

Property Source

jim stafford

Fall 2024 v2024-08-03: Built: 2024-11-19 21:31 EST

Table of Contents

| | |
|---|----|
| 1. Introduction | 1 |
| 1.1. Goals | 1 |
| 1.2. Objectives | 1 |
| 2. Property File Source(s) | 2 |
| 2.1. Property File Source Example | 2 |
| 2.2. Example Property File Contents | 3 |
| 2.3. Non-existent Path | 4 |
| 2.4. Path not Ending with Slash ("/") | 5 |
| 2.5. Alternate File Examples | 5 |
| 2.6. Series of files | 6 |
| 3. @PropertySource Annotation | 7 |
| 4. Profiles | 9 |
| 4.1. Default Profile | 10 |
| 4.2. Specific Active Profile | 11 |
| 4.3. Multiple Active Profiles | 11 |
| 4.4. No Associated Profile | 12 |
| 5. Property Placeholders | 13 |
| 5.1. Placeholder Demonstration | 13 |
| 5.2. Placeholder Property Files | 14 |
| 5.3. Placeholder Value Defined Internally | 14 |
| 5.4. Placeholder Value Defined in Profile | 14 |
| 5.5. Multiple Active Profiles | 15 |
| 5.6. Mixing Names, Profiles, and Location | 15 |
| 6. Other Common Property Sources | 17 |
| 6.1. Property Source | 17 |
| 6.2. application.properties | 18 |
| 6.3. Profiles | 18 |
| 6.4. Environment Variables | 18 |
| 6.5. Java System Properties | 19 |
| 6.6. spring.application.json | 20 |
| 6.7. Command Line Arguments | 20 |
| 6.8. @SpringBootTest.properties | 21 |
| 7. Summary | 22 |

Chapter 1. Introduction

In the previous section we defined a value injection into an attribute of a Spring Bean class and defined a few ways to inject a value on an individual basis. Next, we will set up ways to specify entire collection of property values through files.

1.1. Goals

The student will learn:

- to supply groups of properties using files
- to configure a Spring Boot application using property files
- to flexibly configure and control configurations applied

1.2. Objectives

At the conclusion of this lecture and related exercises, the student will be able to:

1. configure a Spring Boot application using a property file
2. specify a property file for a basename
3. specify a property file packaged within a JAR file
4. specify a property file located on the file system
5. specify a both `properties` and `YAML` property file sources
6. specify multiple files to derive an injected property from
7. specify properties based on an active profile
8. specify properties based on placeholder values
9. specify property overrides using different external source options

Chapter 2. Property File Source(s)

Spring Boot uses three key properties when looking for configuration files (Ref: docs.spring.io):

1. `spring.config.name` — one or more base names separated by commas. The default is `application` and the suffixes searched for are `.properties` and `.yml` (or `.yaml`)
2. `spring.profiles.active` — one or more profile names separated by commas used in this context to identify which form of the base name to use. The default is `default` and this value is located at the end of the base filename separated by a dash (-; e.g., `application-default`)
3. `spring.config.location` — one or more directories/packages to search for configuration files or explicit references to specific files. The default is:
 - a. `file:config/` - within a `config` directory in the current directory
 - b. `file:./` - within the current directory
 - c. `classpath:/config/` - within a `config` package in the classpath
 - d. `classpath:/` — within the root package of the classpath

Names are primarily used to identify the base name of the application (e.g., `application` or `myapp`) or of distinct areas (e.g., `database`, `security`). Profiles are primarily used to supply variants of property values. Location is primarily used to identify the search paths to look for configuration files but can be used to override names and profiles when a complete file path is supplied.

2.1. Property File Source Example

In this initial example I will demonstrate `spring.config.name` and `spring.config.location` and use a single value injection similar to previous examples.

Value Injection Target

```
//AppCommand.java
...
@Value("${app.audience}")
private String audience;
...
```

However, the source of the property value will not come from the command line. It will come from one of the following property and/or YAML files in our module.

Source Tree

```
src
|-- main
|   |-- java
|   |   |-- ...
|   |-- resources
|       |-- alternate_source.properties
|       |-- alternate_source.yml
```

```
|-- application.properties
|-- property_source.properties
```

JAR File

```
$ jar tf target/appconfig-property-source-example-*-SNAPSHOT-bootexec.jar | \
  egrep 'classes.*(properties|yml)'
BOOT-INF/classes/alternate_source.properties
BOOT-INF/classes/alternate_source.yml
BOOT-INF/classes/property_source.properties
BOOT-INF/classes/application.properties
```

2.2. Example Property File Contents

The four files each declare the same property `app.audience` but with a different value. Spring Boot primarily supports the two file types shown (`properties` and `YAML`). There is [some support for JSON](#) and `XML` is primarily used to define configurations.

