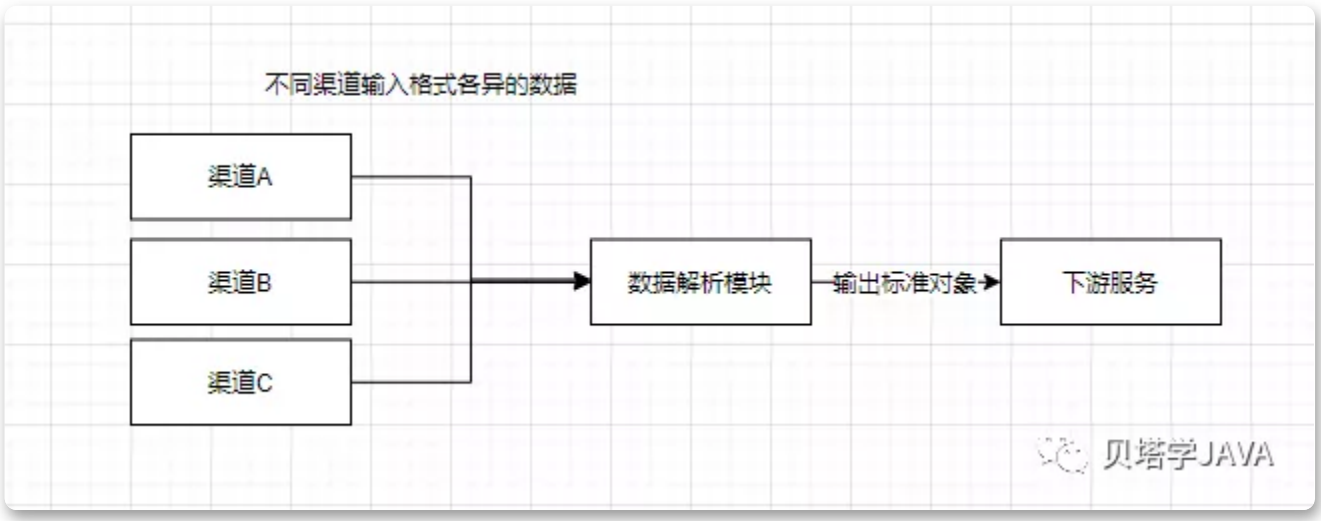
SpEL这么香的功能都没有使用过，还敢说玩转Spring?

原创 Silently9527 贝塔学JAVA 1月25日

> 本文已被Github仓库收录 https://github.com/silently9527/JavaCore

前言

最近工作中接到一个需求，需要对接第三方公司的API接口，由于每个公司提供的数据格式都不一样以及后期可能会对接更多的公司，我们给出的技术方式是设计一个数据解析模块，把数据结构各异的解析成我们内部定义的通用数据对象，如果后期再新增第三方公司的接口，我们只需要在后台配置好数据的解析规则即可完成对接



我们调研过表达式语言OGNL、SpEL（表达式语言不止这两种），由于考虑到我们项目本身都是依赖于Spring的，所以选择了SpEL

SpEL介绍以及功能概述

SpEL是spring提供的强大的表达式语言，本身也是作为了Spring的基石模块，在Spring的很多模块中都是使用到；虽然SpEL是Spring的基石，但是完全脱离Spring独立使用。SpEL提供的主要功能：

- 文字表达式
- 布尔和关系运算符
- 正则表达式
- 类表达式
- 访问 properties, arrays, lists, maps
- 方法调用
- 关系运算符
- 参数
- 调用构造函数
- Bean引用
- 构造Array
- 内嵌lists、内嵌maps
- 三元运算符
- 变量
- 用户定义的函数
- 集合投影
- 集合筛选
- 模板表达式

SpEL表达语言初级体验

下面的代码使用SpEL API来解析文本字符串表达式 Hello World.

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("'Hello World'");
String message = (String) exp.getValue();
```

接口 `ExpressionParser` 负责解析表达式字符串，在表达式中表示字符串需要使用单引号

SpEL支持很多功能特性，如调用方法，访问属性，调用构造函数。

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("'Hello World'.concat('!')");
String message = (String) exp.getValue();
```

SpEL还支持使用标准的“.”符号，即嵌套属性prop1.prop2.prop3和属性值的设置

```
ExpressionParser parser = new SpelExpressionParser();

// invokes getBytes().length
Expression exp = parser.parseExpression("'Hello World'.bytes.length");
int length = (Integer) exp.getValue();
```

也可以使用 `public <T> T getValue(Class<T> desiredResultType)` 来获取返回指定类型的结果，不用强制类型转换，如果实际的结果不能转换成指定的类型就会抛出异常 `EvaluationException`

```
ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("new String('hello world').toUpperCase()");
String message = exp.getValue(String.class);
```

