

Basic Spring 4.0

Lesson 3: Spring Expression
Language (SpEL)

Lesson Objectives

- Introduction to SpEL (Spring Expression Language)
 - SpEL Expression fundamentals
 - Expression Language features
 - Reduce configuration with @Value

3.1 : SpEL Expression fundamentals

What is SpEL?

- The Spring Expression Language is a powerful expression language that supports querying and manipulating an object graph at runtime.
- SpEL supports many functionalities including:
 - Literal expressions
 - Boolean and relational operators
 - Regular and class expressions
 - Accessing properties, arrays, lists, maps
 - Method invocation
 - Calling constructors
 - Bean references
 - Array construction
 - Inline lists
 - User defined functions
 - Templated expressions



Copyright © Capgemini 2015. All Rights Reserved 3

So far we have seen how to wire dependencies using setter and constructor injections. But these have been statically defined in the Spring configuration file. When we wired the `exchangeService` property into the `CurrencyConverter` bean, that value was determined at development time. Likewise, when we wired references to other beans, those references were also statically determined in the Spring configuration.

What if we want to wire properties with values at runtime? Spring 3's **Spring Expression Language (SpEL)** is a powerful way of wiring values into a bean's properties or constructor arguments using expressions that are evaluated at runtime. SpEL syntax is similar to Unified EL but offers additional features like method invocation and basic string templating functionality.

SpEL is based on a technology agnostic API allowing other expression language implementations to be integrated should the need arise. It thus can be used independently. It supports many functionalities as listed above.

3.2 : Expression Language features

Exploring literals and Types

- Working with Literals:

- Wire SpEL expression into a bean's property by using `#{ <exprn-string> }`

```
<property name="count" value="#{5}"/>
<property name="message" value="The value is #{5}"/>
<property name="frequency" value="#{89.7}"/>
<property name="name" value="#{'Chuck'}"/>
```

examples

- Working With Types:

- Use the `T()` operator to work with class-scoped methods & constants
- Example: `T(java.lang.Math)` //expresses Java's Math class in SpEL

```
<property name="multiplier" value="#{T(java.lang.Math).PI}"/>
<property name="randomNumber" value="#{T(java.lang.Math).random()}/>
```

examples



Copyright © Capgemini 2015. All Rights Reserved 4

Literal Values: SpEL expressions, like any other expression, are evaluated for some value. SpEL can evaluate literal values, references to a bean's properties, a constant on some class etc. The simplest SpEL expression for example is 5, which evaluates to an integer value of 5. We can wire this value into a bean's property by using `#{ }` markers in a `<property>` element's value attribute, as shown in example above. The `#{ }` markers indicate that the content that they contain is a SpEL expression.

Similarly, see example above for expressing Floating-point numbers. Literal String values can be expressed in SpEL with either single or double quote marks. You can also use Boolean true and false values. Eg- `<property name="enabled" value="#{false}"/>`

Working With Types: The result of the `T()` operator is a Class object that represents the given class. The `T()` operator thus gives us access to static methods and constants on a given class. Eg, to wire the value of pi into a bean property, use: `<property name="multiplier" value="#{T(java.lang.Math).PI}"/>` Likewise, static methods can also be invoked on the result of the `T()` operator. Eg, to wire a random number (between 0 and 1) into a bean property, use: `<property name="randomNumber" value="#{T(java.lang.Math).random()}/>` When the application is starting up and Spring is wiring the `randomNumber` property, it'll use the `Math.random()` method to determine a value for that property!

