COURSEWARE

Profiles

Contents

Overview

Spring profiles provide two major functions: allowing for multiple project configurations through multiple property files, and allowing us to control which beans are created.

Property files

Spring profiles are set up via property files; like those seen in the **Configuration** module.

Rather than using a single `application.properties/yaml` file, we can create files for specific profiles by following the `application-<profile-name>.properties` format.

The below shows example properties files for a `development` profile (**`application-dev.properties`**):

```
spring.h2.console.enabled=true
server.port=8081

spring.jpa.hibernate.ddl-auto=create-drop
```

And an example properties file for production (**`application-prod.properties`**):

```
server.port = 8181
spring.data.rest.base-path=/api

spring.jpa.hibernate.ddl-auto=validate

spring.datasource.url=jdbc:mysql://33.222.178.42:3306/ducks
spring.datasource.username=root
spring.datasource.password=notpassword

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
spring.datasource.initialization-mode=always
```

Setting the active profile

The active profile can be set in `application.properties` using `spring.profiles.active`:

```
spring.profiles.active=dev
```

Alternatively, the active profile can be set using `@ActiveProfiles` in **test classes**:

```
@SpringBootTest
@ActiveProfiles("test")
public class ExampleTest {
    //
}
```

Making components profile-specific

To set when a component will be created, we use the `@Profile` annotation.

Here, we've set our `AppConfig` file to be loaded in when accessed by both the `dev` and `prod` profiles:

```
@Configuration
@Profile({"dev", "prod"})
public class AppConfig {

    @Bean
    public ModelMapper mapper() {
        return new ModelMapper();
    }
}
```

`@Profile` can also be used to set when *not* to create a component.

Here, we've ensured that this configuration file is never used with the `test` profile:

```
@Configuration
@Profile("!test")
public class AppConfig {

    @Bean
    public ModelMapper mapper() {
        return new ModelMapper();
    }
}
```

Tutorial

There is no tutorial for this module.

Exercises

- Create `dev` and `prod` profiles for your project.
- Make the `dev` profile point at an *h2 database*.
- Make the `prod` profile connect to a MySQL database on GCP - if you're unsure of how to do this, a tutorial can be found in the GCP MySQL Database module.)