**70. Spring Boot – Overview**

Spring boot solution:

* Make it easier to get started with Spring development
* Minimize the amount of manual configuration
  + Perform auto-configuration based on props files and JAR classpath
* Help to resolve dependency conflicts (Maven or Gradle)
* Provide an embedded HTTP server
  + Tomcat, Jetty, Undertow, …
  + Jar file includes your application code and includes the server
  + Apps can be run standalone
  + Run the Spring Boot app from the IDE or command-line

Spring Initializr (start.spring.io)

* Quickly create a starter Spring project
* Select your dependencies
* Creates a Maven / Gradle project
* Import the project into your IDE

Deploying Spring Boot Apps:

* Spring Boot apps can also be deployed in the traditional way
* Deploy WAR file to an external server: Tomcat, JBoss, WebSphere etc …

Note 1: Once you do Spring Boot configs then you make use of regular Spring coding

Spring Boot Demo development process:

1. Configure project at Spring initializr
2. Download the zip file
3. Unzip the file
4. Import Maven project into IDE

Maven Wrapper Files. mvnw allows you to run a Maven project. Two files provided:

* mvnw.cmd – for MS Windows
* mvnw.sh – Linux, Mac

If you already have Maven installed, you don’t need these files

pom.xml includes info that you entered in initializer. There is also springframework.boot plugin

Can also just use:

1. mvn package
2. mvn spring-boot:run

@SpringBootApplication annotation enables:

* Auto configuration (@EnableAutoConfiguration)
* Component scanning (@ComponentScan) of current package and subpackages
* Additional configuration (@Configuration) – able to register extra beans with @Bean or import other configuration classes

SpringApplication class is bootstrap your application:

SpringApplication.run(MycoolApplication.class, args)

Place your main application class in the root package above your other packages:

* This implicitly defines a base search package
  + Allows you to leverahe default component scanning
  + No need to reference package manually

You can explicitly list base packages to scan:

scanPackeages={“”,””}

By default, Spring boot will load properties from: application.properties. You can use this file in application using injection:

@Value(“${coach.name}”)

Private String coachName;

By default, Spring Boot will load static resources from “/static” directory.

Do not use source/main/webapp directory if your 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 you generate a JAR.

Spring Boot includes auto-configuration for following template engines:

* FreeMarker
* Thymleaf
* Mustache

Spring Boot Starters a curated list of Maven dependencies. A collection of dependencies grouped together. Tested and verified by the Spring development team. Makes it much easier for the developer to get started with Spring. Spring boot provides: spring-boot-starter-web. Contains:

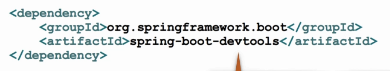
* Spring-web
* Spring-webmvc
* Hibernate-validator
* Json
* Tomcat and etc.

There are 30+ Spring Boot Starters from the Spring Development Team (luv2code.com/spring-boot-starters)

**71. Spring Boot - Spring Boot Dev Tools and Spring Boot Actuator**

spring-boot-devtools:

* Automatically restarts application when code is updated
* Simply add the dependency to POM file



Spring Boot Actuator exposes endpoints to monitor and manage application. Simply add the dependency to POM file. Rest endpoints are added automatically. Adding dependency:



Endpoints are prefixed with: /actuator:

* /health – health information:
  + Checks status of application
  + Normally used by monitoring apps
* /info – info about application. By default is empty.
  + Update application.properties with app info
    - Info.app.name
    - Info.app.description
    - Info.app.version
  + Properties strating with info will be used by info endpoint

There are 10+ Spring Boot Actuator endpoints:

* /auditevents
* /beans
* /mappings

Full list – luv2code.com/actuator-endpoints

By default, only /health and /info are exposed. To expose all actuator endpoints over HTTP in application.properties:

* management.endpoints.web.exposure.include=\*
* can also expose individual endpoints with a comma-delimited list

You may not want to expose all of this information. Add Spring Security to project and endpoints are secured. /health and /info are still available. Now when you access: /actuator/beans Spring will prompt for login. Default user name: use. Check console log for password. You can override default user name and generated password:

spring.security.user.name

spring.security.user.password

You can customize spring security for spring actuator. To exclude /healt and /info:

management.endpoints.web.exposure.exclude=…

Actuator: luv2code.com/actuator-docs

Development Process:

1. Add spring-boot-starter-security
2. Verify security on actuator endpoints for /bens
3. Disable endpoints for health and info

**72. Spring Boot - Running Spring Boot Apps from the Command Line**

Two options:

* java –jar
* mvnw spring-boot:run

You could also use mvn if maven is installed in your computer.

**73. Spring Boot - Application Properties**

Spring can be configured in the application properties files: luv2code.com/spring-boot-props

The properties are roughly grouped into the following categories:

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

Core Properties:

* logging.level.org.springframework=DEBUG
* logging.level.org.hibernate=TRACE
* logging.level.com.luv2code=INFO
* logging.file=my-craz.log

luv2code.com/spring-boot-logging

Web Properties:

* server.port
* server.servlet.context-path=/my-silly-app
* server.servlet.session.timeout=15m

Actuator:

* management.endpoints.web.base-path=/actuator

Data Properties:

* spring.datasource.url=….
* spring.datasource.username=…
* spring.datasource.password=…

**74. Spring Boot - Build a REST CRUD API with Hibernate - Real-Time Project**

Development Process:

1. Set up Database Dev Environment
2. Create Spring Boot Project using spring boot initializr
3. Get list of employee
4. Get single employee by id
5. Add a new employee
6. Update an existing employee
7. Delete an existing employee

Spring Boot will automatically configure data source based on entries from Maven pom file, JDBC Driver, Spring Data (ORM - jpa), DB connection info from application.properties

Application properties:

* spring.datasource.url=…
* string.datasource.username=…
* string.datasource.password=…

Spring data source properties: luv2code.com/spring-boot-props

Based on configs, Spring Boot will automatically create Beans (DataSource, EntityManager, etc). You can inject them into app.

