What is difference between system.gc() and runtime.gc().

* Both are same . system.gc() is effectively equivalent to runtime.gc(). System.gc() internally calls runtime.gc()

The only difference is system.gc() is class method where as runtime.gc() is an instance method so system.gc() is more convenient.

How to force garbage collection in java?

* Your best option is no call system.gc() which simply is a hint to the garbage collector that you want it to do a garbage collection.

There is no way to force and immediate collection through as the garbage collector is non-deterministic.

* The most commonly used http verbs POST,GET,PUT,DELETE are similar to CURD operation in database we specify these HTTP verbs in the capital case so the below is the comparison between them.

1. Create – POST
2. Read – GET
3. Update -PUT
4. Delete-DELETE

PATCH- submit a partial modification to a resource if you only need to update one field for the resource you may want to use the patch method.

Construction VS Setter Injection?

* We have two type of dependency injections constructor injection and setter injection here are the point that make understand when to use contractor inject and setter injection.

|  |  |
| --- | --- |
| Constructor Injection | Setter Injection |
| 1. At the time of creating your target class object , the dependent objects are injected (can be accessed in the constructor of target class) 2. In case of constructor injection all the dependent objects are mandatory to be injected . if you don’t provide any of the dependent object through < constructor-arg> tag the core container will detects and throw BeanCreationException 3. If classes have cyclic dependencies via constructor, these dependent beans cannot be configured through constructor injection | 1)the dependent objects are not injected while creating the target classes object. Those will be injected after the target class has been instantiated, by calling the setter on the target object.  2) in case of setter injection your dependent objects are optional to be injected. Even you don’t provide the <property> tag while declaring the bean , the container will creates the bean an initializers all the properties to their default .  3)cyclic dependencies are allowed in setter injection. |

What is diff between Save() & Persist()

|  |  |
| --- | --- |
| Save() | Persist() |
| 1)The return type of save is java.io.serializable it returns the generated id value .  2)save() method can be used inside or outside the transaction boundries .  3)save() method is only supported by hibernate | 1)the retrun type of pesist() is void() it doesn’t retrun any value  2)persist() method can be used only within the boundry of a transaction.  3)persist() method is supported by JPA |

What is diff between Load() and get().

|  |  |
| --- | --- |
| Load() | Get() |
| 1)only use Load() method if you are sure that the object exists.  2)load() method will throw an exception if the unique id is not found in the database  3)load() just return a proxy be hit default and database won’t be hit until the proxy is first invoked | 1) if you are nit sure that the object exist then use one of get() method.  2)get() method will return null if the unique id is not found in database  3)get() will hit the database immediately. |

What is @Qualifier annotation?

When you need more control of the dependency injection process @Qualifier can be used

This annotation is used to avoid confusion which occurs when you create more than one bean of the same type and want to wire only one of them with a property.

Consider an example where an interface BeanInterface is implemented by two beans BeanBI and BeanB2

@Component

Public class BeanB1 implements BeanInterface{

// }

@component

Public class BeanB2 implements BeanInteface{

// }

Now if BeanB1 autowire this interface, spring will not know which one of the two implementation to iject one solution to this problem is the use of the @Qualifoer annotation

@Component

Public classBeanA {

@Autowired

@Qualifier(“BeanB2”)

Private BeanInterface depenedcy;

// }

* Spring Boot Scheduling

Scheduling is the process of executing a piece of logic at a specific time in the future. Scheduled jobs are a piece of business logic that should run on a timer. Spring allows us to run scheduled jobs in the Spring container by using some simple annotations.

@SpringBootApplication

@EnableScheduling

public class JobschedulingApplication {

public static void main(String[] args) {

SpringApplication.run(JobschedulingApplication.class, args);

}

}

@Scheduled(initialDelay = 1000, fixedRate = 10000)

public void performDelayedTask() {

System.out.println("Delayed Regular task performed at "

+ dateFormat.format(new Date()));

}

* what are some standard java per-defined functional interface ?

some of the famous pre-defined functiona interfaces from prvious java versions are Runnable, Callable, Comparator and Comarable. While java 8 introduces functional interfaces like Supplier ,Cansumer,Predicate etc.

Runnable – use to execute the instance of the class over another thred with no argument and no retrun value.

Callable – use tjo execute the instances of class over another thread with no arguments and it either retruns a value or thorew an exception.

Comparator – use to sort diff objects in a user defined order.

Comparable – use to sort object in the natural sort order

* spring singleton scope.

Singleton is default bean scope in spring container. It tells the container to create and manage only one instance of bean class, per container. This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean retrun the cached instance.

Prototype – definition to any number of object instances

Request – single bean definition to the lifecycle of a single http request, that si, each HTTp request has its own instance of a bean created off the back of a single bean definition only valid in the contextof a web aware spring applcationcontext.

IOC(Inversion Of Control) :-

Giving control to the container to create an dinject instances of objects that your application depend upon, means instead of you are creating an object using the new operator , let the container do that for you . inversion of container relies on dependency injection because a mechanism is needed in order to activate the components providing the specific functionality

The two concepts work together in this way to allow for much more flexible , reusable and encapusated code to be written as such they are important concept in designing object -oreinted solution

DI(Depedency Injection) :-

Dependecy Injection is a design pattern used to create instances of objects that other object rely upon without knowing at complie time which class will be used to provide that functionality or simply the way of injecting properties to an object is called depenecy injection.

There are three type of Dependency Injection

1)Constrcutor Injection

2)Setter/Getter Injection

3)Interface Injection

Spring Support only Constructor and Setter Injeection.

JAVA ================🡺 JSON

1. convert JAVA to JSON

ObjectMapper mapper = new ObjectMapper();

User user = new User();

//Object to json in file

Mapper.writeValue(new File(“c:\\user.json”),user);

//object to Json in String

String josonInString = mapper,writeValueStrin(user);

* Spring Boot Actuator?

Spring boot incudes a number of additional features to help you monitor and manage your application when you push it to production you can choose to manage and monitor your application

By usin Http endpoints or with JMX. Auditing , health and matrics gathering can also be automatically applied to your application

To add actuator to a Maven based project and the ‘starter’ dependency in pom.xml

By default , all endpoints except for shutdown are enabled to configure the enablememt of an endpoint use its management endpoint <id> enabled property

To access endpoint :

<http://localhost:8080/actuator/logger>

more endpoint:

beans , caches , condition,configrops etc