The first three below are in `properties` format.

```
#property_source.properties
app.audience=Property Source value
```

```
#alternate_source.properties
app.audience=alternate source property file
```

```
#application.properties
app.audience=application.properties value
```

This last file is in `YAML` format.

```
#alternate_source.yml
app:
  audience: alternate source YAML file
```

That means the following—which will load the `application.(properties|yml)` file from one of the four locations ...

```
$ java -jar target/appconfig-property-source-example-*-SNAPSHOT-bootexec.jar
...
Application @Bean says Hey application.properties value
```

can also be completed with

```
$ java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.location="classpath:" \  
... \  
Application @Bean says Hey application.properties value
```

```
$ java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.location="file:src/main/resources/" \  
... \  
Application @Bean says Hey application.properties value
```

```
$ java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.location="file:src/main/resources/application.properties" \  
... \  
Application @Bean says Hey application.properties value
```

```
$ cp src/main/resources/application.properties /tmp/xyz.properties \  
$ java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.name=xyz --spring.config.location="file:/tmp/" \  
... \  
Application @Bean says Hey application.properties value
```

2.3. Non-existent Path

If you supply a non-existent path, Spring will report that as an error.

Location Directories Must Exist

```
java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
--spring.config.location="file:src/main/resources/,file:src/main/resources/does_not_exit/" \  
[main] ERROR org.springframework.boot.diagnostics.LoggingFailureAnalysisReporter -- \  
***** \  
APPLICATION FAILED TO START \  
***** \  
Description: \  
Config data location 'file:src/main/resources/does_not_exit/' does not exist \  
Action: \  
Check that the value 'file:src/main/resources/does_not_exit/' is correct, or prefix it
```

```
with 'optional:'
```

You can mark the location with `optional:` for cases where it is legitimate for the location not to exist.

Making Location Directory Optional

```
java -jar target/appconfig-property-source-example-*-SNAPSHOT-bootexec.jar \  
  
--spring.config.location="file:src/main/resources/,optional:file:src/main/resources/do  
es_not_exit/"
```

2.4. Path not Ending with Slash ("/")

If you supply a path not ending with a slash ("/"), Spring will also report an error.

```
java -jar target/appconfig-property-source-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.location="file:src/main/resources"  
...  
14:28:23.544 [main] ERROR org.springframework.boot.SpringApplication - Application run  
failed  
java.lang.IllegalStateException: Unable to load config data from  
'file:src/main/resources'  
...  
Caused by: java.lang.IllegalStateException: File extension is not known to any  
PropertySourceLoader. If the location is meant to reference a directory, it must end  
in '/' or File.separator
```

2.5. Alternate File Examples

We can switch to a different set of configuration files by changing the `spring.config.name` or `spring.config.location` so that ...

```
#property_source.properties  
app.audience=Property Source value
```

```
#alternate_source.properties  
app.audience=alternate source property file
```

```
#alternate_source.yml  
app:  
  audience: alternate source YAML file
```

can be used to produce

```
$ java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.name=property_source \  
  ...  
Application @Bean says Hey Property Source value
```

```
$ java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.name=alternate_source \  
  ...  
Application @Bean says Hey alternate source property file
```

```
$ java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.location="classpath:alternate_source.properties,classpath:alternate_source.yml" \  
  ...  
Application @Bean says Hey alternate source YAML file
```

2.6. Series of files

```
#property_source.properties  
app.audience=Property Source value
```

```
#alternate_source.properties  
app.audience=alternate source property file
```

The default priority is last specified.

```
$ java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.name="property_source,alternate_source" \  
  ...  
Application @Bean says Hey alternate source property file
```

```
$ java -jar target/appconfig-propertysource-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.name="alternate_source,property_source" \  
  ...  
Application @Bean says Hey Property Source value
```


Chapter 3. @PropertySource Annotation

We can define a property to explicitly be loaded using a Spring-provided `@PropertySource` annotation. This annotation can be used on any class that is used as a `@Configuration`, so I will add that to the main application. However, because we are still working with a very simplistic, single property example—I have started a sibling example that only has a single property file so that no priority/overrides from application.properties will occur.

