Implementing Spring Profiles for Data Loading

Spring Feature Description

Spring Profiles allow for conditional bean registration based on the active profile, enabling different configurations for different environments (e.g., development, testing, production). In this implementation, we will configure the application to read flight data from either `flights.csv` or `flights_small.csv` based on the active profile.

Why Do We Need It in Flight Application?

The ability to switch data sources based on the active profile enhances the flexibility and adaptability of the application. During development or testing, using a smaller dataset (`flights_small.csv`) can improve performance and simplify debugging. In production, the application can utilize a more comprehensive dataset (`flights.csv`).

Refactoring Application

We will introduce two profiles: `dev` for development and `prod` for production. Each profile will load the appropriate CSV file when initializing the `CSVDataLoader`.

The Updated Code

- **Define application.properties for Profiles**

Create an `application.properties` file with the following entries:

```
spring.profiles.active=dev  # Set the active profile (dev or prod)
```

Create Profile-Specific Properties Files

Create two property files: `application-dev.properties` and `application-prod.properties`.

application-dev.properties:

```
csv.file.path=flights_small.csv
```

application-prod.properties:

```
csv.file.path=flights.csv
```

`CSVDataLoader` is reading data from a CSV file based on the active profile. Test it by switching between `dev` and `prod` profiles in `application.properties`.

Implementing Profile-Based Bean Activation

We will demonstrate profile-based bean activation by creating a `DatabaseConnection` class that loads different database configurations based on the active profile. This will allow us to switch between `dev` and `prod` profiles to use different database connections. Then, we will update the `DBDataLoader` class to use the `DatabaseConnection` bean based on the active profile. This will showcase how to switch between different data loading strategies based on the active profile.

- **Define DatabaseConnection Class**

Create a simple `DatabaseConnection` class to demonstrate profile-based bean activation:

```
public class DatabaseConnection {

    private String url;
    private String username;
    private String password;

    public DatabaseConnection(String url, String username, String password) {
        this.url = url;
        this.username = username;
        this.password = password;
    }

    // Getters and setters
}
```

Now you should be able to switch between the `dev` and `prod` profiles to load different CSV files.

- **Define Beans Activated depending on Profile**

Add beans in a configuration class that should only be activated under specific profiles:

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.Profile;

@Configuration
public class AppConfig {

    @Bean
    @Profile("prod")
    public DatabaseConnection prodDatabaseConnection() {
        return new DatabaseConnection("jdbc:prod-db-url", "prod-user", "prod-password
");
    }
```

```
    @Bean
    @Profile("dev")
    public DatabaseConnection devDatabaseConnection() {
        return new DatabaseConnection("jdbc:dev-db-url", "dev-user", "dev-password");
    }
}
```

- **Update DBDataLoader to use DatabaseConnection**

Modify the `DBDataLoader` class to use the `DatabaseConnection` bean based on the active profile:

```
    private final DataService dataService;
    private final DatabaseConnection databaseConnection;

    public DBDataLoader(DataService dataService, DatabaseConnection
databaseConnection) {
        this.dataService = dataService;
        this.databaseConnection = databaseConnection;
    }

    @Override
    public void loadData() {

        System.out.println("Loading data from database...");
        System.out.println("Using database connection: " + databaseConnection.getUrl(
));

    }
```

Now you should be able to change profile in `application.properties` to switch between `dev` and `prod` configurations:

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

To see how it works with DatabaseConnection, use @Qualifier annotation to specify which bean to use in FlightsApplication:

```
    @Bean
    CommandLineRunner run(DataService dataService, @Qualifier("dbDataLoader")
DataLoader dataLoader) {
        return args -> {
            dataLoader.loadData();
            // Existing data loading logic
        };
    }
```

Changing Profiles with Command Line

You can change the active profile when starting your Spring Boot application using the command line. For example:

```
java -jar your-application.jar --spring.profiles.active=prod
```

This command sets the active profile to `prod`, which will load the configuration specified in `application-prod.properties`. Similarly, you can set it to `dev` as follows:

```
java -jar your-application.jar --spring.profiles.active=dev
```

Switching Between CSV and Database DataLoader with Profiles

By using Spring Profiles, you can easily switch between different data loading strategies based on the active profile. This flexibility allows you to adapt the application's behavior to different environments without changing the code.

This is how you can implement Spring Profiles for data loading in your Flight Application:

```
@Component
@Qualifier("csvDataLoader")
@Profile("csv")
@Primary
public class CSVDataLoader implements DataLoader {
    // Existing CSV data loading logic
}
```

```
@Component
@Qualifier("dbDataLoader")
@Profile("db")
public class DBDataLoader implements DataLoader {
    // Existing database data loading logic
}
```

Now, you can activate the `csv` or `db` profile to switch between CSV and database data loading strategies. This can be used in combination with other profiles to configure different aspects of your application based on the active profile.

```
spring.profiles.active=csv,dev
```

Or this way you can activate database in production:

```
spring.profiles.active=db,prod
```

Try to run different configurations and observe how the application behaves based on the active profiles.

This demonstrates the flexibility and power of Spring in managing application configurations based.

Benefits of Using Spring Profiles

- **Environment-Specific Configuration**: Easily switch between different configurations for development and production environments.
- **Improved Flexibility**: Tailor the application's behavior based on the active profile, enabling optimized performance and resource usage.
- **Simplified Management**: Manage properties for different environments in a clean and organized manner without altering code.

Further Advices

Ensure that the profiles are properly activated in different deployment scenarios (e.g., through environment variables or command-line arguments). Regularly review and update profile-specific configurations to align with evolving application requirements.