***Wiring with the Spring Expression Language***

* Spring 3 introduced Spring Expression Language (SpEL), a powerful yet succinct way of wiring values into a bean’s properties or constructor arguments using expressions that are evaluated at runtime.
* Things a SpEL can do:
* The ability to reference beans by their IDs
* Invoking methods and accessing properties on objects
* Mathematical, relational, and logical operations on values
* Regular expression matching
* Collection manipulation
* SpEL can also be used for purposes other than dependency injection.
* Spring Security, for example, supports defining security constraints using SpEL expressions.
* And if you’re using Thymeleaf templates as the view in your Spring MVC application, those templates can use SpEL expressions to reference model data.

**A FEW SPEL EXAMPLES**

* SpEL is such a flexible expression language that it would be impossible to show you all the ways it can be used in the space allowed in this book. But there is enough room to show you a few basic examples from which you can draw inspiration for your own expressions.
* The First thing to know is that SpEL expressions are framed with #{ … }, much as property placeholders are framed with ${ … }. What follows is possibly one of the simplest SpEL expressions you can write:

AcroRd32_2017-07-29_14-32-19.png

* Stripping away the #{ … } markers, what’s left is the body of a SpEL expression, which is a numeric constant. It probably won’t surprise you much to learn that this expression evaluates to the numeric value of 1. Of course, you’re not likely to use such a simple expression in a real application. You’re more likely to use such a simple expression in a real application. You’re more likely to build up more interesting expressions, such as this one:

AcroRd32_2017-07-29_14-32-19.png

* Ultimately this expression evaluates to the current time in milliseconds at the moment when the expression is evaluated. The T( ) operator evaluates *java.lang.System* as a type so that the *staticcurrentTimeMillis()* method can be invoked.
* SpEL expressions can also refer to other beans or properties on those beans. For example, the following expression evaluates to the value of the *artist* property on a bean whose ID is *sgtPeppers:*

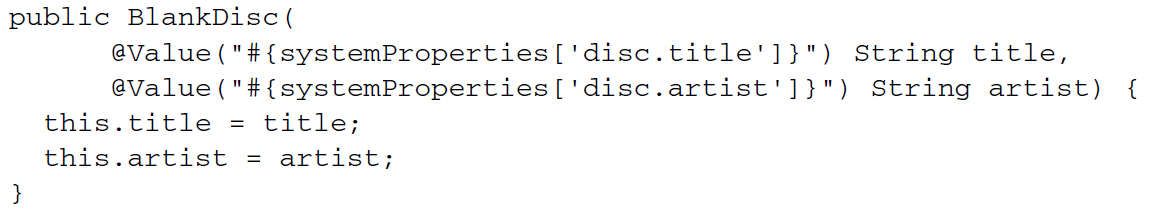
AcroRd32_2017-07-29_14-32-19.png

* You can also refer to system properties via the *systemProperties* object:

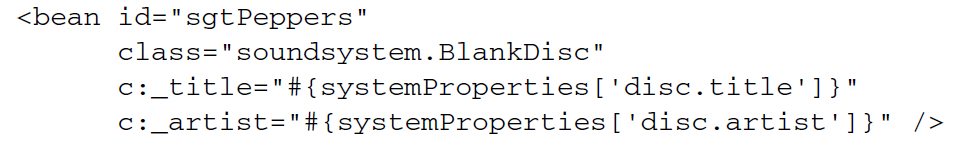
AcroRd32_2017-07-29_14-32-19.png

* These are just a few examples of SpEL. You’ll see more before this chapter ends. But first, let’s consider how you might use these expressions during bean wiring.

When injecting properties and constructor arguments on beans that are created via component-scanning, you can use the *@Value* annotation, much as you saw earlier with property placeholders. Rather than use a placeholder expression, however, you use a SpEL expression. For example, here’s what the *BlankDisc* constructor might look like, drawing the album title and artist from system properties:



In XML configuration, you can pass in the SpEL expression to the *value* attribute of *<property> or <constructor-arg>,* or as the value given to a p-namespace or c-namespace entry. For example, here’s the XML declaration of the *BlankDisc* bean that has its constructor arguments set from a SpEL expression:



Now that we’ve looked at a few simple examples and how to inject values resolved from SpEL expressions, let’s go over some of the primitive expressions supported in SpEL.

