Spring cache annotations:

=============================

* The spring cache annotations provides an easy way to manage caching in applications.
* Spring has provided annotations and different cache providers provided implementations.
* The cache improves an application’s performance by reducing the number of interactions with the database.

@Cacheable:

@Cacheable(value = “users”)

public User getUserById(int id) {

//logic

}

* This @Cacheable is used to mark that a method’s return value as cacheable.
* When the method is called, spring checks whether the result already in the cache or not.
* if not, then spring executes the method, caches the result and then returns it.
* if already exist, then spring returns the result by taking from cache, without executing the method.

@CachePut:

* The spring always executes the method, then updates the cache with the result, irrespective of whether the data already exist in the cache or not.
* This annotation is used to update the cache directly.

@CachePut( value = “users”, key = “#user.id”)

public User updateUser(User user) {

//logic

}

@CacheEvict:

* The spring always executes the method, then removes the entry from the cache.

@CacheEvict(value=”users”, key =”#id”)

public void deleteUser(int id) {

//logic

}

@EnableCaching:

This annotation we write at main application class, to enable caching support.

@SpringBootApplication

@EnableCaching

public class DemoApplication {

p s v m(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

}

<https://github.com/ShekherJava/SB-REST-Cache.git>

Spring Boot profiles:

* Spring Profiles is a powerful feature in the Spring Framework that allows you to define and manage different configurations for different environments (e.g., development, testing, production).
* Profiles let you specify which beans to load or which properties to use based on the current environment, making it easier to adjust application behavior without manually changing configuration files.
* With Spring, you can create separate configuration files for each profile by naming them with a suffix that matches the profile. For example:

application-dev.properties (or .yml for YAML)

application-prod.properties

* Each file will contain properties specific to that environment. Spring automatically loads the file matching the active profile, which overrides the default properties.
* You can specify the active profile in your application.properties file, with the below property.

spring.profiles.active=dev

* You can define multiple profiles in a single application.yml file also.
* You have to use profile-specific sections with the

--- separator.

* Each profile section should start with spring.config.activate.on-profile, to specify the profile it belongs to.

example applicatin.yml file:

spring:

application:

name: SB-Profiles

profiles:

active: test

---

spring:

config:

activate:

on-profile: "dev"

profile:

id: 1

name: Dev Profile

---

spring:

config:

activate:

on-profile: "test"

profile:

id: 2

name: Test Profile

---

spring:

config:

activate:

on-profile: "prod"

profile:

id: 3

name: Prod Profile

<https://github.com/ShekherJava/SB-Profiles.git>

Spring Boot Actuator:

--------------------

* Spring Boot Actuator is a module which provides pre-defined REST endpoints, for monitoring and inspecting the various aspects of the spring boot application.
* It means, like we can get the health checks, metrics, configurations, runtime details of the currently executing application.
* To enable Actuator support in Spring Boot, we need to add spring-boot-starter-actuator dependeny in pom.xml and we need to enable all the actuator endpoints by adding the below property to the properties fie.

management.endpoints.web.exposure.include=\*

* The root endpoint for all the actuator endpoints is /actuator
* Except /actuator/shutdown, the remaining actuator REST endpoints are accessible through HTTP GET. But shutdown endpoint is accessible through HTTP POST.

/actuator/health : provides application’s health status.

It is used to determine, whether an

application is healthy to handle the requests.

. By default, it provides the status as UP or DOWN.

. To get the health status with components like database connectivity, disk space and connection to external services, then we need to add the below property in the application.properties file.

management.endpoint.health.show-details=always

/actuator/metrics : provides the application metrics like JVM metrics, HTTP request metrics, connection pooling metrics, etc.

. This endpoint displays the unique names of the metrics and we can get the details of each metric with metrics/{metric-name}

/actuator/beans : Provides all the beans in the Spring Application Context including their dependencies.

/actuator/mappings: provides all the request mappings in the application. It includes details like, endpoint paths, request methods, handler methods, etc.

/actuator/shutdown: Allows to shutdown the application.

It is disabled by default, so to enable, add

the below property in application.properties.

management.endpoint.shutdown.enabled=true

(This endpoint can’t be tested from browser. User POSTMAN, and send request with HTTP POST).