3.2 : Expression Language features

Referencing Beans, Properties, And Methods

- SpEL allows to wire one bean into another bean's property by using the bean ID as the SpEL expression:

```
<property name="exchangeService " value="#{exchangeService}"/>
```

```
<bean id="currencyConverter" class="training.CurrencyConverterImpl">
  <property name="exchangeRate"
    value="#{exchangeService.exchangeRate}" />
</bean>
```

Bean ID

Property name

referencing
beans
properties

```
<property name="exchangeService "
  value="#{exchangeService.getExchangeRate()}" />
```

Invoking
methods

Copyright © Capgemini 2015. All Rights Reserved 5

A SpEL expression can reference another bean by its ID. See first example above. We used SpEL to wire the bean whose ID is “exchangeService” into an exchangeService property. We can do this by using the ref attribute too! `<property name="exchangeService" ref="exchangeService"/>` The outcome is the same. But let us see how to take advantage of being able to wire bean references with SpEL. The second example configures a new CurrencyConverterImpl bean whose ID is currencyConverter. This is wired to whatever exchangeRate the exchangeService bean provides. This is equivalent to:

```
CurrencyConverterImpl currencyConverter = new CurrencyConverterImpl ();
currencyConverter.setExchangeRate(exchangeService.getExchangeRate());
```

3.2 : Expression Language features

Performing operations on SpEL values

- SpEL includes several operators to manipulate the values of an expression.

Operation type	Operators
Arithmetic	+, -, *, /, %, ^
Relational	<, >, ==, <=, >=, lt, gt, eq, le, ge
Logical	and, or, not,
Conditional	?: (ternary), ?: (Elvis)
Regular expression	matches

examples

```
<property name="adjustedAmount" value="#{counter.total + 42}"/>
<property name="result" value="#{2 * T(java.lang.Math).PI * circle.radius}"/>
<property name="fullName" value="#{emp.firstName + ' ' + emp.lastName}"/>
<property name="redCustomer" value="#{account.balance le 100000}"/>
<property name="outOfStock" value="#{!product.available}"/>
<property name="outcome"
    value="#{T(java.lang.Math).random() > .5 ? 'win' : 'lose'}"/>
```



Copyright © Capgemini 2015. All Rights Reserved 6

Like in Java, + operator is overloaded to perform concatenation on String values. We know that the less-than (<) and greater-than (>) operators are used to compare different values. Unfortunately, they pose a problem when using these expressions in Spring's XML configuration (since they have special meaning in XML). So, when using SpEL in XML,⁵ it's best to use SpEL's textual alternatives like le (<), gt (>) etc.

SpEL supports regular expressions using the matches operator, which is a relational operator returning true or false. Example:

```
<util:map id="regExpsSamples" value-type="java.lang.Object"
    key-type="java.lang.String">
  <entry key="#{'a string' matches 'a.*'}"
    value="#{'a string' matches 'b.*'}"/>
</util:map>
```

3.2 : Expression Language features

Working with collections

```
package trg.spring;
public class City {
    private String name;
    private String state;
    private int population;
    //setter and getter methods for each of these properties
}
```

```
<util:list id="cities">
  <bean class="trg.spring.City"
    p:name="Chicago" p:state="IL" p:population="2853114"/>
  <bean class="trg.spring.City"
    p:name="Atlanta" p:state="GA" p:population="537958"/>
  <bean class="trg.spring.City"
    p:name="Dallas" p:state="TX" p:population="1279910"/>
  .....
</util:list>
```



Copyright © Capgemini 2015. All Rights Reserved 7

Let us digress a bit and see how Spring allows us to create collection via configuration file.

From Spring 2.5 onwards, in addition to standard `<property>` tag, you can use the `p` namespace to set properties of beans.

The `<util:list>` element comes from Spring's `util` namespace. It creates a bean of type `java.util.List` that contains all of the values or beans that it contains. In this case, that's a list of `City` beans.


