SpringBoot

* Purpose: To build java application

Try to type code with me.

Pre-requisites:

OOP, classes, interfaces, inheritance, exception handling, collection framework

Must have installed:

JDK -> JDK 17 or higher to use springboot 3

IntelliJ IDE

* Provides large number of helper classes and annotations

The Problem with spring:

Traditional spring application building was tedious

Qs

1.Which JAR dependencies do I need?

2.How do I set up configuration? (xml or java)

3.How do I install the server? (Tomcat, JBoss etc)

& this is just getting started

* SpringBoot is the Solution for this
* Easier for spring development
* Minimize manual configuration (It performs the auto-configuration)
* Resolve dependency conflicts
* Provide an embedded HTTP server
* SpringBoot and Spring
* SpringBoot uses Spring Behind the scenes.
* SpringBoot simply makes it easier to use spring.
* Spring Initializer (SpringBoot provides it)

<http://start.spring.io>

* Quickly create a starter spring project
* Select dependencies
* Select maven/gradle
* Import project in IDE
* SB Embedded Server

Provide embedded server

* Tomcat, JBoss, Undertow

No need to install server separately

firstapp.jar

Jar file includes our app code & include server

Self-contained unit

Mycode

Tomcat

FAQs

1. Does SB replace Spring MVC, Spring REST..?

No, Instead it uses these technologies

1. Does SB run code faster than regular Spring Code?

No, SB uses same code of spring framework

Maven:

* When building our project, we may need additional JAR files

Ex. Spring, Hibernate, JSON etc

1st Approach:

Download the JAR files from each project website

Manually add the JAR files to our build path/classpath

Maven is Solution

* Tell maven the projects we are working with(dependencies)
* Maven will go out and download the JAR files for those projects
* And Maven will make those JAR files available during compile/run
* We can say maven is our helper or personal shopper (shopping list)

Development Process:

1. Configure our project at spring initializer(dependency: Spring Web)
2. Download zip file
3. Unzip the file
4. Import the project into our IDE

Lets Create RestController

package com.flynaut.springboot.demo.firstapp.rest;  
  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.time.LocalDate;  
  
@RestController  
public class FunRestController {  
  
 //This method will handle GET request at "hello" endpoint  
 @GetMapping("/hello")  
 public String sayHello(){  
 return "Hello Team!!!!!";  
 }  
  
 @GetMapping("/date")  
 public LocalDate date(){  
 LocalDate localdate= LocalDate.*now*();  
 return localdate;  
 }  
  
}

URL: Uniform Resource Locator

<http://localhost:8080>

<http://www.abc.com:8080/banking>

http: Application Layer Protocol(http : hypertext transfer protocol)

[www.abc.com](http://www.abc.com) : DNS qualified host name/IP address(to resolve the host problem)

8080: TCP port (to identify the port)

/banking: path or URI (Uniform resource identifier)

Maven

* What?
* A project management tool(build tool)
* Most popular use of Maven is for build management and dependencies
* What problems does maven solve?

1st Approach – Without using maven

firstApp

Spring JAR files

Spring

Hibernate

JSON

Hibernate JAR files

Developer

JSON JAR files

2st Approach – With using maven

* Tell maven the projects we are working with (dependencies)
* Go out and download Jar Files for us

Maven

Developer

Spring

Hibernate

JSON

* Maven Project Structure

Maven follows standard directory structure.

* Normally when we join a new project
* Every development team used to make their own project directory
* And this is not ideal for new comers and not standardized

|  |  |
| --- | --- |
| Directory | Description |
| src/main/java | Our java source code |
| src/main/resources | Properties/config files used by our app |
| src/test | Unit testing code and properties |
| target | Destination directory for compiled code(Automatically created by maven) |
| pom.xml | Maven configuration file |

POM.xml

Project Object Model file

* Configuration file for our project
* Basically our “Shopping List” for Maven ☺

Located at root of maven project

* POM file Structure

pom.xml

List of projects we depend on

Project name, version

Plug ins

Dependencies

Project meta data

* Project Coordinates
* Project coordinates uniquely identify a project
* Similar to GPS coordinates for home: latitude/ longitude
* Precise info for finding our home(city,street,house)

<groupId>com.flynaut.springboot.demo</groupId> (City)  
<artifactId>firstapp</artifactId> (Street)  
<version>0.0.1-SNAPSHOT</version> (Home No.)