Example Source Tree

```
|-- pom.xml
|-- src
    |-- main
        |-- java
            |-- info
                |-- ejava
                    |-- examples
                        |-- app
                            |-- config
                                |-- propertysource
                                    |-- annotation
                                        |-- AppComponent.java
                                        |-- PropertySourceApp.java
                            |-- resources
                                |-- property_source.properties
```

```
#property_source.properties
app.audience=Property Source value
```

Annotation Reference

```
...
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.PropertySource;

@SpringBootApplication
@PropertySource("classpath:property_source.properties") ①
public class PropertySourceApp {
    ...
}
```

① An explicit reference to the properties file is placed within the annotation on the `@Configuration` class

When we now execute our JAR, we get the contents of the property file.

```
java -jar target/appconfig-propertysource-annotation-example-*-SNAPSHOT-bootexec.jar
...
```

We will cover alternate property sources and their priority later within this lecture, `@PropertySource` ends up being one of the lowest priority sources. This permits the application to package a set of factory defaults and optionally allow many of the other sources to override.



Place Factory-Default Properties in @PropertySource

`@PropertySource` references make for a convenient location for components to package factory-supplied, low-priority defaults that can be easily overridden.

Chapter 4. Profiles

In addition to `spring.config.name` and `spring.config.location`, there is a third configuration property — `spring.profiles.active` — that Spring uses when configuring an application. Profiles are identified by

`-(profileName)` at the end of the base filename (e.g., `application-site1.properties`, `myapp-site1.properties`)

I am going to create a new example to help explain this.

Profile Example

```
|-- pom.xml
|-- src
|   |-- main
|       |-- java
|           |-- info
|               |-- ejava
|                   |-- examples
|                       |-- app
|                           |-- config
|                               |-- propertysource
|                                   |-- profiles
|                                       |-- AppCommand.java
|                                       |-- PropertySourceApp.java
|-- resources
    |-- application-default.properties
    |-- application-site1.properties
    |-- application-site2.properties
    |-- application.properties
```

The example uses the default `spring.config.name` of `application` and supplies four property files.

- each of the property files supplies a common property of `app.commonProperty` to help demonstrate priority
- each of the property files supplies a unique property to help identify whether the file was used

```
#application.properties
app.commonProperty=commonProperty from application.properties
app.appProperty=appProperty from application.properties
```

```
#application-default.properties
app.commonProperty=commonProperty from application-default.properties
app.defaultProperty=defaultProperty from application-default.properties
```

```
#application-site1.properties
```

```
app.commonProperty=commonProperty from application-site1.properties
app.site1Property=site1Property from application-site1.properties
```

```
#application-site2.properties
app.commonProperty=commonProperty from application-site2.properties
app.site2Property=site2Property from application-site2.properties
```

The component class defines an attribute for each of the available properties and defines a default value to identify when they have not been supplied.

```
@Component
public class AppCommand implements CommandLineRunner {
    @Value("${app.commonProperty:not supplied}")
    private String commonProperty;
    @Value("${app.appProperty:not supplied}")
    private String appProperty;
    @Value("${app.defaultProperty:not supplied}")
    private String defaultProperty;
    @Value("${app.site1Property:not supplied}")
    private String site1Property;
    @Value("${app.site2Property:not supplied}")
    private String site2Property;
```



In all cases (except when using an alternate `spring.config.name`), we will get the `application.properties` loaded. However, it is used at a lower priority than all other sources.

4.1. Default Profile

If we run the program with no profiles active, we enact the `default` profile. `site1` and `site2` profiles are not loaded.

```
$ java -jar target/appconfig-property-source-profile-example-*-SNAPSHOT-bootexec.jar
...
commonProperty=commonProperty from application-default.properties ①
appProperty=appProperty from application.properties ②
defaultProperty=defaultProperty from application-default.properties ③
site1Property=not supplied ④
site2Property=not supplied
```

① `commonProperty` was set to the value from `default` profile

② `application.properties` was loaded

③ the `default` profile was loaded

④ `site1` and `site2` profiles were not loaded

4.2. Specific Active Profile

If we activate a specific profile (`site1`) the associated file is loaded and the alternate profiles—including `default`—are not loaded.

```
$ java -jar target/appconfig-property-source-profile-example-*-SNAPSHOT-bootexec.jar \
  --spring.profiles.active=site1
...
commonProperty=commonProperty from application-site1.properties ①
appProperty=appProperty from application.properties ②
defaultProperty=not supplied ③
site1Property=site1Property from application-site1.properties ④
site2Property=not supplied ③
```

