Spring Data JPA

=>Spring data JPA internally uses hiberante as ORM framework

=>So strong knowledge in hibernate Programming definetely helps to work with spring data jpa effectively.. (hibernate knowledge is not mandatory)

=>While developing Entity classes/model classes we need to prefer using annotations in the following order

a)JPA annotations (portable across the mutliple ORM s/ws) b) Java Config annotations (given by JDK/JEE)

c) Hibernate Specific Annotations d) Third party Annotations

Repositories in spring data jpa (boot 2.x)

==========

Common Repository

interfaces

for both SQL

and NO SQl

DB s/w..i.e these

12+2 methods

can be used

in both SQ

or NO SQL DB

Apps

==========

<<Interface>>

Repository<T, ID extends Serializable>

(marker interface-No methods) extends

<<Interface>>

In spring data jpa programming, for 500 db tables we take

--> 500 custom repository interfaces extending from pre-defined Repository interfaces mapped with 500 entity classes --> 500 Entity classes mapped with 500 db tables

In o-r mapping the java bean class that is mapped with Db table is technically called an entity class or domain class or model class (best)

Persistence class (or)

=> By using different annotations given by different vendors we map entity classes with Db tables and we Map the properties of entity classes with Db table cols.

The interface that contains no methods and makes the underlying JVM /server/Container to provide special runtime capabilities to impl class objs is called called Marker Interface /tag interface

eg:: java.io.Serializable java.lang.Cloneable

javax.servlet.SingleThreadModel and etc..

<<Interface>>

CrudRepository<T, ID extends Serializable>

QueryDs PredicateExecutor<T>

(12 methods)

extends

<<Interface>>

PagingAndSortingRepository<T, ID extends Serializable>

(2 methods)

extends

extends

of

=> In spring data jpa, we just develop custom DAO Interface/Repository Interface extending ready made Reposiory Interface to inherit different level common methods that are required for persistence operations. But we do not develop any impl classes having persistence logics.. becoz this job will be taken care spring data jpa internally by generating InMemory Proxy classes as the impl classes of custom DAO/ Repository interface.. In that class even persistnece logics will be developed in o-r mapping style automatically by using the Entity class and Id Property (PK col Property type) as the inputs

note:: The top interface in the Repositories interfaces is

Repository(1) which is an empty marker interface ..It is maintained as empty/marker interface to make all Repository interfaces

belonging to same type/cult.

note:: In spring boot data jpa application, the IOC container generates InMemory Proxy classes for our Repository interfaces having o-r mapping persistence logic by seeing this

<<Interface>>

<<Interface>>

JpaRepository<T, O extends Serializable> (8 methods) JpaSpecificationExecutor<T>

(same as CrudRepository

but contains JPA specific

methods like methods taking

Example objs)

Spring Data JPA

Java Code Geeks

ONERS RESOURCE CENTER

(Based on CrudRepository)

<interface> MongoRepsitory<T,ID>

(4 methods)

(methods specific to

MongoDB)

Procedure to develope our first Spring data JPA application as standalone App (partially layeed App)

=====

(Runner class)

Spring Data mongoDB

Repository (1) directly or indirectly

Client App

(presentation loigc)

service class (b.logic)

custom (1)

- Repository ----> Dbs/w Custom (InMemory Impl class of Repository(!) contains peristence logic)

step1) create spring boot starter Project adding the following starters as dependencies

Available: jpa

▾ SQL

Spring Data JPA

order of development: a) Entity class

Selected:

X Lombok

Spring Data JPA

Oracle Driver

➡gives HB jar files

+

Hikaricp jar files

+

spring data jpa jar files

=>We make our custom Repository interface extending from CrudRepository/PagingAndSortingRepository/JpaRepository interface

but the IOC container generates In Memory Proxy class implementing our interface having lots of persistence logics

b) custom repository(1) c) service (I) and service Impl (c) d) Runner class /Client app step2) add the datasource, hiberante specific properties in application.properties file.