SpEL比较常见的用途是针对一个特定的对象实例（称为root object）提供被解析的表达式字符串．这种方式也是我们项目中使用的方式，把不同渠道传入的json数据设置成root object，然后指定字符串表达式对每个标准字段进行解析

```
GregorianCalendar c = new GregorianCalendar();
c.set(1856, 7, 9);

// The constructor arguments are name, birthday, and nationality.
Inventor tesla = new Inventor("Nikola Tesla", c.getTime(), "Serbian");

ExpressionParser parser = new SpelExpressionParser();
Expression exp = parser.parseExpression("name");

EvaluationContext context = new StandardEvaluationContext(tesla);
String name = (String) exp.getValue(context);
```

EvaluationContext 接口

接口 `EvaluationContext` 有一个开箱即用的实现 `StandardEvaluationContext` ；当计算表达式解析属性,方法并帮助执行类型转换时，使用反射来操纵对象并且缓存java.lang.reflect的Method，Field，和Constructor实例提高性能

`StandardEvaluationContext` 可以通过 `setRootObject()` 或者传递root object到构造函数来设置root object，也可以使用 `setVariable()` 来设置变量， `registerFunc`

tion() 来注册方法；我们还可以通过自定义 ConstructorResolvers， MethodResolvers， PropertyAccessor 来扩展SpEL的功能。

解析器配置

可以通过使用 org.springframework.expression.spel.SpelParserConfiguration 去精细化配置解析器功能，例如：如果在表达式中指定的数组或集合的索引位置值为null，就让它自动地创建的元素；如果索引超出数组的当前大小就去自动扩容

```
class Demo {
    public List<String> list;
}

// Turn on:
// - auto null reference initialization
// - auto collection growing
SpelParserConfiguration config = new SpelParserConfiguration(true,true);

ExpressionParser parser = new SpelExpressionParser(config);

Expression expression = parser.parseExpression("list[3]");

Demo demo = new Demo();

Object o = expression.getValue(demo);
```

语言参考

文字表达式

支持文字表达的类型是字符串，日期，数值（int， real，十六进制），布尔和空。字符串使用单引号分隔。一个单引号本身在字符串中使用两个单引号字符表示。下面的列表 显示文字的简单用法。数字支持使用负号，指数符号和小数点

```
ExpressionParser parser = new SpelExpressionParser();

// evals to "Hello World"
String helloWorld = (String) parser.parseExpression("'Hello World'").getValue();

double avogadrosNumber = (Double) parser.parseExpression("6.0221415E+23").getValue();

// evals to 2147483647
int maxValue = (Integer) parser.parseExpression("0x7FFFFFFF").getValue();

boolean trueValue = (Boolean) parser.parseExpression("true").getValue();
```

```
Object nullValue = parser.parseExpression("null").getValue();
```

Properties, Arrays, Lists, Maps

用属性引用很简单：只要用一个 . 表示嵌套属性值。

```
// evals to 1856
int year = (Integer) parser.parseExpression("Birthdate.Year + 1900").getValue(context);

String city = (String) parser.parseExpression("placeOfBirth.City").getValue(context);
```

数组和列表使用方括号获得内容

```
ExpressionParser parser = new SpelExpressionParser();

// Inventions Array
StandardEvaluationContext teslaContext = new StandardEvaluationContext(tesla);

// evaluates to "Induction motor"
String invention = parser.parseExpression("inventions[3]").getValue(
        teslaContext, String.class);

// Members List
StandardEvaluationContext societyContext = new StandardEvaluationContext(ieee);

// evaluates to "Nikola Tesla"
String name = parser.parseExpression("Members[0].Name").getValue(
        societyContext, String.class);

// List and Array navigation
// evaluates to "Wireless communication"
String invention = parser.parseExpression("Members[0].Inventions[6]").getValue(
        societyContext, String.class);
```

maps的内容通过在方括号中指定key值获得.

```
// Officer's Dictionary

Inventor pupin = parser.parseExpression("Officers['president']").getValue(
        societyContext, Inventor.class);
```

```
// evaluates to "Idvor"
String city = parser.parseExpression("Officers['president'].PlaceOfBirth.City").getValue(
        societyContext, String.class);

// setting values
parser.parseExpression("Officers['advisors'][0].PlaceOfBirth.Country").setValue(
        societyContext, "Croatia");
```

调用方法

表达式中依然支持方法调用

```
// string literal, evaluates to "bc"
String c = parser.parseExpression("'abc'.substring(2, 3)").getValue(String.class);

// evaluates to true
boolean isMember = parser.parseExpression("isMember('Mihajlo Pupin')").getValue(
        societyContext, Boolean.class);
```