① `commonProperty` was set to the value from `site1` profile

② `application.properties` was loaded

③ `default` and `site2` profiles were not loaded

④ the `site1` profile was loaded

4.3. Multiple Active Profiles

We can activate multiple profiles at the same time. If they define overlapping properties, the later one specified takes priority.

```
$ java -jar target/appconfig-property-source-profile-example-*-SNAPSHOT-bootexec.jar \
  --spring.profiles.active=site1,site2 ①
...
commonProperty=commonProperty from application-site2.properties ①
appProperty=appProperty from application.properties ②
defaultProperty=not supplied ③
site1Property=site1Property from application-site1.properties ④
site2Property=site2Property from application-site2.properties ④

$ java -jar target/appconfig-property-source-profile-example-*-SNAPSHOT-bootexec.jar \
  --spring.profiles.active=site2,site1 ①
...
commonProperty=commonProperty from application-site1.properties ①
appProperty=appProperty from application.properties ②
defaultProperty=not supplied ③
site1Property=site1Property from application-site1.properties ④
site2Property=site2Property from application-site2.properties ④
```

① `commonProperty` was set to the value from last specified profile

② `application.properties` was loaded

③ the `default` profile was not loaded

④ `site1` and `site2` profiles were loaded

4.4. No Associated Profile

If there are no associated profiles with a given `spring.config.name`, then none will be loaded.

```
$ java -jar target/appconfig-property-source-profile-example-*-SNAPSHOT-bootexec.jar \  
  --spring.config.name=BOGUS --spring.profiles.active=site1 ①  
...  
commonProperty=not supplied ①  
appProperty=not supplied  
defaultProperty=not supplied  
site1Property=not supplied  
site2Property=not supplied
```

① No profiles were loaded for `spring.config.name` `BOGUS`

Chapter 5. Property Placeholders

We have the ability to build property values using a placeholder that will come from elsewhere. Consider the following example where there is a common pattern to a specific set of URLs that change based on a base URL value.

- `(config_name).properties` would be the candidate to host the following definition

```
security.authn=${security.service.url}/authentications?user=:user
security.authz=${security.service.url}/authorizations/roles?user=:user
```

- profiles would host the specific value for the placeholder
 - `(config_name)-(profileA).properties`

```
security.service.url=http://localhost:8080
```

- `(config_name)-(profileB).properties`

```
security.service.url=https://acme.com
```

- the default value for the placeholder can be declared in the same property file that uses it

```
security.service.url=https://acme.com
security.authn=${security.service.url}/authentications?user=:user
security.authz=${security.service.url}/authorizations/roles?user=:user
```

5.1. Placeholder Demonstration

To demonstrate this further, I am going to add three additional property files to the previous example.

```
\-- src
  \-- main
    ...
    \-- resources
      |-- ...
      |-- myapp-site1.properties
      |-- myapp-site2.properties
      \-- myapp.properties
```

5.2. Placeholder Property Files

```
# myapp.properties
app.commonProperty=commonProperty from myapp.properties ②
app.appProperty="${app.commonProperty}" used by myapp.property ①
```

- ① defines a placeholder for another property
- ② defines a default value for the placeholder within this file



Only the `${}` characters and property name are specific to property placeholders. Quotes (`"`) within this property value are part of this example and not anything specific to property placeholders in general.

```
# myapp-site1.properties
app.commonProperty=commonProperty from myapp-site1.properties ①
app.site1Property=site1Property from myapp-site1.properties
```

- ① defines a value for the placeholder

```
# myapp-site2.properties
app.commonProperty=commonProperty from myapp-site2.properties ①
app.site2Property=site2Property from myapp-site2.properties
```

- ① defines a value for the placeholder

5.3. Placeholder Value Defined Internally

Without any profiles activated, we obtain a value for the placeholder from within `myapp.properties`.

```
$ java -jar target/appconfig-property-source-profile-example-*-SNAPSHOT-bootexec.jar \
  --spring.config.name=myapp
...
commonProperty=commonProperty from myapp.properties
appProperty="commonProperty from myapp.properties" used by myapp.property ①
defaultProperty=not supplied
site1Property=not supplied
site2Property=not supplied
```

- ① placeholder value coming from default value defined in same `myapp.properties`