GroupId: Name of Company, group, organization

Convention is to use reverse domain name: com.flynaut

ArtifactId: Name for our project: firstapp

Version: A specific release version like: 1.0,2.0

* Adding Dependencies
* To add dependency we need GAV

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

How to find the dependency coordinates?

1st Way – Visit the project page (spring.io, hibernate.org etc)

2nd Way – Visit [Maven Repository: Search/Browse/Explore](https://mvnrepository.com/)

Visit [Maven Central](https://central.sonatype.com/)

SB Project Structure

* Maven Wrapper Files

mvnw

mvnw.cmd

* mvnw allow us to run a maven project
* No need to have maven installed or present in our path

mvnw.cmd for windows

mvnw.sh for linux/mac

Maven POM file

Pom.xml includes info which we are entering in Spring Initializer

Spring Boot Starters –(A collection of Maven dependencies{compatible versions})

org.springframework.boot

spring-boot—starter-web -> spring-web

spring-webmvc

tomcat

json

Application Properties

By default, SB will load props from: application.properties

* Created by Spring Initializer
* Empty at the beginning

We can add SB props in this file

Server.port=7070

To add our own custom properties

coach.name=Prasad

@RestController  
public class FunRestController {  
 @Value("${coach.name}")  
 private String coachName;  
  
 //This method will handle GET request at "hello" endpoint  
 @GetMapping("/hello")  
 public String sayHello(){  
 return coachName;  
 }

}

In application.properties

spring.application.name=firstapp  
  
#Customizing the properties  
coach.name=Prasad

Task : add one more property in application.properties

Create restcontroller and return that String

* Static Content
* By default, SB will load static resources from “/static” directory
* Examples of static resources -> images, HTML files, CSS, JS
* Unit Tests

SpringBoot Unit Test Class

Created by Spring Initializer

We can add unit tests to the file

* Spring Boot Starters

Building a spring app is really hard

Why is it hard?

* It would be great if there is a list of maven dependencies
* Collected as group of dependencies… one-stop shop
* So we don’t have to search for each dependency

THERE SHOULD BE AN EASIER SOLUTION

* The Solution – SB Starter
* A curated list of maven dependencies
* A collection of dependencies grouped together
* Tested by SB team
* It makes much easier for the developer to get started with spring
* Reduces the amount of configuration part
* If we are building a Spring app that needs: web, security

Simply select the dependencies in the SI

It will add the appropriate SB starter to our pom.xml

|  |  |
| --- | --- |
| Name | Desc |
| spring-boot-starter-web | Building web apps, includes validation, REST  Uses Tomcat as default embeddedserver |
| spring-boot-starter-security | Adding spring boot security support |
| spring-boot-starter-jpa | Spring database support with JPA & Hibernate |

What is in the starter?

Select View -> Tool Windows -> maven -> dependencies

SpringBoot Dev Tools

The Problem:

* When running SB app
* If we make changes to our source code
* Then manually we have to restart the application ☹

Solution: SpringBoot Dev Tools

- Automatically restarts the our application when we update the code

* Simply add the dependency to our POM file

Step 1: add this dependency in your pom.xml

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

Step 2:

Settings/Preferences -> build,execution,deployment -> Compiler

Checkbox =build project automatically

Step 3:

Settings/Preferences -> Advanced Setting -> Allow auto-make……

SpringBoot Actuator

The Problem?

How can we monitor and manage my application?

How can I check the health of the application?

Solution: SB Actuator

* Exposes endpoints to monitor and manage our application
* REST endpoints are automatically added to our application

No need to write additional code

* Add dependency to our pom.xml file

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

* All endpoints will be prefixed with: /actuator

/health -> To get health information about our application

localhost:8080/actuator/health

* The /info endpoint can provide information about out application

To expose info:

We need to make changes in application.properties

What about Security?

DAY3

/actuator/threaddump

* Listing all the threads running in our application
* Useful for analyzing the performance of our web application

/actuator/mappings

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

Spring Security:

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

We can override the default username and password

spring.security.user.name=Prasad  
spring.security.user.password=1234

To exclude /health and /info

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

* Running the SB app from command line

1st -> Use java –jar <name of our project’s JAR file>

2nd -> Use SB maven plugin

mvnw spring-boot:run

java –jar firstApp.jar (Self contained unit)

Code

Tomcat

(firstApp.jar)

mvnw package – to generate the jar file of our project(It will be generated in target folder)

SpringBoot Properties

The properties are grouped into the following categories

* Core
* Web
* Security
* Data
* DevTools
* Testing
* Actuator

Web Properties

* Http server port

server.port=7878

* To change the context path of our application

server.servlet.context-path=/mypath

* Default HTTP session timeout

Server.servlet.session.timeout=15m (15 minutes)

Default timeout is 30 minutes

* Data properties

# login username of the database

spring.datasource.username=Prasad

# login password of the database

spring.datasource.password=root123

Inversion of Control

The approach of outsourcing the construction and management of objects.

