Value Injection

jim stafford

Fall 2022 v2022-09-10: Built: 2022-12-07 06:11 EST

Table of Contents

Chapter 1. Introduction

One of the things you may have noticed was the hard-coded string in the AppCommand class in the previous example.

```
public void run(String... args) throws Exception {
    greeter.sayHello("World");
}
```

Lets say we don’t want the value hard-coded or passed in as a command-line argument. Lets go down a path that uses standard Spring value injection to inject a value from a property file.

Ref: Spring Boot application.properties file by Daniel Olszewski

1.1. Goals

The student will learn:

- how to configure an application using properties
- how to use different forms of injection

1.2. Objectives

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

1. implement value injection into a Spring Bean attribute using
   - field injection
   - constructor injection
2. inject a specific value at runtime using a command line parameter
3. define a default value for the attribute
4. define property values for attributes of different type

Chapter 2. @Value Annotation

To inject a value from a property source, we can add the Spring `@Value` annotation to the component property.

```
package info.ejava.examples.app.config.valueinject;

import org.springframework.beans.factory.annotation.Value;
...
@Component
public class AppCommand implements CommandLineRunner {
    private final Hello greeter;

    @Value("${app.audience}") ②
    private String audience; ①

    public AppCommand(Hello greeter) {
        this.greeter = greeter;
    }

    public void run(String... args) throws Exception {
        greeter.sayHello(audience);
    }
}
```

① defining target of value as a FIELD

② using FIELD injection to directly inject into the field

There are no specific requirements for property names but there are some common conventions followed using `(prefix).(property)` to scope the property within a context.

- `app.audience`
- `logging.file.name`
- `spring.application.name`

2.1. Value Not Found

However, if the property is not defined anywhere the following ugly error will appear.

```
2019-09-22 20:16:24.286  WARN 38915 --- [main]
s.c.a.AnnotationConfigApplicationContext :
  Exception encountered during context initialization - cancelling refresh attempt:
  org.springframework.beans.factory.BeanCreationException: Error creating bean with
  name 'appCommand': Injection of autowired dependencies failed; nested exception
  is java.lang.IllegalArgumentException: Could not resolve placeholder
  'app.audience' in value "${app.audience}"
```

2.2. Value Property Provided by Command Line

We can try to fix the problem by defining the property value on the command line

```
$ java -jar target/appconfig-valueinject-example-*-SNAPSHOT-bootexec.jar \
  --app.audience="Command line World" ①
...
Application @Bean says Hey Command line World
```

① use double dash (`--`) and property name to supply property value

2.3. Default Value

We can defend against the value not being provided by assigning a default value where we declared the injection

```
@Value("${app.audience:Default World}") ①
private String audience;
```

① use `:value` to express a default value for injection

That results in the following output

Property Default

```
$ java -jar target/appconfig-valueinject-example-*-SNAPSHOT-bootexec.jar
...
Application @Bean says Hey Default World
```

Property Defined

```
$ java -jar target/appconfig-valueinject-example-*-SNAPSHOT-bootexec.jar \
    --app.audience="Command line World"
...
Application @Bean says Hey Command line World
```

Chapter 3. Constructor Injection

In the above version of the example, we injected the `Hello` bean through the constructor and the `audience` property using FIELD injection. This means

- the value for `audience` attribute will not be known during the constructor
- the value for `audience` attribute cannot be made final

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

public AppCommand(Hello greeter) {
    this.greeter = greeter;
    greeter.sayHello(audience); //X-no ①
}
```

① `audience` value will be null when used in the constructor — when using FIELD injection

3.1. Constructor Injection Solution

An alternative to using `field` injection is to change it to `constructor` injection. This has the benefit of having all properties injected in time to have them declared final.

```
@Component
public class AppCommand implements CommandLineRunner {
    private final Hello greeter;
    private final String audience; ②
    public AppCommand(Hello greeter,
                      @Value("${app.audience:Default World}") String audience) {
        this.greeter = greeter;
        this.audience = audience; ①
    }
```

① `audience` value will be known when used in the constructor

② `audience` value can be optionally made final

Chapter 4. @PostConstruct

If field-injection is our choice, we can account for the late-arriving injections by leveraging `@PostConstruct`. The Spring container will call a method annotated with `@PostConstruct` after instantiation (ctor called) and properties fully injected.

```
import javax.annotation.PostConstruct;
...
@Component
public class AppCommand implements CommandLineRunner {
    private final Hello greeter; ①
    @Value("${app.audience}")
    private String audience; ②

    @PostConstruct
    void init() { ③
        greeter.sayHello(audience); //yes-greeter and audience initialized
    }
    public AppCommand(Hello greeter) {
        this.greeter = greeter;
    }
```

① constructor injection occurs first and in-time to declare attribute as `final`

② field and property-injection occurs next and can involve many properties

③ Container calls @PostConstruct when all injection complete

Chapter 5. Property Types

5.1. non-String Property Types

Properties can also express non-String types as the following example shows.