5.4. Placeholder Value Defined in Profile

Activating the `site1` profile causes the placeholder value to get defined by `myapp-site1.properties`.


```
$ java -jar target/appconfig-property-source-profile-example-*-SNAPSHOT-bootexec.jar \
  --spring.config.name=myapp --spring.profiles.active=site1
...
commonProperty=commonProperty from myapp-site1.properties
appProperty="commonProperty from myapp-site1.properties" used by myapp.property ①
defaultProperty=not supplied
site1Property=site1Property from myapp-site1.properties
site2Property=not supplied
```

① placeholder value coming from value defined in `myapp-site1.properties`

5.5. Multiple Active Profiles

Multiple profiles can be activated. By default — the last profile specified has the highest priority.

```
$ java -jar target/appconfig-property-source-profile-example-*-SNAPSHOT-bootexec.jar \
  --spring.config.name=myapp --spring.profiles.active=site1,site2
...
commonProperty=commonProperty from myapp-site2.properties
appProperty="commonProperty from myapp-site2.properties" used by myapp.property ①
defaultProperty=not supplied
site1Property=site1Property from myapp-site1.properties
site2Property=site2Property from myapp-site2.properties
```

① placeholder value coming from value defined in last profile — `myapp-site2.properties`

5.6. Mixing Names, Profiles, and Location

Name, profile, and location constructs can play well together as long as location only references a directory path and not a specific file. In the example below, we are defining a non-default name, a non-default profile, and a non-default location to search for the property files.

```
$ java -jar target/appconfig-property-source-profile-example-*-SNAPSHOT-bootexec.jar \
  --spring.config.name=myapp \
  --spring.profiles.active=site1 \
  --spring.config.location="file:src/main/resources/"
...
commonProperty=commonProperty from myapp-site1.properties
appProperty="commonProperty from myapp-site1.properties" used by myapp.property
defaultProperty=not supplied
site1Property=site1Property from myapp-site1.properties
site2Property=not supplied
```

The above example located the following property files in the filesystem (not classpath)

- `src/main/resources/myapp.properties`

- `src/main/resources/myapp-site1.properties`

Chapter 6. Other Common Property Sources

Property files and profiles are, by far, the primary way to configure applications in bulk. However, it is helpful to know other ways to supply or override properties external to the application. One specific example is in deployment platforms where you do not control the command line and must provide some key configuration via environment variables or system properties.

Spring.io lists about a dozen sources of [other property sources](#) in priority order. Use their site for the full list. I won't go through all of them, but will cover the ones that have been the most common and helpful for me in low-to-high Spring priority order.

Each example will inject value into a component property that will be printed.

Component with app.audience @Value Injection

```
@Value("${app.audience}")  
private String audience;
```

6.1. Property Source

`@PropertySource` is one of the lowest priority sources. In the following example, I am supplying a custom property file and referencing it from a `@Configuration`. The reference will use a `classpath:` reference to use the file from the module JAR.

@Configuration in the Scan Path References the Property Source

```
@Configuration  
@PropertySource(name="propertySource",  
                value =  
                "classpath:info/ejava/examples/app/config/propertySource/packaged_propertySource.properties")  
public class PropertySourceConfiguration { }
```

```
#packaged_propertySource.properties  
app.audience=packaged_propertySource.properties value
```

Since the `@PropertySource` is the lowest priority source of these examples, I will trigger it by supplying a fictitious `spring.config.name` so that no default profiles override it.

Supplying Custom spring.config.name Clears Higher Priority Sources

```
$ java -jar target/appconfig-propertySource-example-6.1.0-SNAPSHOT-bootexec.jar \  
--spring.config.name=none
```

```
Application @Bean says Hey packaged_propertySource.properties value
```

6.2. application.properties

If we allow the application to use the default "application" config name, the `application.properties` file will take next precedence.

application.properties with app.audience Property

```
src
 |-- main
    |-- ...
    |-- resources
        |-- application.properties

#application.properties
app.audience=application.properties value
```

Property File Property Source

```
$ java -jar target/appconfig-propertysource-example-6.1.0-SNAPSHOT-bootexec.jar

Application @Bean says Hey application.properties value ①
```

① "application.properties value" comes from a property file

6.3. Profiles

The next level of override — as you know — is a profile.

application-example.properties

```
#application-example.properties
app.audience=application-example.properties value
```

Profile Overrides

```
$ java -jar target/appconfig-propertysource-example-6.1.0-SNAPSHOT-bootexec.jar \
--spring.profiles.active=example

Application @Bean says Hey application-example.properties value
```

6.4. Environment Variables

We can override the property file specification using an environment variable. The environment variable is expressed in ALL_CAPS with an underscore (_) replacing all dots (.). Therefore `app.audience` is expressed as `APP_AUDIENCE`. The expression of the environment variable in bash is `export APP_AUDIENCE="some value"`.

