Spring config location

The concept of location groups in Spring Boot is used to organize profile-specific configuration files, especially in scenarios where you have a complex setup with multiple locations for configuration files. By grouping these locations, you can provide hints to Spring Boot about how to prioritize and organize them during property resolution.

Here's a breakdown of how location groups work and how you can use them effectively:

1. **Definition of Location Groups**: A location group is a collection of locations that are all considered at the same level. This means that properties defined in any file within a location group are considered together during property resolution. Location groups are defined using semicolons (`;`) to separate individual locations.

2. **Grouping Classpath and External Locations**: For example, you might want to group all classpath locations together and then group all external locations separately. This allows you to specify the order in which Spring Boot should consider these locations when resolving properties.

3. **Using Location Groups with Profile-Specific Files**: When you have profile-specific configuration files, you can provide hints to Spring Boot by organizing these files into location groups. This helps Spring Boot understand how to group and prioritize properties based on the active profiles.

4. **Example Configuration**: Here's an example configuration demonstrating how you can use location groups with profile-specific files:

   ```properties
   spring.config.location=classpath:/config/,classpath:/custom-config/;file:/external-config/
   ```

   In this example:
   - `classpath:/config/` and `classpath:/custom-config/` are grouped together as classpath locations.
   - `file:/external-config/` is considered as an external location.
   - The semicolon `;` separates the two location groups.

5. **Providing Further Hints**: If you have a more complex setup, you may need to provide additional hints to Spring Boot to ensure proper grouping and prioritization of configuration files. This could involve specifying multiple location groups or adjusting the order of locations within each group.

By effectively using location groups in your Spring Boot application, you can ensure that profile-specific configuration files are organized and prioritized correctly, leading to more predictable property resolution behavior.

The `spring.config.location` property in a Spring Boot application allows you to specify additional locations to search for application properties files. When you provide a value for `spring.config.location`, Spring Boot will look for application properties not only in the default locations but also in the locations you specify.

By default, Spring Boot searches for `application.properties` or `application.yml` files in the following locations:

1. The classpath root.
2. The `config` directory in the classpath.
3. The current directory.

If you provide a value for `spring.config.location`, it will override these defaults, and Spring Boot will search for properties in the locations you specify.

For example, if you start your Spring Boot application with the following command:

```
java -jar myapp.jar --spring.config.location=classpath:/custom-location/
```

Spring Boot will look for `application.properties` or `application.yml` files in the `custom-location` directory within the classpath, in addition to the default locations.

You can provide multiple locations separated by commas:

```
java -jar myapp.jar --spring.config.location=classpath:/custom-location/,file:/path/to/external/config/
```

This tells Spring Boot to search for properties in both the `custom-location` directory within the classpath and the `/path/to/external/config/` directory in the filesystem.