Spring boot – CommandLineRunner interface example

howtodoinjava.com/spring-boot/command-line-runner-interface-example

Spring boot's `CommandLineRunner` interface is used to run a code block only once in application's lifetime – after application is initialized.

How to use CommandLineRunner

You can use `CommandLineRunner` interface in three ways:

1) Using CommandLineRunner as @Component

This one is fairly easy.

```
@Component
public class ApplicationStartupRunner implements CommandLineRunner
{
protected final Log logger = LogFactory.getLog(getClass());
@Override
public void run(String... args) throws Exception {
logger.info( "ApplicationStartupRunner run method Started !!" );
}
}
```

2) Implement CommandLineRunner in @SpringBootApplication

This is also possible. Sample code given below:

```
@SpringBootApplication
public class SpringBootWebApplication extends
SpringBootServletInitializer implements CommandLineRunner {
@Override
protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
return application.sources(SpringBootWebApplication. class );
}
public static void main(String[] args) throws Exception {
SpringApplication.run(SpringBootWebApplication. class , args);
}
@Override
public void run(String... args) throws Exception {
logger.info( "Application Started !!" );
}
}
```

3) Using CommandLineRunner as Bean

You can define a bean in `SpringBootApplication` which return the class that implements `CommandLineRunner` interface.

ApplicationStartupRunner.java

```
public class ApplicationStartupRunner implements CommandLineRunner
{
protected final Log logger = LogFactory.getLog(getClass());
@Override
public void run(String... args) throws Exception {
logger.info( "Application Started !!" );
}
}
```

Register ApplicationStartupRunner bean

```
@SpringBootApplication
public class SpringBootWebApplication extends
SpringBootServletInitializer {
@Override
protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
return application.sources(SpringBootWebApplication. class );
}
public static void main(String[] args) throws Exception {
SpringApplication.run(SpringBootWebApplication. class , args);
}
@Bean
public ApplicationStartupRunner schedulerRunner() {
return new ApplicationStartupRunner();
}
}
```

It is important to note that if any exceptions are thrown inside the run(String... args) method, this will cause the context to close and an application to shut down. So put risky code in try-catch block – ALWAYS.

Using @Order if multiple CommandLineRunner interface implementations

You may have multiple implementations of `CommandLineRunner` interface. By default, spring boot to scan all its `run()` methods and execute it. But if you want to force some ordering in them, use `@Order` annotation.

```
@Order (value= 3 )
@Component
class ApplicationStartupRunnerOne implements CommandLineRunner {
protected final Log logger = LogFactory.getLog(getClass());
@Override
public void run(String... args) throws Exception {
logger.info( "ApplicationStartupRunnerOne run method Started !!" );
}
}
@Order (value= 2 )
@Component
class ApplicationStartupRunnerTwo implements CommandLineRunner {
protected final Log logger = LogFactory.getLog(getClass());
@Override
public void run(String... args) throws Exception {
logger.info( "ApplicationStartupRunnerTwo run method Started !!" );
}
}
```

Verify the logs.

```
2017 - 03 - 08 13 : 55 : 04 - ApplicationStartupRunnerTwo run
method Started !!
2017 - 03 - 08 13 : 55 : 04 - ApplicationStartupRunnerOne run
method Started !!
```

Why use CommandLineRunner interface

- Command line runners are a useful functionality to execute the various types of code that only have to be run once, right after application startup.
- FYI, Spring Batch relies on these runners in order to trigger the execution of the jobs.
- We can use the dependency injection to our advantage in order to wire in whatever dependencies that we need and in whatever way we want – in `run()` method implementation.