> 这里使用到的方法 `isMember` 是root object中的方法，如果root object不存在这个方法会报错

运算符

关系运算符：等于，不等于，小于，小于或等于，大于， 和大于或等于的使用方式和正常Java中使用一致

```
// evaluates to true
boolean trueValue = parser.parseExpression("2 == 2").getValue(Boolean.class);

// evaluates to false
boolean falseValue = parser.parseExpression("2 < -5.0").getValue(Boolean.class);
```

除了标准的关系运算符SpEL支持 `instanceof` 和正则表达式的 `matches` 操作。

```
boolean falseValue = parser.parseExpression(
        "'xyz' instanceof T(int)").getValue(Boolean.class);

// evaluates to true
boolean trueValue = parser.parseExpression(
        "'5.00' matches '^-?\\d+(\\.\\d{2})?$'").getValue(Boolean.class);
```

逻辑运算符：支持的逻辑运算符 `and`， `or`， `not`.

```
/ -- AND --

// evaluates to false
boolean falseValue = parser.parseExpression("true and false").getValue(Boolean.class);

// evaluates to true
String expression = "isMember('Nikola Tesla') and isMember('Mihajlo Pupin')";
boolean trueValue = parser.parseExpression(expression).getValue(societyContext, Boolean.cla

// -- OR --

// evaluates to true
boolean trueValue = parser.parseExpression("true or false").getValue(Boolean.class);

// evaluates to true
String expression = "isMember('Nikola Tesla') or isMember('Albert Einstein')";
boolean trueValue = parser.parseExpression(expression).getValue(societyContext, Boolean.cla

// -- NOT --

// evaluates to false
boolean falseValue = parser.parseExpression("!true").getValue(Boolean.class);

// -- AND and NOT --
String expression = "isMember('Nikola Tesla') and !isMember('Mihajlo Pupin')";
boolean falseValue = parser.parseExpression(expression).getValue(societyContext, Boolean.cl
```

#this 和 #root变量

变量#this始终定义和指向的是当前的执行对象。变量#root指向root context object。

```
// 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);
```

函数

有时候你可能需要自定义一些工具函数在表达式中使用，可以先使用 `StandardEvaluationContext.registerFunction` 方法注册函数，然后在表达式中调用函数

```
ExpressionParser parser = new SpelExpressionParser();
StandardEvaluationContext context = new StandardEvaluationContext();

context.registerFunction("reverseString",
    StringUtils.class.getDeclaredMethod("reverseString", new Class[] { String.class }));

String helloWorldReversed = parser.parseExpression(
    "#reverseString('hello')").getValue(context, String.class);
```

bean引用

如果解析上下文已经配置，那么bean解析器能够从表达式使用 `@` 符号查找bean类，使用 `&` 查到FactoryBean对象

```
ExpressionParser parser = new SpelExpressionParser();
StandardEvaluationContext context = new StandardEvaluationContext();
context.setBeanResolver(new MyBeanResolver());

// This will end up calling resolve(context,"something") on MyBeanResolver during evaluatio
Object bean = parser.parseExpression("@something").getValue(context);
```

SpEL默认已经提供了 `BeanFactoryResolver` ，无需自己手动实现

集合选择

选择是一个强大的表达式语言功能，能够通过它设置过滤条件，返回满足条件的子集合，类似于Stream中的filter；选择使用语法 `?[selectionExpression]` 。

列表的过滤事例

```
ExpressionParser parser = new SpelExpressionParser();
StandardEvaluationContext context = new StandardEvaluationContext(Arrays.asList(1, 7, 0, 3,
Object subList = parser.parseExpression("#root.?[#this>5]").getValue(context);  //[7, 9]
```

map的过滤事例

```
ExpressionParser parser = new SpelExpressionParser();
Map<String,Integer> root = new HashMap<>();
root.put("3",3);
root.put("6",6);
root.put("8",8);
StandardEvaluationContext context = new StandardEvaluationContext(root);
Object subMap = parser.parseExpression("#root.?[#this.value>5]").getValue(context); //{6=6,
```

除了返回所有选定的元素以外，也可以用来获取第一或最后一个值。获得第一条目相匹配的选择的语法是 `^[...]` 而获得最后一个匹配选择语法是 `$[...]`

集合投影

集合投影也是个强大的功能，可以把列表或map中对象的某个字段拿出来构成一个集合返回，类似于Stream中的map操作

```
ExpressionParser parser = new SpelExpressionParser();
StandardEvaluationContext context = new StandardEvaluationContext(Arrays.asList(
        new User("Silently9521"),
        new User("Silently9522"),
        new User("Silently9527")));
Object nameList = parser.parseExpression("#root.![#this.name]").getValue(context);//[Silent
```

参考官方文档：https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#expressions