Problem Coding Scenario

getDailyWorkout()

CricketCoach

MyApp

Application should be configurable

Spring Container

Give me “coach ” object

CricketCoach

Configurations

Object Factory

MyApp

FootballCoach

HockeyCoach

* Spring Container
* Primary Function

1. Create and Manage (Inversion of Control)
2. Inject object dependency (Dependency Injection)

* How do we configure Spring Container?
* XML configuration file(legacy)
* Java Annotations(modern)
* Spring Dependency Injection

The dependency inversion principle

The client delegates to another object

The responsibility of providing its dependencies

Returns a “Car” Object

Assemble car for us

Give me a “Car” Object

Car Factory

Coach

DemoController

* Demo Example
* Coach will provide daily workouts
* The DemoController wants to use the Coach

1. New Helper: Coach
2. This is a dependency

* Need to inject this dependency into the controller
* Injection Types
* There are many types of injection with spring
* Will cover only two recommended types

1. Constructor Injection
2. Setter Injection

When to use each?

* Constructor injection
* use it when you have all the required dependencies
* Setter Injection
* Use this when we have some optional dependencies

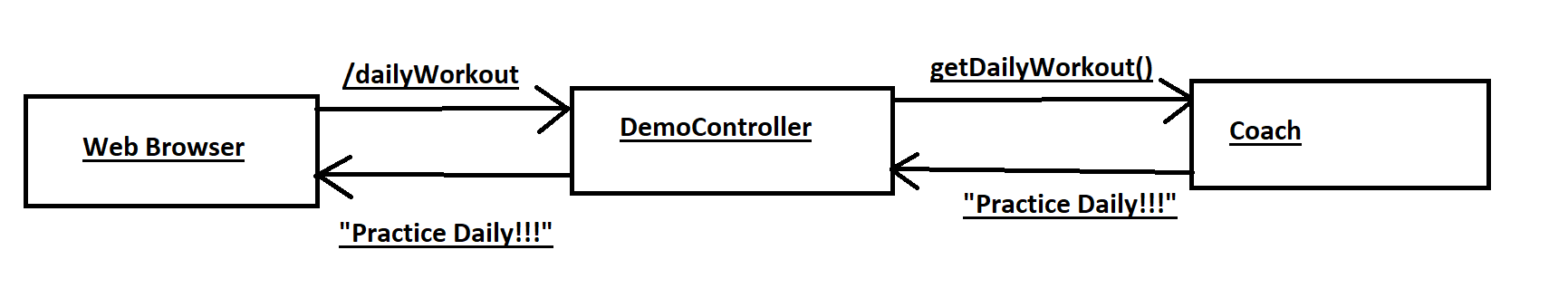
What is Spring Autowiring?

* For dependency injection, spring uses autowiring
* Autowiring example

Injecting a Coach interface

1. Spring will scan for @Components or a class which is annotated with @Components
2. Will ask does any one implements the coach interface??
3. If so, let’s inject them. For Ex. CricketCoach

Example Application



* Development Process for constructor injection

1. Define the dependency interface and class

package com.flynaut.injection.spring\_boot\_injection;  
  
public interface Coach {  
 String getDailyWorkout();  
}

package com.flynaut.injection.spring\_boot\_injection;  
  
import org.springframework.stereotype.Component;  
  
@Component  
public class CricketCoach implements Coach{  
 @Override  
 public String getDailyWorkout() {  
 return "Practice Practice and Practice";  
 }  
}

1. Create DemoController
2. Create a constructor in our class for injections

package com.flynaut.injection.spring\_boot\_injection;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class DemoController {  
  
 private Coach myCoach;  
  
 @Autowired  
 public DemoController(Coach theCoach){  
 myCoach=theCoach;  
 }  
  
 @GetMapping("/dailyWorkout")  
 public String getDailyWorkout(){  
 return myCoach.getDailyWorkout();  
 }  
}

1. Add @GetMapping for /dailyWorkout

@Component Annotation

* Marks the class as Spring Bean
* A spring bean is just a class that is managed by Spring
* Also makes the bean available for DI