Repository intefaces hierarchy of spring data jpa in spring boot 3.x

===

Crud Repository(1)

extends

Repository(1)

extends

(tag/marker interface)

PagingAndSortingRepository(1)

HB-Hibernate

application.properties

#DataSource cfg

spring.datasource.driver-class-name-oracle.jdbc.driver.Oracle Driver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe

spring.datasource.username=system

spring.datasource.password=manager

Spring.datasource.hikari.minimum-idle=10

spring.datasource.hikari.keepalive-time=100000

#JPA-Hiberante properties

spring.jpa.database-platform=org.hibernate.dialect.Oracle Dialect

spring.jpa.show-sql=true

(Dialect)

spring.jpa.hibernate.ddl-auto-update

ListCrudRepository(1)

JpaRepository(1)

(SQL DB s/w)

 # other possible values create,validate,create-drop

note1:: Dialect is HB internally managed component/service that is capable of generating SQL queries based on the underlying DB s/w and its version. All dialects are the classes extending from org.hibernate.dialect.Dialect(c). The dialect value will change based on Db s/w and its version we use.. It is optional property to specify becoz based on JDBC /dataSource properties we cfg..the Dialect comp will be picked up automatically... To get all dilaects :: https://www.javatpoint.com/dialects-in-hibernate

spring.jpa.show-sql=true

=>shows the dialect comp generated SQL queries as log messages..on the console

spring.jpa.hibernate.ddl-auto-update

To use Dynamic shema generation

 # other possible values create, validate,create-drop,nonde feature of Hiberante

create :: Always creates new db tables ..by dropping the db tables if already existted update :: if db tables are already available then uses them,

(best)

if modifications required alters them (only adding of new cols is possible)

if db tables are not there, then creates new db tables according to the entity classes. (So useful project Development and production env..)

validate :: Does nothing on its own expect verifying whether db tables are there accoring entity classes or not i.e db tables must be created by developers/DB team manually.. create-drop :: Creates Db tables at startup of App ... uses them through out App's execution and drops them at the end of the Application. So useful in "test", "UAT" env.. also in Demos of projects and POCS

be

none:: does nothing.. Db tables should created manually, even verification does not takes place (it is default value) For spring boot applicaiton properties

https://docs.spring.io/spring-boot/docs/current/reference/html/application-

properties.html#application-properties.data

step3) Create Entity class having JPA annotations based o-r mapping cfgs

creating

While Entity class we use generally following annotations as minimun annotations @Entity mandatory annotations

extends

ListPagingAndSortingRepository(1)

exstends

MongoRepository(1)

CassendraRepository(1)

↓ (NO SQL DB s/w)

(NO SQL DB s/w)

RDBMS

Dialects

DB2

org.hibernate.dialect.DB2Dialect

DB2 AS/400

org.hibernate.dialect.DB2400Dialect

DB2 05390

org.hibernate.dialect.DB2390Dialect

PostgreSQL

org.hibernate.dialect.PostgreSQLDialect

MySQLS

org.hibernate.dialect.MySQL5Dialect

MySQL5 with InnoDB

org.hibernate.dialect.MySQLSInnoDBDialect

MySQL with MyISAM

org.hibernate.dialect.MySQLMyISAMDialect

Oracle (any version)

org.hibernate.dialect.OracleDialect

Oracle 91

org.hibernate.dialect.Oracle9iDialect

Sybase

org.hibernate.dialect.SybaseASE15Dialect

Microsoft SQL Server 2000

org.hibernate.dialect.SQLServerDialect

Microsoft SQL Server 2008

org.hibernate.dialect.SQLServer2008Dialect

SAP DB

org.hibernate.dialect.SAPDBDialect

Informix

org.hibernate.dialect.InformixDialect

HypersonicSQL

org.hibernate.dialect.HSQLDialect

H2 Database

org.hibernate.dialect.H2Dialect

SAP DB

org.hibernate.dialect.SAPDBDialect

Informix

org.hibernate.dialect.InformixDialect

HypersonicSQL

org.hibernate.dialect.HSQLDialect

H2 Database

org.hibernate.dialect.H2Dialect

Ingres

org.hibernate.dialect.IngresDialect

Progress

org.hibernate.dialect.ProgressDialect

Mckol SQL

org.hibernate.dialect.MckolDialect

Interbase

org.hibernate.dialect.InterbaseDialect

Pointbase

org.hibernate.dialect.PointbaseDialect

FrontBase

Firebird

org.hibernate.dialect.FrontbaseDialect org.hibernate.dialect.FirebirdDialect

jpa

@Id

annotaitons @Column

@Table optional annotations

and etc...