Environment Variable Source Overrides Property File Source

```
$ (export APP_AUDIENCE=env && \①  
java -jar target/appconfig-propertysource-example-6.1.0-SNAPSHOT-bootexec.jar \  
--spring.profiles.active=example)  
  
Application @Bean says Hey env ②
```

① "env" comes from an environment variable

② environment variable overrides file source and **example** profile source



Express environment variable and command in subshell parens ('()') versus polluting current shell

```
(export APP_AUDIENCE=... && cmd)
```



Use **unset** to remove an environment variable from the current bash shell.

```
unset APP_AUDIENCE
```



Relaxed binding rules prevent environment variables from expressing case-sensitive values

```
logging.level.org.springframework.web.filter.CommonsRequestLoggingFilter=DEBUG
```

6.5. Java System Properties

Next in priority is the Java system property. The example below shows the **-Dapp.audience=...** Java system property overriding the environment variable.

Java System Property - Property Source

```
$ (export APP_AUDIENCE=env &&  
java -jar \  
-Dapp.audience=sys \①  
target/appconfig-propertysource-example-6.1.0-SNAPSHOT-bootexec.jar)  
  
Application @Bean says Hey sys ②
```

① "sys" comes from system property

② system property overrides environment variable and file source

6.6. spring.application.json

Next in priority are properties expressed within a JSON document supplied by the `spring.application.json` property. The `spring.application.json` property can be expressed as an environment variable, Java system property, or command line. The specific JSON will be used based on the priority of the source. The properties within the JSON will override whatever we have demonstrated so far.

6.6.1. JSON Expressed as Environment Variable

spring.application.json Expressed as Environment Variable

```
$ (export APP_AUDIENCE=env && \  
  SPRING_APPLICATION_JSON='{"app.audience":"envjson"}' && \①\  
  java -jar -Dapp.audience=sys \  
  target/appconfig-property-source-example-6.1.0-SNAPSHOT-bootexec.jar)  
  
Application @Bean says Hey envjson ②
```

- ① "envjson" comes from `spring.application.json` property expressed using environment variable
- ② `spring.application.json` overrides system property, environment variable and file

6.6.2. JSON Expressed as System Property

spring.application.json Expressed as System Property

```
(export APP_AUDIENCE=env && \  
  SPRING_APPLICATION_JSON='{"app.audience":"envjson"}' && \  
  java -jar \  
  -Dapp.audience=sys \  
  -Dspring.application.json='{"app.audience":"sysjson"}' \①\  
  target/appconfig-property-source-example-6.1.0-SNAPSHOT-bootexec.jar)  
  
Application @Bean says Hey sysjson ②
```

- ① "sysjson" comes from `spring.application.json` property expressed using system property
- ② system property expression overrides environment variable expression

6.7. Command Line Arguments

Next in priority are command line arguments. The example below shows the `--app.audience=...` command line argument overriding everything we have shown defined to date.

Command Line Argument Property Source

```
$ (export APP_AUDIENCE=env && \  
  SPRING_APPLICATION_JSON='{"app.audience":"envjson"}' && \  
  java -jar \  
  --app.audience=cmdline
```

```
-Dapp.audience=sys \  
-Dspring.application.json='{"app.audience":"sysjson"}' \  
target/appconfig-property-source-example-6.1.0-SNAPSHOT-bootexec.jar \  
--app.audience=cmdarg) ①
```

Application @Bean says Hey cmdarg ②

- ① "cmdarg" comes from command line argument
- ② command line argument overrides `spring.application.json`, system property, environment variable and file

6.8. @SpringBootTest.properties

We will soon discuss testing, but know now that properties expressed as part of the `@SpringBootTest` declaration overrides all other property sources.

SpringBootTest Property Source

```
@SpringBootTest(  
    properties={"app.audience=test"}  
public class SampleNTest {
```

Chapter 7. Summary

In this module we

- supplied property value(s) through a set of property files
- used both `properties` and `YAML` formatted files to express property values
- specified base filename(s) to use using the `--spring.config.name` property
- specified profile(s) to use using the `--spring.profiles.active` property
- specified paths(s) to search using the `--spring.config.location` property
- specified a custom file to load using the `@PropertySource` annotation
- specified multiple names, profiles, and locations
- specified property overrides through multiple types of external sources

In future modules we will show how to leverage these property sources in a way that can make configuring the Java code easier.