11. Spring boot @Profile annotation | How Profiling works in Spring boot

In Previous topic, I have told you about **@ConditionalOnProperty**

- Bean is created Conditional means Bean can be created or can not be.

And based on this @ConditionalOnProperty, I asked below question

You have 2 Application and 1 Common code base, how you will make sure that, BEAN is only created for 1 application, not for other.

And answers were Mix of:

>@ConditionalOnProperty annotation

and

@Profile annotation

Yes, Using both we can achieve this requirement.

But @Profile is technically intended for environment separation rather than application specific bean creation.

Let's deep dive into @Profile annotation to understand it better.

Dev/Local Machine ----------<>devUsername, devPassword **DB**

QA/Stage Machine ----------<>qaUsername, qaPassword **DB**

Live/Production----------<>prodUsername, prodPassword **DB**

This "username", "password" is just one example.

There are so many other configuration, which are different for different environments, like.

-URL AND PORT NUMBER

-CONNECTION TIMEOUT VALUES

-REQUEST TIMEOUT VALUE

-THROTTLE VALUES

-RETRY VALUES ETC.

How to do it?

We put the configuration in "application.properties" file.

But how to handle, different environment configuration.

That's where Profiling comes into the picture.

application.properties

- application-{profile-1}.properties
- application-{profile-2}.properties.....
- application-{profile-n}.properties

```
//application.properties
username = defaultUsername
password=defaultPassword
```

```
//application-live.properties
username = liveUsername
password=livePassword
```

```
//application-qa.properties
username = qaUsername
password=qaPassword
```

```
//application-dev.properties
username = devUsername
password=devPassword
```

```
package com.springProject.SpringCourseProject.component;

import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.stereotype.Component;
```

```
@Component
public class MySqlConnection {

    @Value("${DBUsername}")
    String username;

    @Value("${DBPassword}")
    String password;

    public MySqlConnection(){
        System.out.println("MySqlConnection Is Initialized");
    }

    @PostConstruct
    public void init(){
        System.out.println("Username: "+username+" password: "+password);
    }
}
output
MySqlConnection Is Initialized
Username: defaultUsername password: defaultPassword
```

```
spring.application.name=SpringCourseProject
isOnlineOrder=false
#prefifix.value=havingValue->matcing string
MySQLConnection.enabled=true
DBPassword=defaultPassword
DBUsername = defaultUsername
spring.profiles.active=qa
Output:
The following 1 profile is active: "qa"
MySqlConnection Is Initialized
Username: qaUsername password: qaPassword
```

We can pass the value to this configuation "spring.profiles.active" during application startup itself

Dynamic Usage

actual command to run spring boot application

mvn spring-boot:run -Dspring-boot.run.profiles=qa

>THIS WILL ALWAYS GET'S PRIORITY EVEN THOUGH IF WE ARE SETTING VALUE FOR SPRING.PROFILES.ACTTIVE=QA

-D > means setting of some environments variables at run time

or

```
<profiles>
    <profile>
       <id>local</id>
       <properties>
          <spring-boot-run.profiles>dev</spring-boot-run.profiles>
       </properties>
    </profile>
    <profile>
       <id>production</id>
       <properties>
          <spring-boot-run.profiles>prod</spring-boot-run.profiles>
       </properties>
    </profile>
    <profile>
       <id>stage</id>
       <properties>
          <spring-boot-run.profiles>qa</spring-boot-run.profiles>
       </properties>
    </profile>
</profiles>
```

mvn spring-boot:run -Pproduction

so, now we know, what's profiling is. let's see **what is @Profile annotation.**

Using @Profile annotation, we can tell spring boot, to create bean only when particular profile is set.

```
package com.springProject.SpringCourseProject.component;

import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;

@Component
@Profile("dev")
public class NoSqlConnection {

    public NoSqlConnection(){
        System.out.println("NoSqlConnection Is Initialized");
    }

    @Value("${DBUsername}")
    String username;
```

```
    @Value("${DBPassword}")
    String password;


    @PostConstruct
    public void init(){
        System.out.println("Username: "+username+" password: "+password);
    }
}
```

```
package com.springProject.SpringCourseProject.component;

import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;

@Component
//@ConditionalOnProperty(prefix = "MySQLConnection",value = "enabled",havingValue =
"true",matchIfMissing = false)
@Profile("prod")
public class MySqlConnection {

    @Value("${DBUsername}")
    String username;

    @Value("${DBPassword}")
    String password;

    public MySqlConnection(){
        System.out.println("MySqlConnection Is Initialized");
    }

    @PostConstruct
    public void init(){
        System.out.println("Username: "+username+" password: "+password);
    }
}
```

output

The following 1 profile is active: "qa"

None of these bean will be created.

mvn spring-boot:run -Dspring-boot.run.profiles=prod

Bean initialization for MySqlConnection.

We can also set multiple profiles at a time.

```
DBPassword=defaultPassword
DBUsername = defaultUsername
spring.profiles.active=prod,qa
```

Application.properties profile for actual application running on. Is going to pick last one which is QA

But in context of code with @Profile annotation BEAN will be created for Both for **MySQL** as well as **NOSQL**

OUTPUT

The following 2 profiles are active: "prod", "qa"

DBConnection Initialized

MySqlConnection Is Initialized

Username: qaUsername password: qaPassword

NoSqlConnection Is Initialized

Username: qaUsername password: qaPassword

DBConnection BEAN create with below dependencies

Now let's Come back to our question which is ask in begining

YOU HAVE 2 APPLICATION AND 1 COMMON CODE BASE, HOW YOU WILL MAKE SURE THAT, BEAN IS ONLY CREATED FOR 1 APPLICATION, NOT FOR OTHER.

COMMON CODE BASE

```
package com.springProject.SpringCourseProject.component;

import jakarta.annotation.PostConstruct;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.context.annotation.Profile;
import org.springframework.stereotype.Component;

@Component
```

```
@Profile("app1")
public class NoSqlConnection {

    public NoSqlConnection(){
        System.out.println("NoSqlConnection Is Initialized");
    }

    @Value("${DBUsername}")
    String username;

    @Value("${DBPassword}")
    String password;


    @PostConstruct
    public void init(){
        System.out.println("Username: "+username+" password: "+password);
    }
}
```

Application1

application.properties

```
DBPassword=defaultPassword
DBUsername = defaultUsername
spring.profiles.active=app1
```

Application2

application.properties

```
DBPassword=defaultPassword
DBUsername = defaultUsername
spring.profiles.active=app2
```

For only aap1 BEAN will be created.

NOT A GOOD APPOROACH

Use @ConditionalOnProperty