Date : 11/09/2020

Spring Boot 7AM

Mr. RAGHU

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

application.properties --> java.util.Properties(C) object

Q) What are difference between Map<K,V> and Properties(C) ?

A) Map<K,V> supports generics (new collection, JDK 1.2, Interface)

Properties(C) supports no Generics (legacy collection, JDK1.0,class)

Here Properties(C) read data from application.properties in key-val

format as both String type only.

-Syntax to provide Properties(C) data in application.properties file-

## prefix.variable.KEY=VALUE

--Example--

1. Create one Starter Project

Name : SpringBoot2ConfigPropsProperties

Dependecy: Spring configuration processor

2. Runner class

package in.nareshit.raghu.runner;

import java.util.Properties;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

@Component

@ConfigurationProperties(prefix = "my.app")

public class MyDataReader implements CommandLineRunner {

private Properties pdata;

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

System.out.println(pdata);

}

public Properties getPdata() {

return pdata;

}

public void setPdata(Properties pdata) {

this.pdata = pdata;

}

}

3. application.properties

#prefix.variable.KEY=VALUE

my.app.pdata.P1=A

my.app.pdata.P2=B

my.app.pdata.P3=C

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

Association Mapping (HAS-A): Using one class(child) as a DataType in another class(Parent)

and creating one variable.

@ConfigurationProperties even supports converting properties data into one class object,

by using HAS-A Relation.

Syntax:

prefix.hasAvariable.variable=value

--code--

1. Create one Starter Project

Name : SpringBoot2ConfigPropsProperties

Dependecy: Spring configuration processor

2. model class

package in.nareshit.raghu.model;

public class DbConn {

private String driver;

private String url;

private String username;

private String password;

//set,get..

//toString..

}

3. Runner class

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.context.properties.ConfigurationProperties;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.DbConn;

@Component

@ConfigurationProperties(prefix = "my.app")

public class MyDataRunner implements CommandLineRunner {

private DbConn con;

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

System.out.println(con);

}

//set,get..

}

4. application.properties

my.app.con.driver=oracle

my.app.con.url=jdbc:oracle

my.app.con.username=nit

my.app.con.password=raghu

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

Project Lombok | Lombok API

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

Project Lombok is Opensource Java API, that is used to generate code for set/get methods,

toString, equals, hashCode, constructors..etc

\*\*\* Lombok generates code based on annotations provided and given to java compiler.

\*\*\* Steps to Activate Lombok \*\*\* (onetime setup)

#1. Create one Starter Project with lombok dependency

> File > new > Spring Starter Project > Enter details > Next

> Search using Lombok > Select same > Finish

#2. Wait until project gets created in STS, then close STS

#3. Goto location:

C:\Users\Raghu\.m2\repository\org\projectlombok\lombok\1.18.12

#4. Double click on jar name : lombok-1.18.12.jar

(or)

> open cmd > type commands as

c:/> cd C:\Users\Raghu\.m2\repository\org\projectlombok\lombok\1.18.12

> press enter

c:./> java -jar lombok-1.18.12.jar

#5. Click on Specify location of STS

Ex location: E:\SoftwaresLatest\sts-4.7.1.RELEASE

#6. Click on Install/Update

#7. Close Installer after success message

#8. Open STS again

#9. create below class in our project

package in.nareshit.raghu.model;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

//ctrl+shift+O (imports)

@Getter

@Setter

@ToString

@EqualsAndHashCode

public class Student {

private Integer sid;

}

#10 Expand Package Explorer to view added/genrated code in class

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

Lombok Annotations:

1. @Getter and @Setter :

This annotation says to lombok generate get and set methods for variables

2. @ToString : Lombok generates toString() method code using variables.

3. @EqualsAndHashCode:

Lombok generates equals() and hashCode() method code using variables.

4. @NoArgsConstructor : To generate one default/zero param constrcutor

5. @AllArgsConstructor : To generate all variables as Parameters constrcutor

6. @RequiredArgsConstructor + @NonNull :

To generated selected variables as params constrcutor.

Variable selection is done using @NonNull

7. @Data = @Getter + @Setter + @ToString + @EqualsAndHashCode + @RequiredArgsConstructor

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

--Example#1--

#1. Create one Starter project

Name : SpringBoot2LombokEx

Dependency: Lombok

#2. Write one class

package in.nareshit.raghu.model;

import lombok.EqualsAndHashCode;

import lombok.Getter;

import lombok.Setter;

import lombok.ToString;

//ctrl+shift+O

@Setter

@Getter

@ToString

@EqualsAndHashCode

public class Student {

private Integer sid;

private String sname;

}

#3. Runner class

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Student;

@Component

public class TestLombok implements CommandLineRunner {

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

Student s1 = new Student();

s1.setSid(100);

s1.setSname("ABC");

System.out.println(s1);

//System.out.println(s.getSid() + "-" + s.getSname());

Student s2 = new Student();

s2.setSid(100);

s2.setSname("ABC");

System.out.println(s1==s2); //compare references

System.out.println(s1.equals(s2)); //compare data(if we override)

}

}

\*) equals() is a method defined in java.lang.Object(C). We can override this method

in every class [non-private,non-final, non-static].

\*) by default equals() method internally uses == operator, to compare data override

this method using annotation @EqualsAndHashCode.

Q) If we do not write any constructor in class then which constructor is provided?

A) Default(Zero param) by java compiler if no constructor exist in class.

--Example#2----------

1. Model class

package in.nareshit.raghu.model;

import lombok.AllArgsConstructor;

import lombok.NoArgsConstructor;

import lombok.ToString;

//ctrl+shift+O

@NoArgsConstructor

@AllArgsConstructor

@ToString

public class Student {

private Integer sid;

private String sname;

}

2. Runner class

package in.nareshit.raghu.runner;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import in.nareshit.raghu.model.Student;

@Component

public class TestLombok implements CommandLineRunner {

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

Student s1 = new Student();

Student s2 = new Student(101, "AA");

System.out.println(s1);

System.out.println(s2);

}

}

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