**EXPRESSING LITERAL VALUES**

* You’ve already seen an example of using SpEL to express a literal integer value. But it can also be used for floating-point numbers, *String* values, and *Boolean* values.
* Here’s an example of a *SpEL* expression that is a floating-point value:

AcroRd32_2017-07-29_14-32-19.png

* Numbers can also be expressed in scientific notation. For example, the following expression evaluates to 98,700:

AcroRd32_2017-07-29_14-32-19.png

* A SpEL expression can also evaluate literal *String* values, such as:

AcroRd32_2017-07-29_14-32-19.png

* Finally, Boolean literals *true* and *false* are evaluated to their Boolean value. For Example,

AcroRd32_2017-07-29_14-32-19.png

* Working with literal values in SpEL is mundane. After all, you don’t need SpEL to set an integer property to 1 or a Booleaan property to *false.* I admit there’s not much use in SpEL expressions that only contain literal values. But remember that more interesting SpEL expressions are composed of simpler expression, so it’s good to know how to work with literal values in SpEL.

**REFERENCING BEANS, PROPERTIES, AND METHODS**

* Another basic thing that a SpEL expression can do is reference another bean by its ID.
* For example, you could use SpEL to wire one bean into another bean’s property by using the bean ID as the SpEL expression (in this case, a bean whose ID it sgtPeppers):

AcroRd32_2017-07-29_14-32-19.png

* Now let’s say that you want to refer to the *artist* property of the *sgtPeppers.* What follows the period delimiter is a reference to the *artist* property.
* In addition to referencing a bean’s properties, you can also call methods on a bean. For example, suppose you have another bean whose ID is *artistSelector.* You can call that bean’s *selectArtist()* method in a SpEL expression like this:

AcroRd32_2017-07-29_14-32-19.png

* You can also call methods on the value returned from the invoked method. for example, if *selectArtist()* returns a String, you can call *toUpperCase()* to make the entire artist name uppercase lettering:

AcroRd32_2017-07-29_14-32-19.png

* This will work fine, as long as *selectArtist()* doesn’t return *null.* To guard against a *NullPointerException,* you can use the type-safe operator:

AcroRd32_2017-07-29_14-32-19.png

* Instead of a lonely dot (.) to access the *toUpperCase()* method now you’re using the *?.* operator. This operator makes sure the item to its left is’t *null* before accessing the thing on its right. So, if *selectArtist()* *returns* null, then SpEL won’t even try to invoke *toUpperCase().* The expression will to *null.*

**WORKING WITH TYPES IN EXPRESSIONS**

* The key to working with class-acoped methods and constants in SpEL is to use the *T()* operator. For example, to express Java’s *Math* class in SpEL, you need to use the *T()* operator like this:

T(java.lang.Math)

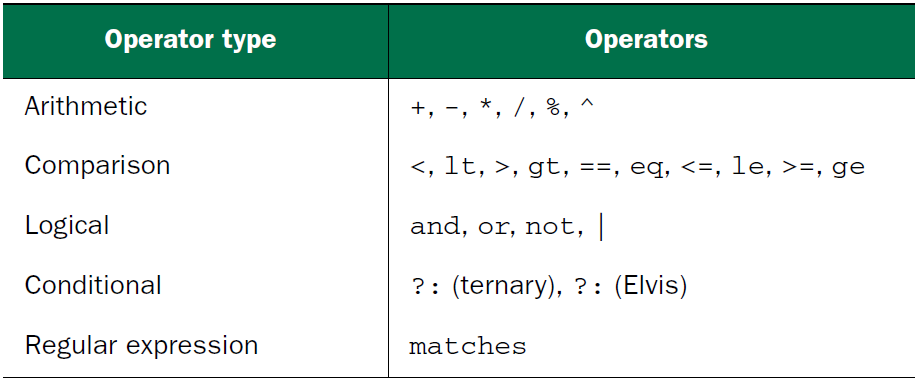
* The result of the *T()* operator, as shown here, is a *Class* object that represents *java.lang.Math.* You can even wire it into a bean property of type *Class,* if you want. But the real value of the *T( )* operator is that it gives you access to static methods and constants SpEL expression does the trick:

**AcroRd32_2017-07-29_14-32-19.png**

* Similarly, static methods can be invoked in the type resolved with the *T( )* operator. You’ve seen an example of using *T( )* to make a call to *System.currentTimeMillis().*
* Here’s another example that evaluates to a random value between 0 and 1:

**AcroRd32_2017-07-29_14-32-19.png**

* SpEL offers several operators that you can apply on values in SpEL expressions:

****

* As a simple example of using one of these operators ( \* ), but it also shows how you can compose simpler expression into a more complex expression.:

AcroRd32_2017-07-29_14-32-19.png

Here the value of pi is multiplied by 2, and the result is multiplied by the value of the *radius* property of a bean whose *ID* id *circle.* Essentially, it evaluates to the circumference of the circle defined in the *circle* bean.

