Conflict

In a Spring Boot application, when both `application.yml` and `application.properties` files are present in the classpath, Spring Boot will prioritize the `application.properties` file over the `application.yml` file.

The properties defined in the `application.properties` file will take precedence over those defined in the `application.yml` file.

This precedence is due to the default order of property source locations in Spring Boot, where `.properties` files are considered before `.yml` files.

If conflicting properties are defined in both files, the value defined in the `application.properties` file will be used. However, you can still override properties defined in `application.properties` with command-line arguments or environment variables if needed.

Here's the default order of property sources in Spring Boot:

1. Command line arguments (`--property=value`)
2. Java System properties (`System.getProperties()`)
3. OS environment variables
4. `application.properties` in the classpath
5. `application.yml` in the classpath

If you need to ensure that properties defined in `application.yml` take precedence over those defined in `application.properties`, you can either remove or rename the `application.properties` file, or explicitly specify the order of property sources using the `spring.config.location` property in your application configuration.

Certainly! In a Spring Boot application, properties are loaded from various sources, and they have a defined order of precedence. Here's the typical order of precedence for property sources:

1. **Command-Line Arguments**: Properties provided as command-line arguments take the highest precedence. You can specify properties using the `--property=value` syntax when starting the application.

2. **Java System Properties**: Properties set as Java system properties using the `-D` option when launching the JVM. They override properties defined in configuration files.

3. **OS Environment Variables**: Environment variables set in the operating system environment. Spring Boot automatically converts environment variables to properties.

4. **Profile-Specific Properties Files**: Spring Boot supports profile-specific properties files. Properties defined in these files override properties defined in general properties files. The order of precedence for profile-specific properties files is determined by the active profiles.

5. **General Properties Files (application.properties/application.yml)**: Properties defined in general properties files (`application.properties` or `application.yml`) located in the classpath. These files are used as the default configuration for the application.

6. **Custom Property Sources**: Custom property sources that you define in your application. These can be implemented using the `PropertySource` interface or by extending the `PropertySourceLoader` class.

7. **Default Properties**: Spring Boot provides default properties that are applied if no other properties are specified. These defaults are built into Spring Boot and can be overridden by explicitly setting properties in the configuration.

It's important to note that properties defined in sources with higher precedence override properties defined in sources with lower precedence. This allows for flexibility in configuring Spring Boot applications based on different environments and deployment scenarios.