Spring Boot

Traditional approach :

maven archetype, dependencies, configuration, web server

Solution:

make it easier to get started with Spring development

min : configuration

jar file in classpath

properties in prop file

resolve dependency conflict

embedded server

webapp : war files (needs to be configured)
spring boot : jar (by default)

web app : web resources (view files)
jsp-jstl : no configured
spring boot configure for view-template
eg :
Thymeleaf
FreeMarker
Mustache

>mvnw clean compile test
> mvn --------

static : all web static res : HTML, css, js

template : views/ view template

Custom scanning path

com.training.bootapp

org.edu.res

com.prod.res

1 AutoConfiguration:

1. read the pom.xml
2. read the application.properties

2. Launch the web server (container)

use regular spring flow

boot-starter

: a curated list of Maven dependencies

: container of dependencies

starter parent:

Default MAven management,: java version, encoding, spring boot plugin version of boot-starter

relative-Path : spring boot project :

: relative (auto identifies the locaiton of boot-starter)

: any other repository

==> develop rest api

spring-boot-devtools

Spring Boot Actuator:

support added with boot-starter web

Exposes rest endpoint to expose info (monitor/manage)

need to add dependency (to expose endpoint explicitly)

default :

/health : health info

/info : info about project

exposing other endpoints : config the prop file

Add spring-security starter project : thats all

default credentials

username : user

password : generated password

for custom credentials : need to update prop files

for security customization : add security config file...

Implementing DAO in Spring Boot

ORM : Hibernate

Traditional approach (manual config):

1 used Hibernate SessionFactory<----- DataSource<----Connection Info

Spring boot : auto config

based on

1. entries in pom.xml
2. config fld in prop file

Based on config

create beans

DataSource

EntityManager (JPA) (Wrapper around session object)

#will be injected to dao layer

3 version

1. Use EM but leverage native hibernate api
2. use EM and standard JPA API
3. Spring Data JPA

Standard JPA API

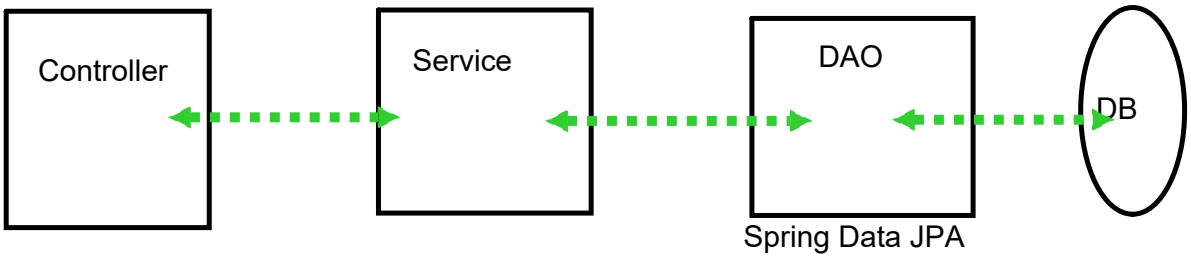
session.save() : entityManager.persist()

session.get() : ---.find()

session.createQuery : ------.createQuery()

HQL ~ JPQL

Spring Data Project : based on JPA spec : uses any backend ORM framework (Hibernate)



DAO for Student Entity

Customer, Employee, Book : need DAO impl.

a specific pattern is there :

two difference in impl:

- Entity Type
- Primary Key

Abstraction of Spring Data JPA

allows to tell Spring:

1. Create a DAO

==>based on entity type and primary key

2. give me all basic crud functionality

findAll()

finById()

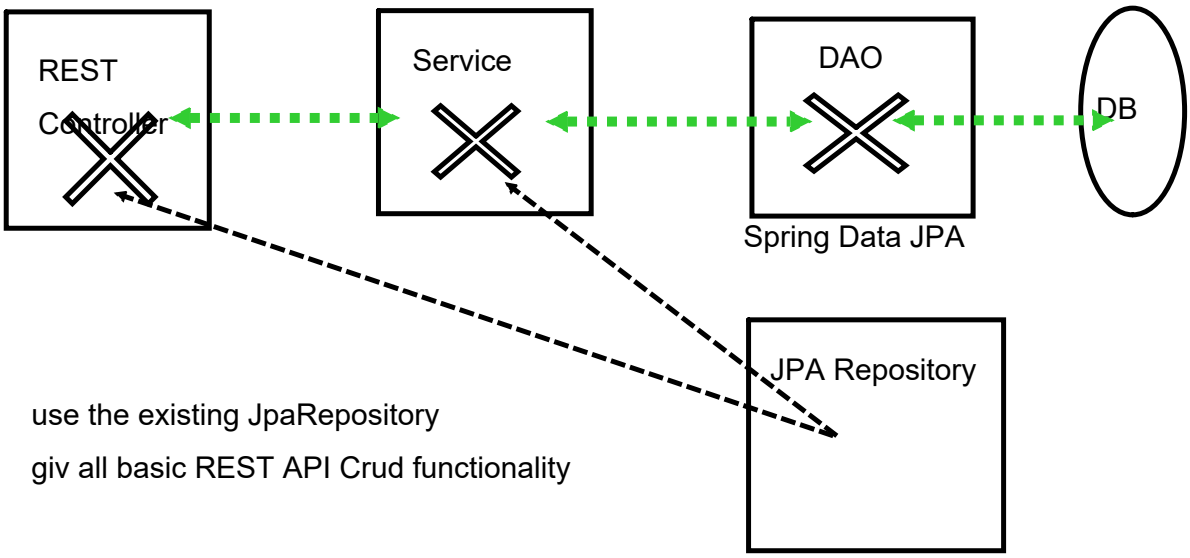
save()

Special interface : JpaRepository (class behind the scene)

inherit this interface in dao interface (plugin entity type and primary key)

Spring Data REST :

abstraction support for REST APIs



use the existing JpaRepository

giv all basic REST API Crud functionality

REST endpoint exposed based on entity class

entity class -> plural form

eg: Student ------>

/students

/students/{studentId}

/students/{studentId}

...

Spring data will scan for all JpaRepository implementation

need to add dependency for data-rest

Spring Data REST endpoints are HATEOAS compliant

Hypermedia as the engine if Application State

meta-data for REST data

provide info as pages

Config :

1. rest endpoints:

customize the rest endpoint

==> JpaRepository implementation : config using annotation

by default : spring boot project create .jar packaging

.war packaging

need to config the main class to let the application start as servlet