* Similarly, you can use the carat symbol ( ^ ) in an expression to calculate a circle’s area:

AcroRd32_2017-07-29_14-32-19.png

The carat symbol is the power-of operator. In this case, it’s used to calculate the square of the circle’s radius.

* When working with *String* values, the + operator performs concatenation, just as in Java:

AcroRd32_2017-07-29_14-32-19.png

* SpEL also offers comparison operators for comparing values in an expression. Notice in previous table the comparison operators come in two forms: symbolic and textual. For the most part, the symbolic sperators are equivalent to their textual counterparts, and you’re welcome to use whichever one suits you best.
* For example, to compare two numbers for equality, you can use the double-equal ( == ) operator:

AcroRd32_2017-07-29_14-32-19.png

or you can use the textual *eq* operator:

AcroRd32_2017-07-29_14-32-19.png

* Either way, the result is the same. The expression evaluates to a Boolean: *ture* if *counter.total* is equal to 100 or *false* if it’s not.
* SpEL also offers a ternary operator that works much like Java’s ternary operator. For example, the following expression evaluates to the *String “Winner!”* if *scoreboard.score* > 1000 or “Loser” if not:

AcroRd32_2017-07-29_14-32-19.png

* A common use of the ternary operator is to check for a *null* value and offer a default value in place of the *null.* If *disc.title* is *null,* then the expression evaluates to “Rattle and Hum”.

AcroRd32_2017-07-29_14-32-19.png

* This expression is commonly referred to as the *Elvis* operator. This strange name comes from using the operator as an emoticon, where the question mark appears to form the shape of Elvis Presley’s hair style.

**EVALUATING REGULAR EXPRESSIONS**

* When working with text, it’s sometimes useful to check whether that text matches a certain pattern. SpEL supports pattern matching in expressions with its *matches* operator. The *matches* operator attempts to apply a regular expression (given as its rightside argument) against a *String* value ( given as the left-side *argument ) .* The result of a *matches* evaluation is a Boolean value: *true* if the value matches the regular expression, and false otherwise.

To demonstrate, suppose you want to check whether a *String* contains a valid email address. In that case, you can apply *matches* like this:

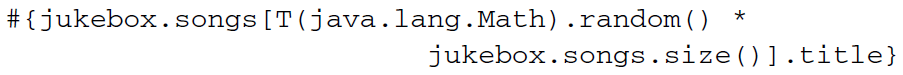
AcroRd32_2017-07-29_14-32-19.png

**EVALUATING COLLECTIONS**

* Some of SpEL’s most amazing tricks involve working with collections and arrays. The most basic thing you can do is reference a single element from a list:

**AcroRd32_2017-07-29_14-32-19.png**

* This evaluates to the *title* property of the fifth (zero-based) element from the *songs* collection property on the bean whose ID is *jukebox*.
* To spice things up a bit, I suppose you could randomly select a song from the jukebox:

****

* As it turns out, the [ ] operator used to fetch an indexed element from a collection or array can also be used to fetch a single character from a *String.* For example,

**AcroRd32_2017-07-29_14-32-19.png**

* This references the fourth (zero-based) character in the *String,* or s.
* **SpEL** also offers a selection operator ( .?[ ] ) to filter a collection into a subset of the collection. As a demonstration, suppose you want a list of all songs in the jukebox where the *artist* property is *Aerosmith.* The following expression uses the selection operator to arrive at the list of available Aerosmith songs:

**AcroRd32_2017-08-02_20-08-14.png**

* **As you can see, the selection operator accepts another expression within its square brackets. As SpEL iterates over the list of songs, it evaluates that expression each entry in the songs collection.** If the expression evaluates to true, then the entry is carried over into the new collection. Otherwise it’s left out of the new collection. In this case, the inner expression checks to see if the song’s *artist* property equals *Aerosmith.*
* **SpEL** also offers two other selection operations: **.^[ ]** for selecting the first matching entry and **.$[ ]** for selecting the last matching entry. To demonstrate, consider this expression, which finds the first song in the list whose *artist* property is *Aerosmith:*

**AcroRd32_2017-08-02_20-08-14.png**

* Finally, SpEL offers a projection operator ( **.![ ]** ) to project properties from the elements in the collection onto a new collection. As an example, suppose you don’t want a collection of the song objects, but a collection of all the song titles. The following expression projects the *title* property into a new collection of *Strings:*

**AcroRd32_2017-08-02_20-08-14.png**

* Naturally, the projection operator can be combined with any of SpEL’s other operators, including the selection operator. For example, you could use this expression to obtain a list of all of Aerosmith’s songs:

**AcroRd32_2017-08-02_20-08-14.png**

* We’ve only scratched the surface of what SpEL can do.