36 SpEL 正则表达式应用示例及背后原理探究

更新时间：2020-08-12 10:48:47

> 勤学如春起之苗，不见其增，日有所长。——陶潜

背景

SpEL 不仅支持对值进行算术、关系和逻辑运算；SpEL 也支持使用正则表达式，其中对应的关键字为 match。如下示例中即在表达式中使用了正则表达式，表示 123 是否匹配正则表达式“\d{3}”。

```
public void testRegex () {
    ExpressionParser parser = new SpelExpressionParser();
    System.out.println(parser.parseExpression("'***123**' matches '\\d{3}'").getValue(Boolean.class));//正则匹配三位数字
}
```

SpEL 正则表达式应用示例

正则表达式 **bean**：

```
package com.davidwang456.test;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("regexTestBean")
public class RegexTest {
    // Check if this is a digit or not?
    @Value("#{'250' matches '\\d+' }")
    private boolean validDigit;

    public boolean isValidDigit() {
        return validDigit;
    }

    public void setValidDigit(boolean validDigit) {
        this.validDigit = validDigit;
    }
```

配置文件 **applicationContext.xml**：

放在上面的包 com.davidwang456.test 下面。

```xml
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:context="http://www.springframework.org/schema/context"
 xsi:schemaLocation="http://www.springframework.org/schema/beans
 http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
 http://www.springframework.org/schema/context
 http://www.springframework.org/schema/context/spring-context-3.0.xsd">

 <context:component-scan base-package="com.davidwang456.test" />

</beans>
```

测试类：

```java
package com.davidwang456.test;

import org.springframework.context.ApplicationContext;

public class RegexBeanTest {

    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext(
                "com/davidwang456/test/applicationContext.xml");

        RegexTest regex = context.getBean(RegexTest.class);
        System.out.println("Is valid: " + regex.isValidDigit());
    }

}
```

控制台打印结果：

> Is valid: true

深入 **SpEL** 正则表达式应用原理

我们知道，SpEL 最终要转换为 AST，然后执行 SpelNodeImpl 的实现类来完成相应的计算，那么本示例中，可以通过步步调试进入内部找到 OperatorMatches，也可以通过猜测然后验证的方式，可以在 OperatorMatches 的 getValueInternal() 方法打出调用链接。

```
/**
 * Check the first operand matches the regex specified as the second operand.
 * @param state the expression state
 * @return {@code true} if the first operand matches the regex specified as the
 * second operand, otherwise {@code false}
 * @throws EvaluationException if there is a problem evaluating the expression
 * (e.g. the regex is invalid)
 */
@Override
public BooleanTypedValue getValueInternal(ExpressionState state) throws EvaluationException {
    StackUtils.getStack();
    SpelNodeImpl leftOp = getLeftOperand();
    SpelNodeImpl rightOp = getRightOperand();
    String left = leftOp.getValue(state, String.class);
    Object right = getRightOperand().getValue(state);

    if (left == null) {
        throw new SpelEvaluationException(leftOp.getStartPosition(),
                SpelMessage.INVALID_FIRST_OPERAND_FOR_MATCHES_OPERATOR, (Object) null);
    }
    if (!(right instanceof String)) {
        throw new SpelEvaluationException(rightOp.getStartPosition(),
                SpelMessage.INVALID_SECOND_OPERAND_FOR_MATCHES_OPERATOR, right);
    }

    try {
        String rightString = (String) right;
        Pattern pattern = this.patternCache.get(rightString);
        if (pattern == null) {
            pattern = Pattern.compile(rightString);
            this.patternCache.putIfAbsent(rightString, pattern);
        }
        Matcher matcher = pattern.matcher(new MatcherInput(left, new AccessCount()));
        return BooleanTypedValue.forValue(matcher.matches());
    }
    catch (PatternSyntaxException ex) {
        throw new SpelEvaluationException(
                rightOp.getStartPosition(), ex, SpelMessage.INVALID_PATTERN, right);
    }
    catch (IllegalStateException ex) {
        throw new SpelEvaluationException(
                rightOp.getStartPosition(), ex, SpelMessage.FLAWED_PATTERN, right);
    }
}
```

其中，打印调用链的程序如下：

```
package com.davidwang456.test;

public class StackUtils {

 public static void getStack() {
  java.util.Map<Thread,StackTraceElement[]> ts = Thread.getAllStackTraces();
  StackTraceElement[] ste = ts.get(Thread.currentThread());
  int cnt = 1;
  for(int i=ste.length;i>0;i--) {
   StackTraceElement s = ste[i];
   System.out.println("调用序号："+cnt+" 调用类和方法 "+s.getClassName()+"$"+s.getLineNumber());
   cnt++;
  }
 }
}
```

正则运算需要借助 OperatorMatches 实现，此时打印出完整的调用链，如下所示：

> 调用序号：1 调用类和方法 com.davidwang456.test.SpELTest$main
>
> 调用序号：2 调用类和方法 org.springframework.context.support.ClassPathXmlApplicationContext$<init>
>
> 调用序号：3 调用类和方法 org.springframework.context.support.ClassPathXmlApplicationContext$<init>
>
> 调用序号：4 调用类和方法 org.springframework.context.support.AbstractApplicationContext$refresh