@Entity ---> To mark Java bean as the the entity class

Student obj

10 101

raja hid 899

90.0

(obj to table record) update Student_Tab set avg=90.0

where sno=101

delhi select * from student_tab where sno=101 (record to object)

@Id -----> To mark the Property as Identity property (based on this property value, the entire Object will be identified)

(generally apply this on the property that is mapped with PK colum of the db table)

note:: the ORM f/w takes @Id property value as the criteria value

to get db table record value into entity class object and vice-versa. (meaning is this id proerty value of the entity class objs will be used as the criteria/condition values to get get Synchronization b/w Entity class obejcts and db table records)

@Column -----> To map the entity class property with db table colum @Table ------> to map Entity class name with db table name

student Tab record

101 raja hyd 899

90.0

delhi

In ORM, the objects of Entity class

will be identified with id value (id property value)

if the entity class name matching with db table name

and entity class property names are matching with

db table col names facing of @Table and @Column

is an optional process. In this scenario, the Entity class name will be taken as

the table name and property names

note:: Do not take table with out PK column in O-R mapping Any how, we prefer generating new DB tables in spring data jpa where the @Id property column automatically becomes PK column of the DB table

will be taken as the col names

```
//Doctor.java package com.nt.entity;

import jakarta.persistence.Column; import jakarta.persistence.Entity; import
jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id; import jakarta.persistence.Table; import lombok.Data;

@Entity

@Table(name="JPA_DOCTOR_INFO")

@Data

public class Doctor {

@Column(name="DOC_ID")

@Id
```

=>We need to place @Id property in the Entity class irrespective of Db table is having PK column or not

=> Only for @Id property, we generate the values dynamically towards saving the objects (inserting the records) by using one or another generators support like AUTO, SEQUENCE,TABLE,IDENTITY and etc..

(default)

lombok api annotation

@GeneratedValue(strategy = GenerationType.AUTO)

private Integer docId;

@Column(name="DOC_NAME",length = 25)

private String docName;

@Column(name="SPECIALIZATION",length = 20)

private String specialization; @Column(name="INCOME")

private Double income;

}

=> DB table name, col names, SQL keywords are not case- sensitive, but db table records data (col values) are case- sensitive

=> Every thing in java programming is case sensitive

ALL JPA generators are AUTO(default), SEQUENCE, TABLE,IDENTITY,UUID (new)

These generators common for all the ORM frameworks

=> Do not insert values colleted from enduser that are having link with outside business or govt policies to the PK column of db table becoz

they might be chaged time to time (Do not take natural key column as the PK column) eg:: making aadharno, mobile no, voterid cols as the pk cols

=> Always insert either App generated or DB s/w generated values to Db table PK col dynamically

eg: making DB sequence generated value as the PK col value making application generated value as the PK col value

@GenerateValue is applicable along with @Id property to cfg one or another generatate/JPA generators generated value as the PK col value

to generate the value to id property dynamically for each save object operation (record insertion)

i.e it is indirectly inserting values to PK col dynamically based on generator we have cfg..

=>The AUTO generator is the default generator if no generator is cfg

=>This AUTO generator in oracle DB s/w creates sequence having logic to generate the values start with 1 increment by Bo

as oracle sequence (1,2,50,52,102,152,....)

=> The AUTO Genrator in MySQL DB s/w creates some helper db tables to generate the id values

in the following sequence like 1,2,50,52,102 and etc.. (MySQL DB s/w do not support sequences)

=> using "length" attribute of @Column annotation we can control only string property these a related db table column lengths while using Dynamic schema generation process. (Dynamic db table generation) useful not while i.e we can control the length of numberic properties related db cols using length attribute of @Column annotation creating db tables

based entity class

=> we crete entity classes on 1 per db table basis..

step4) create Custom Repository Interface extending from CrudRepository(1)

note:: we do not develop impl class for this custom Repository (1) becoz the spring JPA generates InMemory Proxy class implementing Custom Repository(1) having implementation for inherited methods of Crud Repository with HB Persistence logic.

