application.properties

* By default, SB loads properties from this file.
* Created by spring initializer
* Empty at the beginning
* server.port=6565 (we can add props in this file)
* To add custom properties

coach.name=Prasad

1. In application.properties

# Customizing the property  
coach.name=Prasad

1. Create a DemoController Class

package com.flynaut.SecondaryApp.rest;  
  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class DemoController {  
  
 @Value("${coach.name}")  
 private String coachName;  
  
 //Defining method which will handle get request at coach  
 @GetMapping("/coach")  
 public String sayCoach(){  
 return coachName;  
 }  
}

* Static directory
* static content
* by default, SB loads static resources from “/static” directory
* Ex. HTML files, images, CSS, JS

Spring Boot Starters:

A collection of maven dependencies with compatible versions

spring-boot-starter-web

* Spring-web
* Spring-webmvc
* Tomcat
* Json

Building a spring application is hard

Why is it hard?

* It will be great if there is a list of maven dependencies

Collected as a group of dependencies. – One stop solution

* If we have one stop solution, then we don’t have to search for each dependency.

Solution – SB starters

* A curated list of maven dependencies
* A collection of dependencies grouped together
* It reduces the configuration part.
* If we are building a Spring app that needs- web, security

We will simply select dependency in SI

It will add all the appropriate dependencies in pom.xml grouped together in SBS.

1. spring-boot-starter-web

* building web apps, Tomcat server as embedded default server

1. spring-boot-starter-security

* Adds the security support

1. spring-boot-starter-jpa

* Gives database support with JPA & Hibernate

SpringBoot Dev Tools

The Problem:

* When running the SB app
* If we are making any changes in the source code
* Then manually we have to restart the application ☹

Solution: SpringBoot Dev Tools ☺

* Automatically restarts our app when we make changes in our code.

How?

* Simply add the dependency in pom.xml

Step 1: Add the dependency in pom.xml

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

Step 2:

File -> Settings -> build,execution,deployment -> compiler -> check the box (build project automatically)-> apply -> ok

Step 3:

Settings/Preferences -> Advanced settings -> check Allow automake ……

SpringBoot Actuators

The Problem:

How can I manage and monitor my application?

How can I check the health of my application?

The Solution: SB Actuator

* Exposes endpoints to monitor and manage our app.
* Endpoints are automatically added in our application.
* We don’t need to write code for this.
* Dependency

<dependency>

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

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

</dependency>

* All endpoints will be available and prefixed with : /actuator
* /health -> To get health information about our application

URL : localhost:8080/actuator/health

* /info endpoint can provide the information about our app.

To expose the endpoint

Empty at beginning.

We need to make changes in our application.properties

management.endpoints.web.exposure.include=info,health

/health endpoint is exposed, by default

To expose all the actuator endpoints we can use wildcard ‘\*’ .

1. /actuator/threaddump

* List all the threads running in our applications
* Useful for analyzing the performance of our application

1. /actuator/mappings

* List all the request mappings for our application
* Useful for finding out what request mappings are available

1. /auditevents
2. /beans

What about Security?

1. ADD DEPENDENCY IN POM.XML

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

#We can override default username and password  
spring.security.user.name=Prasad  
spring.security.user.password=1234

Running our application without IDE:

1. Check java installed in your system or not.
2. Package our app with

* mvnw package

java –jar {name of jar file}

SprinBoot Properties:

* Security
* Web

~ To change the context path

server.servlet.context-path=/Prasad

* Data
* Core
* Testing

Inversion of Control

* Outsourcing the construction and management of objects
* It is a design principle where the control of object creation is given to a framework or container.
* Instead of creating and managing objects manually we are going to give it to spring.
* SpringContainer
* Create and Manage (Inversion of Control)
* Inject object dependency (Dependency Injection)
* How do we configure the SpringContainer?
* XML configuration file(LEGACY)
* Java Annotations (MODERN)

Spring Dependency Injection:

Types of dependency Injection:

1. Constructor Injection
2. Setter Injection
3. Field Injection(using annotations)

Constructor Injection:

In CI, dependencies are passed through the constructor when an object is instantiated.

Class Engine{

void start(){

sout(“Engine Started”);

}

}

Class Car{

Private Engine engine;

// Injecting the dependency via Constructor

Public Car(Engine engine){

this.engine=engine;

}

Void drive(){

engine.start();

}

}

//creating objects and injecting dependencies manually

Public class Main{

Public static void main(String[] args){

Engine engine = new Engine(); //Creating dependency

Car car = new Car(engine); //injecting the dependency via constructor

car.drive();

}

}

When to use constructor injection?

* Use it when we have all the required dependencies

When to use Setter injection?

* Use it when we have optional dependencies

What is Spring Autowiring?