Spring EL 表达式浅析

前言

表达式语言，是一种以语义化的方式，从特定的表达式中提取想要的数据。比如我们使用注解实现锁的时候，通过暴露表达式参数，让业务层使用表达式语言，从而解析出唯一的业务Key。

表达式语言很多，而 Spring EL 表达式无疑是我们最常使用的一种，灵活、高效 是它的代名词，在各种复杂的场景下都有其身影。

总之，表达式语言便捷性，让我们实际开发中，越来越方便：

1. 动态性和灵活性：在运行时动态计算值，而不是在编译时确定。这种动态性使得应用程序可以根据不同的条件和上下文动态调整行为和配置
2. 简化配置：通过使用表达式，可以避免在配置文件中编写冗长的静态值和复杂的逻辑。
3. 统一配置和逻辑：统一的方式来处理配置和逻辑，使得在不同的配置文件、注解和代码中可以使用相同的表达式语法
4. 增强可读性：表达式语言通常比等效的编程代码更简洁明了，开发者可以更容易地理解配置的意图。
5. 支持复杂逻辑：表达式语言支持复杂的逻辑运算、集合操作和函数调用，我们可以在配置文件和模板中可以实现复杂的逻辑，而不需要编写额外的代码。
6. ...

Spring EL 表达式

Spring EL（Spring Expression Language）是一种很强大的表达式语言，可以在 Spring 应用程序中动态地解析和计算表达式。

它可以用于 XML 配置、注解和 Java 代码中。Spring EL 支持变量、函数调用、对象属性访问、集合操作等。

功能特性

Spring EL 表达式的功能已经十分完善，除了我们常见的从 Bean 提取属性值、调用方法等，还有很多的特性，如：

* 字符表达式
* 布尔和关系操作符
* 正则表达式
* 类表达式
* 访问 properties，arrays，lists，maps 等集合
* 方法调用
* 关系操作符
* 赋值
* 调用构造器
* Bean对象引用
* 创建数组
* 内联 lists
* 内联 maps
* 三元操作符
* 变量
* 用户自定义函数
* 集合投影
* 集合选择
* 模板表达式

场景使用

---

1. 访问Bean属性（字段）

```
public class User {
    private String name;

    // getter and setter
}
```

在Spring配置文件中，可以使用Spring EL访问name属性：

```
<bean id="user" class="com.example.User">
    <property name="name" value="John Doe"/>
</bean>

<bean id="userName" class="java.lang.String">
    <constructor-arg value="#{user.name}"/>
</bean>
```

2. 调用方法

```
public class User {
    private String firstName;
    private String lastName;

    // getters and setters

    public String getFullName() {
        return firstName + " " + lastName;
    }
}
```

xml配置 中调用

```
<bean id="user" class="com.example.User">
    <property name="firstName" value="John"/>
    <property name="lastName" value="Doe"/>
</bean>

<bean id="fullName" class="java.lang.String">
    <constructor-arg value="#{user.fullName}"/>
</bean>
```

3. 内置函数

Spring EL提供了一些内置函数，例如T()用于访问类的静态方法：

```
<bean id="randomNumber" class="java.lang.Integer">
    <constructor-arg value="#{T(java.lang.Math).random() * 100}"/>
</bean>
```

4. 条件运算

可以使用三元运算符进行条件判断：

```
<bean id="isAdult" class="java.lang.Boolean">
    <constructor-arg value="#{user.age > 18 ? true : false}"/>
</bean>
```

5. 集合操作

```
public class User {
    private List<String> roles;

    // getter and setter
}
```

在Spring配置文件中，可以访问集合中的元素：

```
<bean id="user" class="com.example.User">
    <property name="roles">
        <list>
            <value>ADMIN</value>
```

```
            <value>USER</value>
        </list>
    </property>
</bean>

<bean id="firstRole" class="java.lang.String">
    <constructor-arg value="#{user.roles[0]}"/>
</bean>
```