In Lombok API::

All JPA Generrators list is =>AUTO (default) =>IDENTITY

=>TABLE

=> SEQUENCE

=>UUID

Prefer taking Surrogate key column as the PK column

@NoArgConstructor :: Gives 0-param constructor @AllArgConstructor :: Gives parameterized constructor involving all props @RequiredArgConstructor :: Gives parameterized constructor involving only required props on whom @NonNull annotation is placed

All Methods

Instance Methods

Abstract Methods

Modifier and Type Method

Description

Returns the number of entities available.

Deletes a given entity.

Deletes all entities managed by the repository.

`deleteAll(Iterable <? extends T> entities) Deletes the given entities.`

long

`count()`

`void`

`delete(T entity)`

`void`

`deleteAll()`

`void`

`void`

`void`

`boolean`

deleteAllById(Iterable <? extends ID> ids) Deletes all instances of the type T with the given IDs.

Deletes the entity with the given id.

Returns whether an entity with the given id exists.

Returns all instances of the type.

`deleteById(ID id)`

existsById(ID id)

Optional <T>

<S extends T>

save(S entity)

S

Iterable <T>

Iterable <T>

findById(ID id)

findAll()

findAllById(Iterable <ID> ids)

Returns all instances of the type T with the given IDs. Retrieves an entity by its id.

Saves a given entity.

<S extends T> saveAll(Iterable <S> entities) Iterable <S>

Saves all given entities.

=>Every stor

sitory mus be mappe

with Entity class and @Id property type

so that inherited mehtods pre-defined Repository interfaces like CrudRepository,

PagingAndSortingRepository and etc.. will get the param types and return types dynamically becoz they are generics based

//IDoctorRepo.java

```
package com.nt.repository;

import org.springframework.data.repository.CrudRepository;

import com.nt.entity.Doctor;

public interface IDoctorRepo extends CrudRepository<Doctor, Integer> {

=> In spring data jpa application, we get DataSoruce object through AutoConfiguration process and we also InMemory Proxy class as the impl class of CustomRepostory Interface and also as @Repository annotation based spring bean injected with DataSource obj dynamically at runtime. So this InMemory Proxy class based Custom Repository Impl class obj can be injected to Service Impl class object using @Autowired annotation

}
```

<T>

ID

Entity class

@Id property type

name

step5) Develop service Interface and Servcie Impl class

note:: Using @Autowired, we can Inject the Dynamically generated InMemory Proxy class object (Custom Repository(1) Impl class obj) to Service Impl class.

Service interface

```
//IDoctorService.java
package com.nt.service;
import com.nt.entity.Doctor;
public interface IDoctorService {
public String registerDoctor(Doctor doctor);
save
}
```

Service Impl class

```
//DoctorMgmtServiceImpl.java
package com.nt.service;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.nt.entity.Doctor;
import com.nt.repository.IDoctorRepo;
@Service("doctorService")
public class DoctorMgmtServiceImpl implements IDoctorService {
@Autowired
private IDoctorRepo doctorRepo;
@Override
public String registerDoctor(Doctor doctor) {
System.out.println("doc Id(before save::"+doctor.getDocId());
Doctor doc=doctorRepo.save(doctor);
return "Doctor obj is saved with id value :"+doc.getDocId();
}
```

`<S extends T> S save(S entity)`

Saves a given entity. Use the returned instance for further operations as the save operation might have changed the entity instance completely.

Parameters:

entity - must not be null.

Returns:

the saved entity; will never be null.

Throws:

`IllegalArgumentException - in case the given entity is null.`

OptimisticLocking FailureException - when the entity uses optimistic locking and has a version attribute with a different value from that found in the persistence store. Also thrown if the entity is assumed to be present but does not exist in the database.

```
Doctor doc=doctorRepo.save(doctor);
```

This save(-) implementation is there in the

dynamically generated in memory proxy class having hibenrate persistence logic.. This method performs

=> The Auto Generator cfg on the @Id Property by taking oracle as underying DB s/w behaves differently in different versions of spring boot 2.x and in spring boot 3.x

a) collects given Entity class obj (Doctor obj) from save(-)

step6) Develop the Client App

//ClientApp

b) collects the generat or configured on the @Id

Property and generates id value to @Id Property

using the cfg generator (here AUTO generator)

c) collects given Entity object data including @Id property

value and generates JDBC code + INSERT SQL Query

to insert the entity obj data as the record to db table

d) returns

same

object Entity class having the id value and old data.. (here doc obj) (we can say this returned

also having

In spring boot 2.x

=> use/creates hibernate_sequence as the sequence to generate

id value having logic start with 1 incremant by 1

In spring boot 3.x

eg: 1, 2,3,4,5,....

