Spring资源处理和SpEL语言

> ## 本节目标
>
> 1.掌握Spring资源加载 2.掌握SpEL语言的基本使用

1. SpEL语言

Spring动态表达式语言（简称：SpEL）是一个支持运行时查询和操作对象图的强大动态语言。 其语法类似于EL（`Expression Language`）表达式，具有诸如显式方法调用和基本字符串模板函数等特性。

思考：为什么要引入SpEL?

- Java是一门强类型的静态语言，所有代码在运行之前都必须进行严格的类型检查并且编译成JVM字节码，因此虽然在安全，性能方面得到了保障，但牺牲了灵活性。这个特征就决定了**Java在语言层面上无法直接进行表达式语句的动态解析。**
- 动态语言恰恰相反，其显著的特点是在程序运行时可以改变程序的结构和变量类型。典型的动态语言有`Python, Ruby, Javascript, Perl`等。这些动态语言能够广泛应用在各种领域，得益于其动态，简单，灵活的特性。
- Java在实现复杂业务系统，大型商业系统，分布式系统以及中间件等方面有着非常强的优势，在开发这些系统的过程中（如：积分规则，促销活动，游戏的技能设置），有时候需要引用动态语言的一些特性，以弥补其在动态性方面的不足。JDK1.6之后内嵌Javascript解析引擎（`NashornScriptEngine`），方便在Java中调用Javascript编写的动态脚本。
- Spring社区为了弥补Java语言动态性方面的不足，提供了Spring动态语言，其是一个支持运行时查询和操作对象图的强大的动态语言（语法类似与EL表达式，具备显式方法调用，基本字符串模板函数等特性）。

1.1 SpEL核心接口

SpEL的所有类与接口都定义在`org.springframework.expression`包及其子包，以及`spel.support`中。

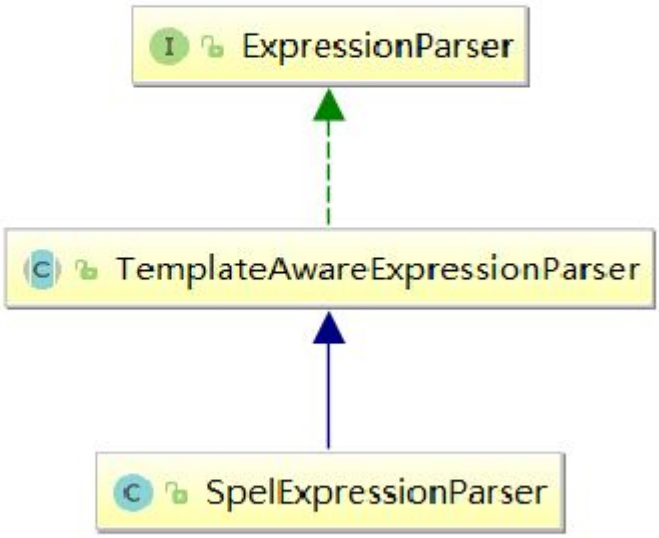
该部分内容位于Spring框架的expression模块。

```
<dependency>
     <groupId>org.springframework</groupId>
     <artifactId>spring-expression</artifactId>
</dependency>
```

1.1.1 ExpressionParse接口

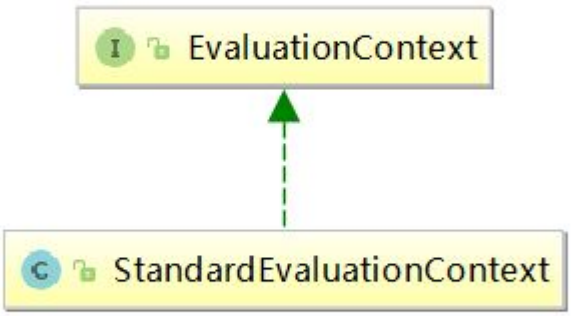
ExpressionParse接口用来解析表达式字符串，**表达式字符串是一个用单引号标注或者用转义的双引号标注的字符串。**

ExpressionParse接口的实现类：



1.1.2 EvaluationContext接口

EvaluationContext接口提供属性，方法，字段解析器以及类型转换器。 默认实现类StandardEvaluationContext的内部使用反射机制来操作对象，为了提高性能，在其会对已经获取的Method, Field和Constructor实例进行缓存。



