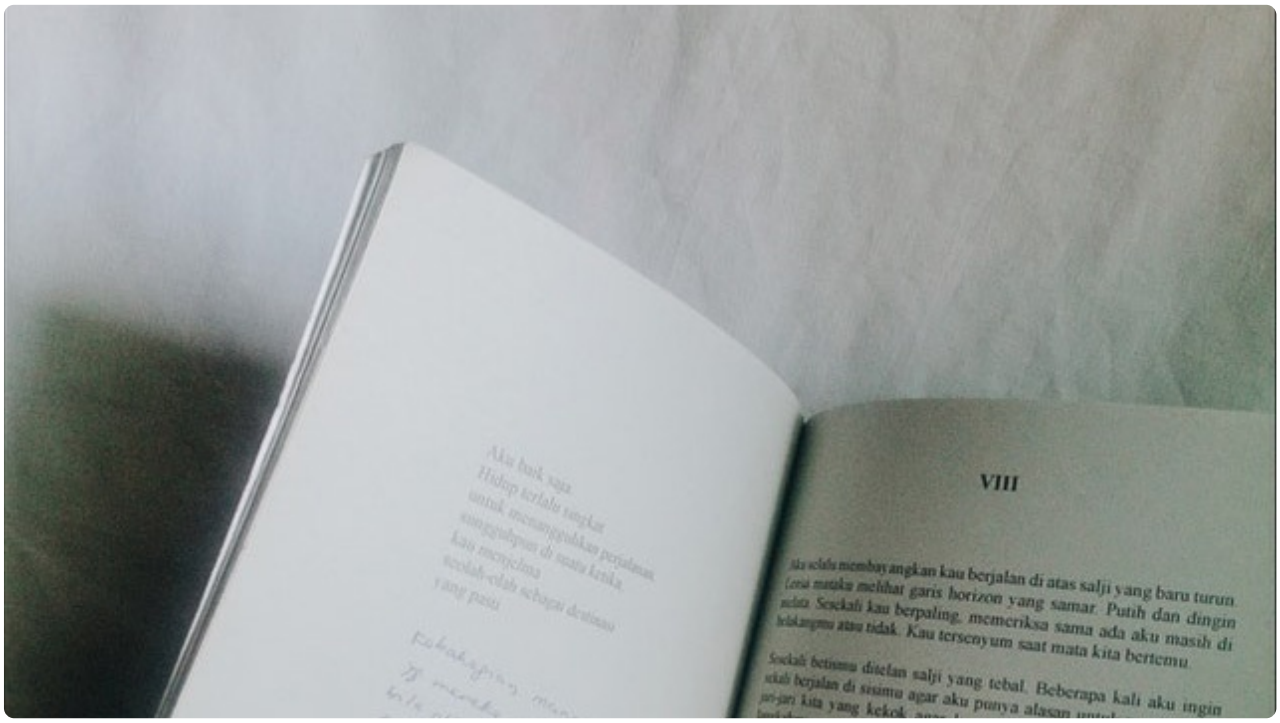
37 SpEL Bean方法属性引用原理

更新时间：2020-08-14 10:42:24



> 生活永远不像我们想像的那样好，但也不会像我们想像的那样糟。——莫泊桑

背景

Spring 容器中，一个 bean 可以引用另一个 bean：

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns="http://www.springframework.org/schema/beans"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">
       <bean class="Bean">
              <property name="a" value="呵呵" />
       </bean>
       <bean class="MySpring">
              <property name="Bean" ref="Bean" />
       </bean>
</beans>
```

那么，如果一个 bean 想要引用另外一个 bean 的属性该如何做呢？

Spring 中的 bean 可以直接引用其它 bean 的属性值来赋值给当前 bean 的属性，也可以直接调用其它 bean 的方法获取返回值来赋值给当前 bean 的属性，并且可以进行参数传递，这样可以省去在 bean 中注入需要获取属性值的 bean。

我们就来看看示例吧！

SpEL Bean 引用应用示例

被引用的 **Bean**：

```java
package com.davidwang456.test;


import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

@Component("addressBean")
public class Address {
    @Value("Sh")
    private String street;
    @Value("123006")
    private int postcode;
    @Value("china")
    private String country;

    public String getStreet() {
        return street;
    }
    public void setStreet(String street) {
        this.street = street;
    }
    public int getPostcode() {
        return postcode;
    }
    public void setPostcode(int postcode) {
        this.postcode = postcode;
    }
    public String getCountry() {
        return country;
    }
    public void setCountry(String country) {
        this.country = country;
    }
}
```

引用的 **Bean**：

```java
package com.davidwang456.test;

import org.springframework.beans.factory.annotation.Value;