6. 注解运用

Spring EL也可以用于注解中，例如在@Value注解中：

```
@Component
@Data
public class MyComponent {

    @Value("#{user.name}")
    private String userName;

    @Value("#{user.age > 18 ? 'Adult' : 'Minor'}")
    private String userStatus;

}
```

7. 表达式参数提取

在很多如锁、重复检测等注解实现的场景下，对业务层只需要留下一个 key 的口子，业务层可以使用表达式语言，提取对应接口的参数，从而达到目的：

```
    @Check(
            key = "#param.userCode + '_' + #param.name + '_' + #param.opsCode",
            expireTime = 4000, timeUnit = TimeUnit.MILLISECONDS,
            message = "请勿频繁操作！"
    )
```

```
    @PostMapping("/user")
    public ApiResult<Boolean> user(@RequestBody @Valid UserParam
param) {
        // ...
    }

```

以上是 Spring EL 表达式的基本使用，其强大之处在于它的灵活性和广泛的应用场景，可以帮助我们在 Spring 应用程序中实现动态配置和复杂的逻辑处理。

工作原理

核心流程：

* 创建解析器：SpEL 使用 ExpressionParser 接口表示解析器，提供 SpelExpressionParser 默认实现；
* 解析表达式：使用 ExpressionParser 的 parseExpression 来解析相应的表达式为 Expression 对象。
* 构造上下文：准备比如变量定义等等表达式需要的上下文数据。
* 求值：通过 Expression 接口的 getValue 方法根据上下文获得表达式值。
* 根对象#root
* 指向当前表达式正在求值的对象#this

核心能力

1. EvaluationContext

EvaluationContext 是用于解析表达式中的属性、字段、方法和类型转换。有两个类实现了该接口。

* SimpleEvaluationContext：只支持 SpEL 语言语法的一个子集，支持部分功能。不包括类型引用，构造函数和bean引用。
* StandardEvaluationContext：公开全套 SpEL 语言功能和配置项。可以使用它来指定默认的根对象并配置每个可用的评估相关策略。

2. 类型转换

SpEL 中默认使用的转换器是 spring-core 中的(org.springframework.core.convert.ConversionService) 这个转换服务中提供了很多通用的 converters 也支持自定义转换器。

3. 解析器配置

解析器配置对象(org.springframework.expression.spel.SpelParserConfiguration)配置解析器。

配置对象可以控制一些表达式组件的行为。如：数组或者集合元素查找的时候如果当前位置对应的对象是 Null，可以通过事先配置来自动创建元素。

这个在表达式是由多次属性链式引用的时候比较重要。如果设置的数组或者List位置越界时可以自动增加数组或者List长度来兼容。

SpEl编译器的行为也可以通过配置来实现。

4. SpEL编译

spring 在 4.1 版本中包含了一个基本表达式编译器。表达式提供了比较大灵活性，但在性能上表现一般。

在表达式动态性需求不强时，新的 SpEL 编译器在此方面进行了定制。编译器会在求值同时创建一个类来包含表达式的行为，利用这种方式来达到更快的表达式求值速度，SpelCompiler#createExpressionClass创建类。

5. 编译器配置

编译器默认是关闭的，编译器有 3种操作模式，定义在枚举(org.springframework.expression.spel.SpelCompilerMode)

* OFF – 编译器关闭。
* IMMEDIATE – 即时成效模式，表示式会被尽快的被编译。表达式会在第一次求值后马上就会执行。
* MIXED – 混合模式，表达式会在解释器模式和编译器模式之间切换。在发生了几次解释执行后会切换到编译执行。如果编译模式出错了，则会切换成解释

执行。

6. 编译执行的局限性

编译执行还有以下四种方式不被支持

* 涉及到赋值的表达式
* 依赖于类型转换服务的表达式
* 使用到自定义解析器或访问器的表达式
* 使用到选择器或者投影的表达式(expressions using selection or projection，类似在sql中中projection是返回列，selection是where条件)