=>use/creates <db talble>_seq as the sequence name to generate id value having logic start with 1 increment by 50

obj and inserted record are

in mapping/synchronization)

```
package com.nt;
import org.springframework.boot.SpringApplication; import
org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext; import
org.springframework.context.ConfigurableApplicationContext;
import com.nt.entity.Doctor;
import com.nt.service.IDoctorService;
@SpringBootApplication
public class BootDataJpaProj1Crud RepositoryApplication {
public static void main(String[] args) {
//get IOC container
ApplicationContext ctx-SpringApplication.run(BootDataJpaProj1CrudRepositoryApplication.class, args);
```

```
//get Service class obj
IDoctorService service=ctx.getBean("doctorService", IDoctorService.class);
}
eg: 1, 2, 52, 102,152,....
try {
// create Doctor class object
Doctor doctor=new Doctor();
doctor.setDocName("raja"); doctor.setSpecialization("MD-Cardio"); doctor.setIncome (890000.0);
//invoke the b.method
String resultMsg=service.registerDoctor(doctor);
Testing the App using Runner class
System.out.println(resultMsg);
=======
========
}
catch(Exception e) {
@Component
e.printStackTrace();
(or)
}
public class Crud RepoTestRunner implements Command Line Runner{ @Autowired
private ICustomerManagmentService custService;
//close the IOC container
((ConfigurableApplicationContext) ctx).close();
step7) Run the main class
output window
2023-01-22109:35:05.572+05:30 INFO 1180 --- [
mainj o.n.e.t.j.p.i.Jtariatтorminitiat
2023-01-22T09:35:05.587+05:30 INFO 1180 --- [ main] j.LocalContainerEntityManage
2023-01-22T09:35:05.930+05:30 INFO 1180 --- [
doc Id(before save::null Hibernate: select jpa_doctor_info_seq.nextval from dual
main] ootDataJpaProj1Crud Reposito
Hibernate: insert into jpa_doctor_info (doc_name, income, specialization, doc_id) values (?, ?, ?, ?)
Doctor obj is saved with id value :1
2023-01-22T09:35:05.980+05:30 INFO 1180 --- [ main] j. LocalContainerEntityManagerFactoryBe
2023-01-22T09:35:05.982+05:30 INFO 1180 --- [ 2023-01-22T09:35:05.988+05:30 INFO 1180 --- [
main] com.zaxxer.hikari.HikariDataSource main] com.zaxxer.hikari.HikariDataSource
```

:

:

Activate Windows

JPA_DOCTOR

INFO

Columns Data Model | Constraints Grants Statistics Triggers Flashback | Depende

INCOME SPECIALIZATION

Sort.. Filter:

DOC_ID

DOC_NAME

1

1 raja

2

2 raja

@Override

public void run(String... args) throws Exception {

Customer cust=new Customer();

}

cust.setCname("rajesh"); cust.setCaddrs("hyd");

cust.setBillAmt(89700.0f);

String msg=custService.registerCustomer(cust);

System.out.println(msg);

Configuring SEQUENCE Generator to generate

id values in the sequence

In the Entity class @Column(name="CID")

890000 MD-Cardio

45.788

890000 MD-Cardio

For two time execution

--> precision:: 5

of the Application

---> scale :: 3

@Id

Making the above app working for mysql

step1) make sure that logical DB ntspbms901db is ready in mysql DB s/w (Use MySQL workbench)

step2) add mysql Connector/j jar file /starter to the project.

Right click on project ---> spring --> add starters--->

Service URL:

https://start.spring.io

Spring Boot Version: 3.0.2

Frequently Used:

Lombok

Available:

mysql

▾ SQL

✓MCOLDuin

Oracle Driver

Spring Data JPA

Selected:

X

X MySQL Driver

MySQL JDBC driver.