1.2 SpEL基础表达式

1.2.1 文本字符解析

文本表达式支持字符串，日期，字符，布尔类型以及null。其中字符串需要使用单引号或者反斜杠+双引号包含起来，比如:`“'Hello World'”,"\"Hello World\""`。

```
//构造一个表达式的解析实例
ExpressionParser parser = new SpelExpressionParser();

//解析字符串
String helloWorld = (String) parser.parseExpression("\"hello world\"").getValue();

//解析布尔类型
boolean trueValue = (Boolean) parser.parseExpression("true").getValue();

//解析整数类型
Integer intValue = parser.parseExpression("120").getValue(Integer.class);
```

1.2.2 对象属性解析

```
public class User {

    /**
     * 用户名
     */
    private String userName;
    /**
     * 最近访问时间
     */
    private Date lastVisit;
    /**
     * 用户积分
     */
    private Integer credits;

    /**
     * 用户出生地
     */
    private PlaceOfBirth placeOfBirth;

    //此处省略 setter getter toString
}

public  class PlaceOfBirth {

    //国家
    private final String nation;

    //地区
    private final String district;
```

```
    public PlaceOfBirth(String nation, String district) {
        this.nation = nation;
        this.district = district;
    }

    //此处省略 setter getter toString
}

public static void objectParse() {

    //1.构造一个User对象
    User user = new User();
    user.setUserName("secondriver");
    user.setLastVisit(new Date());
    user.setCredits(88);
    user.setPlaceOfBirth(new PlaceOfBirth("中国", "咸阳"));

    //2.构造SpEl解析解析上下文
    ExpressionParser parser = new SpelExpressionParser();
    EvaluationContext context = new StandardEvaluationContext(user);

    //3.基本属性值获取
    String userName = (String) parser.parseExpression("userName").getValue(context);
    Integer credits = (Integer) parser.parseExpression("credits").getValue(context);

    //4.嵌套对象属性获取
    String district = (String)
parser.parseExpression("placeOfBirth.district").getValue(context);

    System.out.println("User.username :" + userName);
    System.out.println("User.credits :" + credits);
    System.out.println("User.placeOfBirth.district :" + district);
}
```

从上面示例中可以看出，对象属性解析与文本字符解析有所不同，对象属性解析需要在取值的时候传递一个计算上下文参数 `EvaluationContext`。在这里将 `User` 实例作为上下文的根对象传递给 `EvaluationContext`，这样SpEL表达式解析器就可以根据属性路径表达式获取上下文中根对象的属性值。

1.2.3 数组和集合解析

在SpEL中，支持数组，集合类型（Map,List)的解析。数组支持标准Java语言创建数组的方法，如：`"new int[]{1,2,3}"`。List支持大括号括起来的内容，数据项之间用逗号隔开，如 `"{1,2,3}","{{'a','b'},{'x','y'}}"`。目前SpEL还不支持多维数组初始化，如：`"new int[2][3]{{1,2,3},{4,5,6}}"`。Map采用如下表达：`"{userName:'Jack',credits:100}"`。

```
public static void collectionParse(){
    ExpressionParser parser = new SpelExpressionParser();

    //解析List
    List listValue = (List) parser.parseExpression("{1,2,3}").getValue();
    System.out.println(listValue);
```

```
    //解析Map
    Map mapValue = (Map) parser.parseExpression("{userName:'secondriver', age:
18}").getValue();
    System.out.println(mapValue);

    //解析array
    int[] arrayValue = (int[]) parser.parseExpression("new int []{1,2,3}").getValue();
    System.out.println(Arrays.toString(arrayValue));
}
```

1.2.4 方法解析

在SpEL中，方法调用支持Java可以访问的方法，包括对象方法，静态方法，并支持可变方法参数，还可以调用String类型的所有可访问的方法。

```
public static void methodParse() {
    ExpressionParser parser = new SpelExpressionParser();

    //String方法
    String subValue = (String) parser.parseExpression("'Spring
SpEL'.substring(7)").getValue();
    System.out.println(subValue);
    Integer indexValue = (Integer) parser.parseExpression("'Spring
SpEL'.indexOf('Sp')").getValue();
    System.out.println(indexValue);

    //静态方法
    //java.lang.System.currentTimeMillis()
    Long currentTimeMillis =
parser.parseExpression("T(java.lang.System).currentTimeMillis()")
                .getValue(Long.class);
    System.out.println(currentTimeMillis);
     //java.lang.Math.min()
    double minValue =
 parser.parseExpression("T(java.lang.Math).min(10,20)").getValue(double.class);
    System.out.println(minValue);

    //实例方法
    EvaluationContext context = new StandardEvaluationContext(new
SimpleDateFormat("yyyy-MM-dd HH:mm:ss"));
    //注意:表达式字符串中要创建对象需要使用类的全限定名
    String strValue = parser.parseExpression("format(new
java.util.Date())").getValue(context, String.class);
    System.out.println(strValue);
}
```

