Spring Boot

Advanced Configuration

Agenda

- Inject Random Configuration Values
- Accessing Command line Properties
- Application Property Files
- Use Profiles
- Run code at startup
- Automatic Restarts
- LiveReload !!
- Summary
- Questions

Injecting Random Configuration values

- The RandomValuePropertySource is useful for injecting random values
  - for example, into secrets or test cases.
- It can produce integers, longs, uuids, or strings

```
my.secret=${random.value}
my.number=${random.int}
my.bignumber=${random.long}
my.uuid=${random.uuid}
my.number.less.than.ten=${random.int(10)}
my.number.in.range=${random.int[1024,65536]}
```

Example Usage:

```
@Value("${my.number}")
int mynumber;
```

Accessing Command Line Properties

- By default, SpringApplication converts any command line option arguments to a property and adds them to the Spring Environment.
- Command Line arguments start with --, such as

  --server.port=9000

- Example: java -jar myapp.jar --server.port=9000

- command line properties always take precedence over other property sources.

- you can disable them by using *SpringApplication.setAddCommandLineProperties(false)* in your Boot Application's Main Class.

Application Property Files

- SpringApplication loads properties from application.properties files in the following locations and adds them to the Spring Environment:
  - A /config subdirectory of the current directory
  - The current directory
  - A classpath /config package
  - The classpath root
- The list is ordered by precedence (properties defined in locations higher in the list override those defined in lower locations).

Application Property Files

- Override the defaults using
  - spring.config.name
  - spring.config.location

```
$ java -jar myproject.jar --spring.config.name=myproject
$ java -jar myproject.jar \
 -spring.config.location=classpath:/default.properties, \
       classpath:/override.properties
```

Profile-specific Properties

- Profile specific properties files are defined as per the given template.

application-{profile-name}.properties

Example:

application-dev.properties *For dev profile*

application-production.properties *For production profile*

application-test.properties *For test profile and so on..*

Profiles

Spring Profiles

- Help segregate parts of the application configuration.
- Make it be available only in certain environments.
- Any @Component or @Configuration can be marked with **@Profile** to limit when it is loaded

```
@Configuration
@Profile("production") public class
ProductionConfiguration { // ...

}
```

The class will be loaded only in Active Profile “production”

How to set Active Profile

- Use a spring.profiles.active Environment property to specify which profiles are active
- Active Profile can be set in various ways.

```
spring.profiles.active=dev,hsqldb          (application.properties)

--spring.profiles.active=dev,hsqldb        (Command line parameter)
```

Programmatically Setting Profiles

- You can programmatically set active profiles by calling `SpringApplication.setAdditionalProfiles(…)` before your application runs.