```
@Component
public class PropertyExample implements CommandLineRunner {
    private final String strVal;
    private final int intVal;
    private final boolean booleanVal;
    private final float floatVal;

    public PropertyExample(
            @Value("${val.str:}") String strVal,
            @Value("${val.int:0}") int intVal,
            @Value("${val.boolean:false}") boolean booleanVal,
            @Value("${val.float:0.0}") float floatVal) {
        ...
```

The property values are expressed using string values that can be syntactically converted to the type of the target variable.

```
$ java -jar target/appconfig-valueinject-example-*-SNAPSHOT-bootexec.jar \
  --app.audience="Command line option" \
  --val.str=aString \
  --val.int=123 \
  --val.boolean=true \
  --val.float=123.45
...
Application @Bean says Hey Command line option
strVal=aString
intVal=123
booleanVal=true
floatVal=123.45
```

5.2. Collection Property Types

We can also express properties as a sequence of values and inject the parsed string into Arrays and Collections.

```
    ...
    private final List<Integer> intList;
    private final int[] intArray;
    private final Set<Integer> intSet;
    public PropertyExample(...
            @Value("${val.intList:}") List<Integer> intList,
            @Value("${val.intList:}") Set<Integer> intSet,
            @Value("${val.intList:}") int[] intArray) {
        ...

  --val.intList=1,2,3,3,3
...
intList=[1, 2, 3, 3, 3] ①
intSet=[1, 2, 3] ②
intArray=[1, 2, 3, 3, 3] ③
```

① parsed sequence with duplicates injected into List maintained duplicates

② prased sequence with duplicates injected into Set retained only unique values

③ parsed sequence with duplicates injected into Array maintained duplicates

5.3. Custom Delimiters (using Spring EL)

We can get a bit more elaborate and define a custom delimiter for the values. However, it requires the use of Spring Expression Language (EL) `#{}` operator. (Ref: A Quick Guide to Spring @Value)

```
private final List<Integer> intList;
private final List<Integer> intListDelimiter;

public PropertyExample(
...
        @Value("${val.intList:}") List<Integer> intList,
        @Value("#{'${val.intListDelimiter:}'.split('!')}") List<Integer>
intListDelimiter, ②
...

    --val.intList=1,2,3,3,3 --val.intListDelimiter='1!2!3!3!3' ①
...
intList=[1, 2, 3, 3, 3]
intListDelimeter=[1, 2, 3, 3, 3]
...
```

① sequence is expressed on command line using two different delimiters

② `val.intListDelimiter` String is read in from raw property value and segmented at the custom `!` character

5.4. Map Property Types

We can also leverage Spring EL to inject property values directly into a Map.

```
private final Map<Integer,String> map;
```

```
public PropertyExample( ...
        @Value("#{${val.map:{}}}") Map<Integer,String> map) { ①
    ...

   --val.map="{0:'a', 1:'b,c,d', 2:'x'}"
...
map={0=a, 1=b,c,d, 2=x}
```

① parsed map injected into Map of specific type using Spring Expression Language (`#{}') operator

5.5. Map Element

We can also use Spring EL to obtain a specific element from a Map.

```
    private final Map<String, String> systemProperties;

    public PropertyExample(
...
        @Value("#{${val.map:{0:'',3:''}}[3]}") String mapValue, ①
...
   (no args)
...
mapValue= ②

   --val.map={0:'foo', 2:'bar, baz', 3:'buz'}
...
mapValue=buz ③
...
```

① Spring EL declared to use Map element with key 3 and default to a Map of 2 elements with key 0 and 3

② With no arguments provided, the default `3:''` value was injected

③ With a map provided, the value `3:'buz'` was injected

5.6. System Properties

We can also simply inject Java System Properties into a Map using Spring EL.

```
    private final Map<String, String> systemProperties;

    public PropertyExample(
...
     @Value("#{systemProperties}") Map<String, String> systemProperties) { ①
...
     System.out.println("systemProperties[user.timezone]=" + systemProperties.get(
"user.timezone")); ②
...
```

```
systemProperties[user.timezone]=America/New_York
```

① Complete Map of system properties is injected

② Single element is accessed and printed

5.7. Property Conversion Errors

An error will be reported and the program will not start if the value provided cannot be syntactically converted to the target variable type.

```
$ java -jar target/appconfig-valueinject-example-*-SNAPSHOT-bootexec.jar \
    --val.int=abc
...
TypeMismatchException: Failed to convert value of type 'java.lang.String'
to required type 'int'; nested exception is java.lang.NumberFormatException:
For input string: "abc"
```

Chapter 6. Summary

In this section we

- defined a value injection for an attribute within a Spring Bean using
    - field injection
    - constructor injection
- defined a default value to use in the event a value is not provided
- defined a specific value to inject at runtime using a command line parameter
- implemented property injection for attributes of different types
    - Built-in types (String, int, boolean, etc)
    - Collection types
    - Maps
- Defined custom parsing techniques using Spring Expression Language (EL)

In future sections we will look to specify properties using aggregate property sources like file(s) rather than specifying each property individually.