1.3 Spring中使用SpEL

1.3.1 基于XML的配置

- XML配置表达式

```
public class GuessNumber {
    private double number;
    public double getNumber() {
        return number;
    }
    public void setNumber(double number) {
        this.number = number;
    }
    @Override
    public String toString() {
        return "GuessNumber{" +
                "number=" + number +
                '}';
    }
}
```

```
<bean id="guessNumber" class="com.bittech.springcore.GuessNumber" scope="prototype">
     <property name="number" value="#{T(java.lang.Math).random()}"/>
</bean>
```

在示例中，通过SpEL提供的T类型操作符，直接调用java.lang.Math的静态方法生成一个随机数赋值给GuessNumber的number属性。

- 内置Bean `systemProperties` 和 `systemEnvironment`

Spring中内置了 `systemProperties` 和 `systemEnvironment` 两个Bean,可以通过它们分别获取系统属性变量值和系统环境变量值。

> 备注：回顾Java的两个重要方法
>
> System.getProperties()；方法获取系统属性
>
> System.getenv()；方法获取系统环境变量

```
<bean id="systemPropertiesBean" class="com.bittech.spel.SystemPropertiesBean">
    <property name="classPath" value="#{systemProperties['java.class.path']}"/>
    <property name="javaHome" value="#{systemProperties['java.home']}"/>
    <property name="javaVersion" value="#{systemProperties['java.version']}"/>
    <property name="osName" value="#{systemProperties['os.name']}"/>
</bean>

<bean id="systemEnvironmentBean" class="com.bittech.spel.SystemEnvironmentBean">
    <property name="path" value="#{systemEnvironment['Path']}"/>
    <property name="systemDrive" value="#{systemEnvironment['SystemDrive']}"/>
    <property name="appData" value="#{systemEnvironment['APPDATA']}"/>
</bean>
```

1.3.2 基于注解的配置

@Value注解可以标注在类的属性，方法，构造函数上。下面是一个从配置文件中加载一个参数值的示例。

- 配置文件

```
#配置文件: database.properties
url=jdbc:mysql://localhost:3306/memo
username=root
password=root
classname=com.mysql.jdbc.Driver
```

- 在XML中配置一个id为`properties`的bean用来加载配置文件

```
<!-- 开启自动扫描 -->
<context:component-scan base-package="com.bittech.example"/>

<util:properties id="properties" location="database.properties"/>
```

- 在Bean中通过@Value注解注入值

```
@Component
public class MyDataSource {

    @Value(value = "#{properties['url']}")
    private String url;

    @Value(value = "#{properties['username']}")
    private String username;

    @Value("#{properties['password']}")
    private String password;

    @Value("#{properties['classname']}")
    private String classname;

    //此处省略 setter getter toString
}
```

上面采用的方式略显复杂，Spring提供了一种更加简洁的写法——属性占位符。接下来我们讲解该用法。

2. 资源配置文件

我们在配置Bean的属性时，大多数情况下值基本时固定的。这样就存在一个问题，在配置例如数据库的用户名，密码；邮箱的用户名，密码；以及一些编码阶段不可预知的信息（支付宝的AppId,私钥，公钥等），这个时候我们就需要外置资源文件来解决这个问题。

比如我们的数据库配置外置文件（database.properties）内容如下：

```
jdbc.url=jdbc:mysql://127.0.0.1:3306/product_dev?characterEncoding=UTF-
8&createDatabaseIfNotExist=true
jdbc.username=root
jdbc.password=root
```

2.1 使用PropertyPlaceholderConfigure配置属性文件

- 单个资源文件：

```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="fileEncoding" value="UTF-8"/>
    <property name="location" value="classpath:database.properties"/>
</bean>
```

- 多个资源文件：

```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="locations">
        <list>
            <!-- 文件系统的文件 -->
            <value>file:D:\db.properties</value>
            <!-- classpath的文件 -->
            <value>classpath:database.properties</value>
        </list>
    </property>
    <property name="fileEncoding" value="UTF-8"/>
</bean>
```

2.2 使用`<context:property-placeholder>`配置属性文件

```
<context:property-placeholder file-encoding="UTF-8"
location="classpath:database.properties"/>
```