Example for creating Map:

```
<util:map id="numbersMap" value-type="java.lang.Integer"
  key-type="java.lang.String">
  <entry key="one" value="1"/>
  <entry key="two" value="2"/>
  <entry key="three" value="3"/>
  <entry key="four" value="4"/>
  <entry key="five" value="5"/>
</util:map>
```

3.2 : Expression Language features

Accessing collections

- 1 `<property name="customerCity" value="#{cities[2]}" />`
- 2 `<property name="customerCity" value="#{cities['Dallas']}" />`
- 3 `<property name="userName" value="#{userprops['user.name']}" />` selection
- 4 `<property name="smallCities" value="#{cities.?[population lt 100000]}" />`
- 5 `<property name="cityNames" value="#{cities.![name]}" />` projection
- 6 `<property name="cityNames" value="#{cities.![name + ', ' + state]}" />`
- 7 `<property name="cityNames" value="#{cities.?[population gt 100000].![name + ', ' + state]}" />`

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 8

Example -1 : This selects the third city from the **list** to assign to customerCity

Example -2: Assuming that cities is a **java.util.Map** collection, with city-name as the key. The example shows how to retrieve the entry for Dallas.

Example -3: We can use the **[]** operator is to retrieve a value from a **java.util.Properties** collection too. First: load a properties configuration file into Spring using the **<util:properties>** element as follows:

```
<util:properties id="userprops" location="classpath:user.properties"/>
```

The userprops bean is a **java.util.Properties** that contains all of the entries in the file named **user.properties**. Accessing a property from that file is similar to accessing a member of a **Map**. The 3rd example above reads a property whose name is **user.name** from the **userprops** bean

Example -4: We would like a list of cities whose population is less than 100,000. Use the selection operator (**.?[]**) when doing the wiring. The selection operator creates a new collection whose members include only those members from the original collection that meet the criteria expressed between the square braces. In this case, the **smallCities** property will be wired with a list of **City** objects whose population property is less than 100,000.

Example -5: **Projecting collections means collecting a particular property from each of the members of a collection into a new collection.** Use SpEL's projection operator (**.![]**) to do this. Example-5 retrieves a list of city names from the collection of **City** objects. You can also retrieve multiple members as the next example shows. The final example (7) is a combination of selection and projection.

3.3 : Reduce configuration with @Value


@Value annotation

```
package training.spring.spel;
@Component("user")
public class UserBean {
    @Value("#{userprops.username}")
    private String username;
    @Value("#{userprops.password}")
    private String password;
    // setter and getter methods for properties
}
```

inject values from the properties files using a SpEL expression

```
<beans xmlns="http://www.springframework.org/schema/beans"
..... >
    <context:component-scan base-package="training.spring.spel" />
    <util:properties id="userprops" location="classpath:user.properties" />
</beans>
```

properties file

 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 9


SpEL combined with @Value annotation is great. We have already seen how component scan and autowiring reduces the size of XML configuration files. However, we still have to deal with beans that need literal values as input. With @Value you can inject values from your properties files using SpEL. Assume that you have a properties file configured as shown above.


To use SpEL in annotation, you must register your component via annotation. If you register your bean in XML and define @Value in Java class, the @Value will fail to execute.



Demo : SpringDemo_SPEL

- Using SpEL



 Capgemini
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 10

Please refer to demo, SpringDemo_SPEL

Lesson Summary

- We have so far seen:
 - SpEL Expression fundamentals
 - Expression Language features
 - Reduce configuration with @Value



Review Questions

- Question 1: _____ markers indicate that the content that they contain is a SpEL expression.
 - Option 1: \${ }
 - Option 2: #{ }
 - Option 3: %{ }

- Question 2: SpEL can access instances of java.lang.Class using the _____ operator
 - Option 1: #{ }
 - Option 2: T
 - Option 3: Type



Review Questions

- Question 3: The _____ effectively creates a bean of type `java.util.Map` that contains all of the values or beans that it contains .
 - Option 1: `<util:list>`
 - Option 2: `<util:properties>`
 - Option 3: `<util:map>`

- Question 4: If `@Value` annotation is used in a component, it is mandatory that the component be annotated with `@Component`
 - Option 1: True
 - Option 2: False