Guides

• Accessing data with MySQL

-->next-->next --->... --> finish

step3) change the jdbc properties and jpa properties to mysql

application.properties

#jdbc properties (for mysql)

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql:///NTSPBMS616DB

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.hikari.maximum-pool-size=100

spring.datasource.hikari.minimum-idle=10

spring.datasource.hikari.keepalive-time=100000

#Hibenrate specific JPA Properties

spring.jpa.datasource-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto-update

step4) run the Application

@Id

(logical name)

Replace Customer with

Doctor while doing the pratice.

(sequence name)

@SequenceGenerator(name="gen1",sequenceName = "CNO_SEQ", (increment by value)

(start with value) initialValue = 100,allocationSize =1)

@GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE) private Integer cno;

(must match

with above logical name)

Generates the id values 100,101,102,103 and etc..

another example on Sequence Generator with default inputs

@Column(name="DOCTOR_ID") @GeneratedValue(strategy = GenerationType.SEQUENCE)

private Integer did;

This crerates the sequence with name

<table_name>_seq and generates the id values

as 1,2, 52,102, 152 and etc..

(Start with 1 increment by 50)

=>There is no support for sequences in MYSQL, but the effect of sequence will come

by generating helper table having name "<table_name>".seq

=>The helper table that is generated in MYSQL for "AUTO" and SEQUENCE Generator cfgs will be having one column "next_value" holding the next value to generate

JPA_DOCTOR_SEQ (helper table)

==========

next_value

1-> 2 --> 52-> 102 ....

Result Grid

Filter Rows:

Edit:

doc_id

doc_name

income

specialization

1

raja

890000

MD-Cardio

2

raja

890000

MD-Cardio

NULL

NULL

NULL

NULL

note:: In MYSQL DB s/w, there is no support for Sequences..

So it will use helper db tables in the process of inserting records

in db table. This table always hold next value to generate

to the db table as part next record insertion or next obj saving

The "AUTO" Generator of JPA in spring boot 3.x uses diffrent

underlying concepts in different Db s/ws to generate id values

as shown here 1, 2, 52, 102, 152,202 and etc..

JPA Generators that can be applied on @Id property of the Entity class

AUTO (default)

SEQUENCE

=> if no generator is specified in

What is Candidate key in SQL DB s/w?

TABLE

IDENTITY

UUID

Ans) => The column in DB table that holds unique values using which each record can be identified and retrived is

called Candidate key column

=> eg:: voterid column, mobile no colum, aadharno column, panCard column and etc..

What is natural key columm?

Ans) The candidate key column is DB table whose values are dependent on outside world business or govt policies are called natural key column. Values these columns can be collected from endusers

eg:: voter id colum, mobileno column, aadhar no column and etc..

What is Surrogate key column?

Ans) The candidate key column in DB table whose values are generated by underlying

DB s/w or Application with out having any link with outside world business policies and

govt policies and also not expected from the end users is called surrogate key column eg: emp id generated through oracle sequence customer id generated through MySQL auto increment

order id generated through hibernate generators (pre-defined /custom)

Limitations of taking natural key column as the PK column?

a) values very lengthy, so they need more memory in the columns

b) these values will be changed becoz of the change in govt or business policies

which effect db tables and their dependent db tables and java classes which is a costly process

c) These values are expected from endusers, if enduser fails gives then record insertion can not be done

Advantages of taking surrogate key column as the PK column?

a) they are small values so they allocate less memory

b) Not expected from endusers

c) Generated by the underlying App/Db software dynamically So they will not be changed/distributed for the outside world GOVT/Business Policies

count() method of CrudRepository

========== there

=> Gives records count that are in Db table

count

long count()

Returns the number of entities available.

Returns:

the number of entities.

Example code

In service Interface

public long showDoctorsCount();

In Service Impl class

@Override public long showDoctorsCount() {

long count=doctorRepo.count();

return count;

}

In Runner class

==========

try {

}

System.out.println("Doctors count ::" + docService.showDoctorsCount());

catch (Exception e) {

}

e.printStackTrace();

the @Generated Value annotation then

AUTO will be taken as the default generator