以上配置相当于在Spring容器中配置了一个`PropertyPlaceholderConfigurer`的Bean，显然这种方式更为优雅。

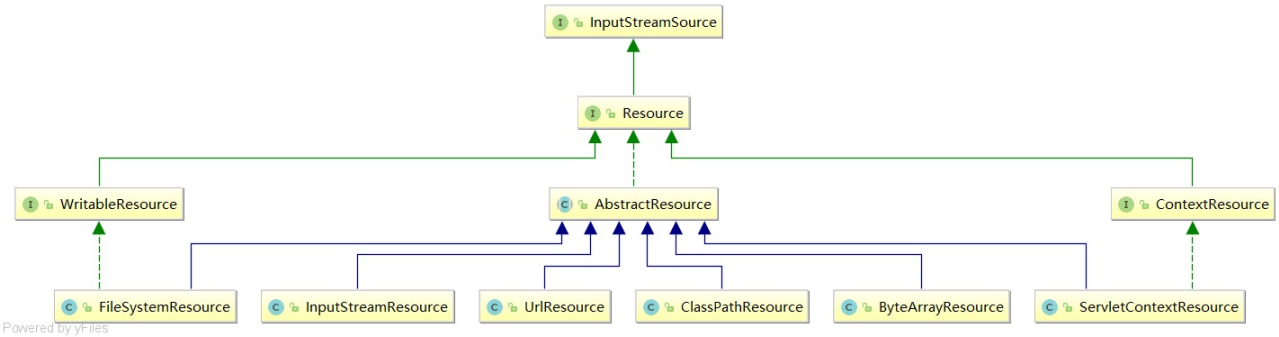
2.3 使用@Value注解给Bean的属性注入值

```
@Component
public class MyDataSource {

    @Value("${jdbc.url}")
    private String url;

    @Value("${jdbc.username}")
    private String username;

    @Value("${jdbc.password}")
    private String password;
}
```

2.4 Spring资源接口

我们在使用Spring框架过程中，离不开和各类资源文件打交道，这些资源文件有来自本地文件，有来自一个jar包，也有来自一个URL。因为JDK操作底层资源基本是从 `java.net.URL, java.io.File, java.util.Properties` 这些类开始，获取资源要么是从绝对路径，要么就是当前类的相对路径。从类路径和web容器上下文中获取资源是很不方便的。 Spring的Resource接口提供了统一的访问底层资源的能力。

Resource接口的实现类结构图:



> 备注：课堂上通过源码分析的方式开展Resource接口以及实现类的讲解，主要目的是了解面向接口编程，策略模式。

- `ByteArrayResource`： 代表 `byte[]` 数组资源， `getInputStream` 将返回一个 `ByteArrayInputStream` 。 `ByteArrayResource` 可多次读取数组资源，即 `isOpen ()` 永远返回 `false` 。
- `InputStreamResource`： 代表 `java.io.InputStream` 字节流，对于 `getInputStream` 操作将直接返回该字节流，因此只能读取一次该字节流，即 `isOpen` 永远返回 `true` 。
- `FileSystemResource`： 代表 `java.io.File` 资源，对于 `getInputStream` 操作将返回底层文件的字节流， `isOpen` 将永远返回 `false` ，从而表示可多次读取底层文件的字节流。
- `ClassPathResource`：代表 `classpath` 路径的资源，将使用 `ClassLoader` 进行加载资源。classpath 资源存在于类路径中的文件系统中或jar包里，且 `isOpen` 永远返回 `false` ，表示可多次读取资源。 `ClassPathResource` 加载资源替代了Class类和ClassLoader类的 `getResource(String name)` 和 `getResourceAsStream(String name)` 两个加载类路径资源方法，提供一致的访问方式。

了解Spring资源接口实现类之后，可以通过文件名的前缀来动态决定加载资源时使用的实现类（实际Spring也是通过这种方式来实现资源的加载时策略的选择）。

比如:

```
public static Resource loadResource(String filename) {
    if (filename == null || filename.length() == 0) {
        return null;
    }
    if (filename.startsWith("file:")) {
        return new FileSystemResource(filename.substring("file:".length()));
    }
    if (filename.startsWith("classpath:")) {
        return new ClassPathResource(filename.substring("classpath:".length()));
    }
    if (filename.startsWith("http") || filename.startsWith("ftp")) {
        try {
            return new UrlResource(filename);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
```

```
        }
        return null;
    }
```

总结

知识块	知识点	分类	掌握程度
SpEL	1.SpEL核心接口 2. SpEL基本使用	实战型	掌握
资源配置	1. 资源接口实现类 2. 属性文件配置	实战型	掌握