调用序号：5 调用类和方法 org.springframework.context.support.AbstractApplicationContext$finishBeanFactoryInitialization

调用序号：6 调用类和方法 org.springframework.beans.factory.support.DefaultListableBeanFactory$preInstantiateSingletons

调用序号：7 调用类和方法 org.springframework.beans.factory.support.AbstractBeanFactory$getBean

调用序号：8 调用类和方法 org.springframework.beans.factory.support.AbstractBeanFactory$doGetBean

调用序号：9 调用类和方法 org.springframework.beans.factory.support.DefaultSingletonBeanRegistry$getSingleton

调用序号：10 调用类和方法 org.springframework.beans.factory.support.AbstractBeanFactory$Lambda$40/1151020327$getObject

调用序号：11 调用类和方法 org.springframework.beans.factory.support.AbstractBeanFactory$lambda$0

调用序号：12 调用类和方法 org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$createBean

调用序号：13 调用类和方法 org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$doCreateBean

调用序号：14 调用类和方法 org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory$populateBean

调用序号：15 调用类和方法 org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$postProcessProperties

调用序号：16 调用类和方法 org.springframework.beans.factory.annotation.InjectionMetadata$inject

调用序号：17 调用类和方法 org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement$inject

调用序号：18 调用类和方法 org.springframework.beans.factory.support.DefaultListableBeanFactory$resolveDependency

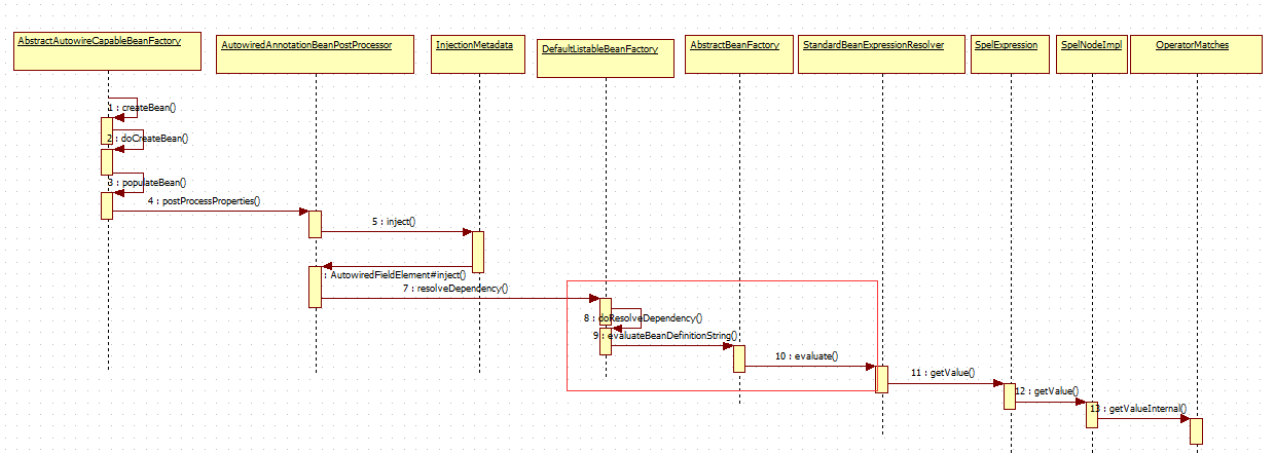
调用序号：19 调用类和方法 org.springframework.beans.factory.support.DefaultListableBeanFactory$doResolveDependency

调用序号：20 调用类和方法 org.springframework.beans.factory.support.AbstractBeanFactory$evaluateBeanDefinitionString

调用序号：21 调用类和方法 org.springframework.context.expression.StandardBeanExpressionResolver$evaluate

> 调用序号：22 调用类和方法 org.springframework.expression.spel.standard.SpelExpression$getValue
>
> 调用序号：23 调用类和方法 org.springframework.expression.spel.ast.SpelNodeImpl$getValue
>
> 调用序号：24 调用类和方法 org.springframework.expression.spel.ast.OperatorMatches$getValueInternal
>
> 调用序号：25 调用类和方法 org.springframework.expression.spel.ast.OperatorMatches$getValueInternal
>
> 调用序号：26 调用类和方法 com.davidwang456.test.StackUtils$getStack
>
> 调用序号：27 调用类和方法 java.lang.Thread$getAllStackTraces

为了方便，整理出完整的时序流程图：



总结

SpEL 表达式，能够以一种强大和简洁的方式将值装配到 bean 属性和构造器参数中，在这个过程中所使用的表达式会在运行时计算得到值。特性包括：

- 使用 bean 的 ID 来引用 bean；
- 使用方法和访问对象的属性；
- 对值进行算术、关系和逻辑运算；
- 正则表达式匹配；
- 集合操作。

SpEL 表达式要放到“#{…}”中。

}

35 SpEL 操作符应用示例及背后原理探究

37 SpEL Bean方法属性引用原理