In Spring Boot Hibernate is default implementation of JPA. EntittyManager is similar to Hibernate SessionFactory. EntityManager can serve as a wrapper to a Hibernate Session object. We can inject the EntityManager into our DAO.

Development Process for DAO:

1. Update db configs in application.properties
2. Create Employee entity
3. Create DAO interface
4. Create DAO implementation
5. Create REST controller to use DAO

**75. Spring Boot - Build a REST CRUD API with JPA - Real-Time Project**

Standard JPA API methods are similar to Native Hibernate API. JPA also supports a query language: JPQL (JPA Query Language)

JPA methods:

* entityManager.persiste(…) – save
* entityManager.find(…) – get, load
* entityManager.createQuery(…) – retrieve list of entities (TypedQuery)
* entityManager.merge(…) – save or update
* entityManager.remove(…) – delete

**76. Spring Boot - Spring Data JPA - Real-Time Project**

Spring Data JPA – Solution:

Create DAO and just plug in your entity type and primary key:

* <https://spring.io/projects/spring-data-jpa>

Spring will give you a CRUD implementation

Spring Data JPA provides the interface: JpaRepository. Exposes methods:

* findAll()
* findById()
* save()
* deleteById()
* etc

Development Process:

1. Extend JpaRepository
2. User Repository in your app

No need for implementation class

public interface EmployeeRepository extends JpaRepository<Employee, Integer>

* Integer is primary key
* Employee is entity

Full list of methods:

luv2code.com/jpa-repository-javadoc

Advanced features available for:

* Extending and adding custom queries with JPQL
* Query Domain Specific Language (Query DSL)
* Define custom methods (low-level coding)

More information: luv2code/spring-data-jpa-defining-custom-queries

JPA repository provides auto Transaction. No need for @Transactional

**77. Spring Boot - Spring Data REST - Real-Time Project**

Spring Data REST:

* Leverages existing JpaRepository
* Gives REST CRUD
* No new coding required

In the background Spring Data REST will scan project for JpaRepository and expose REST APIs for each entity type in repository. By default, Spring will create endpoints based on entity type. Simple pluralized form. First character is lower and entity is in plural.

Add spring data rest to pom file

For spring data rest you only need 3 items:

1. Entity
2. JpaRepository
3. Maven dependency spring-boot-starter-data-rest

In this architecture you don’t need service

Spring Data REST endpoints are HATEOAS compliant. HATEOAS: hypermedia as the Engine of Application State. Hypermedia-driven sites provide information to access REST interfaces:

* spring.io/understanding/HATEOAS

Spring data Rest response using HATEOAS. HATEOAS uses Hypertext Application Language data format

Spring Rest advanced features:

* Pagination, sorting and searching
* Extending and adding custom queries with JPQL
* Query Domain Specific Language (Query DSL)

more information: spring.io/projects/spring-data-rest

Change base path of the REST endpoints

spring.data.rest.base-path=/magic-api

Specify plural name / path with annotation in repository: @RepositoryRestResource(path=”members”)

By default, Spring Data Rest will return the first 20 elements. You navigate to different pages of data using query param (?page=0).

Spring data rest properties:

* default-page-size
* max-page-size

You can sort by the property names of your entity

In out Employee example, we have: firstName, lastName and email:

* ?sort=lastName,firstName,asc

**78. Spring Boot – Thymeleaf**

Thymeleaf is a Java templating engine commonly used to generate the HTML views for web apps. However, it is a general purpose templating engine. You can use Thymeleaf outside of web apps. You can create Java apps with Thymeleaf no need for spring.

Thymeleaf can be an HTML page with some Thymeleaf expressions, include dynamic content from Thymeleaf expressions. In a web app, Thymeleaf is processed on the server. Results included in HTML page returned to browser.

Thymeleaf Use Cases (non-web):

* Email template
* CSV Template
* PDF template

Development process:

1. Add Thymeleaf to Maven pom file

spring-boot-starter-thymeleaf

1. Develop Spring MVC controller
2. Create Thymeleaf template
   1. In Spring Boot, Thymeleaf template files go in src/main/resources/templates directory
   2. For web apps Thymeleaf templates have a .html extension
   3. To use thymeleaf expressions:
      1. <html xmlns:th=htto://thymeleaf.org”>

Additional features:

* Looping and conditionals
* CSS and JavaScript integration
* Template layouts and fragments

Spring Boot will look for static resources in the directory:

* src/main/resources/static

@symbol – Reference context path of your application (app root): @{/css/…}

Spring Boot will search following directories for static resources:

1. /META-INF/resources
2. /resources
3. /static
4. /public

**79. Spring Boot - Thymeleaf - Build HTML Tables**

Development process:

1. Create Employee class
2. Create Employee controller
3. Create Thymeleaf template
   1. Looping <tr th:each=”tempEmployee : ${employees}”>

**80. Spring Boot - Thymeleaf - CRUD Database Real-Time Project**

Thymeleaf has special expressions for binding Spring MVC form data. Automatically setting/retrieving data from Java object:

th:action – Location to send form data

th:object – Reference to model attribute

th:field – Bind input field to a property on model attribute

<form action=”#” th:action=”@{/employees/save}”

th:object=”${employee}” method=”POST”>

<input type=”text” th:field=”\*{firstName} “ placeholder=”Last name”>

luv2code.com/query-methods – repository query