Spring Expression Language与EL表达式类似,但是更强大 主要用于Spring中

但是可以独立于Spring使用

特点

* 具有字面意义 支持布尔运算 正则表达式 类表达式 属性,数组,list,map 方法调用 三元运算符
* Assignment
* Calling constructors
* Ternary operator
* Variables
* User defined functions
* Collection projection
* Collection selection
* Templated expressions

**class** Simple {

**public** List<Boolean> booleanList = **new** ArrayList<Boolean>();

}

Simple simple = **new** Simple();

simple.booleanList.add(true);

StandardEvaluationContext simpleContext = **new** StandardEvaluationContext(simple);

*// false is passed in here as a string. SpEL and the conversion service will*

*// correctly recognize that it needs to be a Boolean and convert it*

parser.parseExpression("booleanList[0]").setValue(simpleContext, "false");

*// b will be false*

Boolean b = simple.booleanList.get(0);

Spring会认得泛型的类型 在上面是Boolean 所以当你设置为字符串"false"的时候会自动转成Boolean的false

El表达式可以在注解和xml配置文件里使用,要用这样的格式

<bean id="numberGuess" class="org.spring.samples.NumberGuess">

<property name="randomNumber" value="#{ T(java.lang.Math).random() \* 100.0 }"/>

<*!-- other properties --*>

</bean>

You can also refer to other bean properties by name, for example.

<bean id="numberGuess" class="org.spring.samples.NumberGuess">

<property name="randomNumber" value="#{ T(java.lang.Math).random() \* 100.0 }"/>

<*!-- other properties --*>

</bean>

<bean id="shapeGuess" class="org.spring.samples.ShapeGuess">

<property name="initialShapeSeed" value="#{ numberGuess.randomNumber }"/>

<*!-- other properties --*>

</bean>

**6.5.8.1 The #this and #root variables**

The variable #this is always defined and refers to the current evaluation object (against which unqualified references are resolved). The variable #root is always defined and refers to the root context object. Although #this may vary as components of an expression are evaluated, #root always refers to the root.

*// create an array of integers*

List<Integer> primes = **new** ArrayList<Integer>();

primes.addAll(Arrays.asList(2,3,5,7,11,13,17));

*// create parser and set variable 'primes' as the array of integers*

ExpressionParser parser = **new** SpelExpressionParser();

StandardEvaluationContext context = **new** StandardEvaluationContext();

context.setVariable("primes",primes);

*// all prime numbers > 10 from the list (using selection ?{...})*

*// evaluates to [11, 13, 17]*

List<Integer> primesGreaterThanTen =

(List<Integer>) parser.parseExpression("#primes.?[#this>10]").getValue(context);

ExpressionParser parser = new SpelExpressionParser();

String name = parser.parseExpression("null?:'Unknown'").getValue(String.class);

System.out.println(name); // 'Unknown'

ExpressionParser p = **new** SpelExpressionParser();

// 字符串要用''括起来 否则出错

Expression e = p.parseExpression( "'Hello World'" );

Object o = e.getValue();

assertEquals( o, "Hello World" );

e = p.parseExpression( "'ceshi'.length()" );

assertEquals( e.getValue(), 5 );

// 可以调用构造函数

e = p.parseExpression( "new sp.jdbctemplate.Account()" );

o = e.getValue();

assertEquals( o.getClass(), Account.**class** );

Account a = (Account) o;

a.setName( "2337" );

e = p.parseExpression( "name" );

// 以a为根

o = e.getValue( a );

assertEquals( o, "2337" );

/\*

The StandardEvaluationContext is relatively expensive to construct

and during repeated usage it builds up cached state that enables

subsequent expression evaluations to be performed more quickly.

For this reason it is better to cache and reuse them where possible,

rather than construct a new one for each expression evaluation.

\*/

StandardEvaluationContext ec = **new** StandardEvaluationContext(/\*"root"\*/);

ec.setRootObject( a );

o = e.getValue( ec );

assertEquals( o, "2337" );

ec.setRootObject( **null** );

// 在这里注册参数

ec.setVariable( "x", 77 );

// 参数前面加#

e = p.parseExpression( "77\*88+#x" );

o = e.getValue( ec );

assertEquals( o, 77 \* 88 + 77 );

e = p.parseExpression( "'7086'" );

o = e.getValue();

// 不支持 ||

assertEquals( p.parseExpression( "true and !false" ).getValue( Boolean.**class** ), **true** );

assertEquals( p.parseExpression( "1++3" ).getValue( Integer.**class** ), **new** Integer( 4 ) );

// 上面都是getValue 这里来个set

p.parseExpression( "name" ).setValue( a, "改你的名字" );

assertEquals( a.getName(), "改你的名字" );

// 当要指类的时候需要用到T

// 比如

assertEquals( p.parseExpression( "T(Object)" ).getValue( Class.**class** ), Object.**class** );

a.setName( **null** );

// 如果name为null就会返回空而不是抛出异常

System.out.println( p.parseExpression( "name?.length()" ).getValue( a ) );

// 集合选择

// root表示根对象

// this表示当前评估的对象

List<Integer> list = Arrays.asList( 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 );

list = (List<Integer>) p.parseExpression( "#root.?[#this%2==0]" ).getValue( list );

System.out.println( list );

Map<String, Integer> map = **new** HashMap<String, Integer>();

map.put( "一", 1 );

map.put( "二", 2 );

map.put( "三", 3 );

map.put( "四", 4 );

map.put( "五", 5 );

Map<String, Integer> map2 = (Map<String, Integer>) p.parseExpression( "#root.?[value<3]" ).getValue( map );

System.out.println( map2 );

//投影所有的value构成一个list

List<Integer> list3 = (List<Integer>) p.parseExpression( "#root.![value]" ).getValue( map );

System.out.println( list3 );

String randomPhrase =

p.parseExpression("random number is #{T(java.lang.Math).random()}",

**new** TemplateParserContext()).getValue(String.**class**);

System.out.println(randomPhrase);