in modern spring boot approach

url for the web app: http://localhost:8080/courses

src/main/java

* Create a controller class

Package name: com.sau.learnspringboot.courses.controller

Class Name: CourseController

package com.sau.learnspringboot.courses.controller;

import java.util.List;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

import com.sau.learnspringboot.courses.bean.Courses;

@RestController

public class CourseController {

@GetMapping("/courses")

public List<Courses> getAllCourses(){

return List.of(new Courses(10,"sau the beautiful girl","sau"));

}

}

* Create a class

Package name: com.sau.learnspringboot.courses.bean

Class Name: Courses

**package** com.sau.learnspringboot.courses.bean;

**public** **class** Courses {

**private** **long** id;

**private** String name;

**private** String author;

**public** Courses(**long** id, String name, String author) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.author = author; }

**public** **long** getId() {

**return** id;}

**public** String getName() {

**return** name;}

**public** String getAuthor() {

**return** author;

}

@Override

**public** String toString() {

**return** "Courses [id=" + id + ", name=" + name + ", author=" + author + "]";

}}

Webpage output

[{"id":10,"name":"sau the beautiful girl","author":"sau"}]

Understanding the world before spring boot

Setting up Spring web projects before spring boot was not easy

* Define maven dependencies and manage versions for frameworks

Spring-webmvc, Jackson-databind, log4j etc

* Define web.xml(src/main.webapp/WEB-INF/web.xml)

Define front controller for spring framework(DispatcherServlet)

* Define a spring context XML file(/src/main/webapp/WEB-INF/todo-servlet.xml)

Define a component Scan(<context:component-Scan base-package=”com.sau” />)

* Install tomcat or use tomcat7-maven-plugin plugin(or any other web server)
* Deploy and run the application in Tomcat

How does spring boot do its magic?

* Spring boot starter projects
* Spring boot auto configuration

Spring boot starter projects

1. Goal of starter projects: help you get a project up and running quickly

* Web application- spring boot starter web
* REST API- spring boot starter web
* Talk to database using JPA- spring boot starter web
* Talk to database using JDBC - spring boot starter JDBC
* Secure your web application or REST API-spring boot starter security

1. Manage list of maven dependencies and versions for different kinds of apps:
   * Spring boot starter Web: Frameworks needed by typical web applications

Spring-webmvc, spring-web, spring-boot-starter-tomcat, spring-boot-starter-json

Auto configuration

Spring boot provides Auto configuration

Basic configurations to run your application using the frameworks defined in maven dependencies

Auto-configuration is decided based on:

* Which frameworks are in class path?
* What is the existing configuration(Annotations etc)?

An example:

Enable debug logging for more details(src/main/resources -> application.properties)

If we use spring-boot-starter-web, following are auto configured

* Dispatcher Servlet(DispatcherServletAutoConfiguration)
* Embedded servlet Container – Tomcat is the default

(EmbeddedWebServerFactoryCustomizerAutoConfiguration)

* Default error pages(ErrorMvcAutoConfiguration)
* Bean to/from JSON conversion

(JacksonHttpMessageConvertersConfiguration)

Spring boot embedded servers

How do we deploy our application before spring boot embedded servers?

1)Install java

2)install web/application server

Tomcat/WebSphere/webLogic etc

3)Deploy the application WAR (Web Archive)

* This is the old war approach
* Complex to setup

After Embedded server (it is a simpler alternative)

1:install java

2:run JAR file

Embedded server examples:

Spring-boot-starter-tomcat

Spring-boot-starter-jetty

Spring-boot-starter-undertow

Create JAR file

Right click on learn-spring-boot(project)

Run as->4 maven build

Goal name -> clean install

Spring boot Actuator

* Monitor and manage your application in your production
* Provides a number of endpoints
* Beans - Compile list of spring beans in your app
* Health - application health information
* Metrics – application metrics
* Mappings – details around request mappings
* Add dependency in pom.xml

<dependency>

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

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

</dependency>

* application.properties
* management.endpoints.web.exposure.include=\*
* management.endpoints.web.exposure.include=health,metrics
* in chrome type url to get info about health/metric these kinds of parameter