@Component("studentBean")
public class StudentBean {
    @Value("Hello")
    private String name;

    @Value("#{addressBean.street}")
    private String street;

    @Value("#{addressBean.country}")
    private String country;

    @Value("#{addressBean.postcode}")
    private String postcode;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public String getStreet() {
        return street;
    }
    public void setStreet(String street) {
        this.street = street;
    }

    public String getCountry() {
        return country;
    }
    public void setCountry(String country) {
        this.country = country;
    }

    public String getPostcode() {
        return postcode;
    }
    public void setPostcode(String postcode) {
        this.postcode = postcode;
    }

}
```

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：

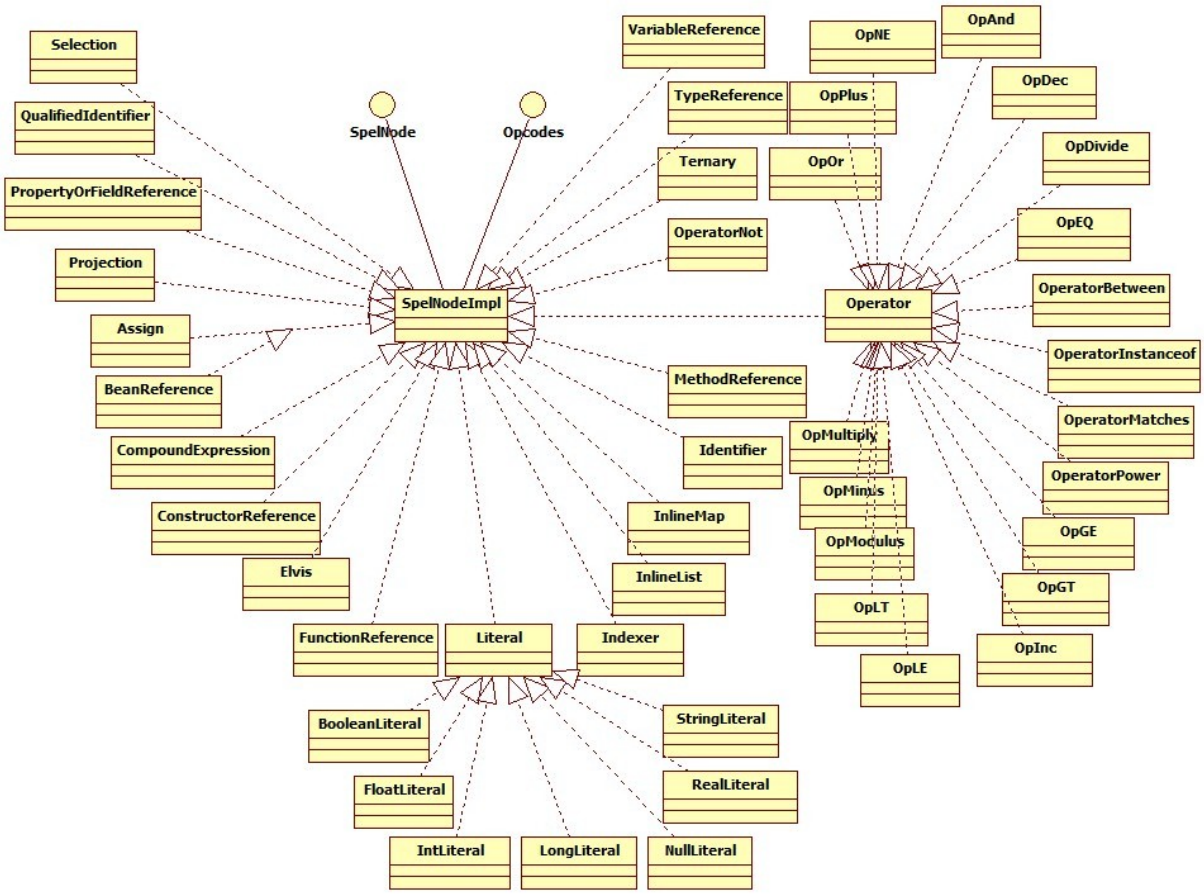
```java
package com.davidwang456.test;

import org.springframework.context.ApplicationContext;

public class SpELTest {

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

         StudentBean student = context.getBean(StudentBean.class);

         System.out.println("Name: "+student.getName());
         System.out.println("Street: "+student.getStreet());
         System.out.println("Postal: "+student.getPostcode());
         System.out.println("Country: "+student.getCountry());
    }

}
```

运行后结果如下：

> Name: Hello
>
> Street: Sh
>
> Postal: 123006
>
> Country: china

深入 **SpEL Bean** 引用应用原理

通过以前的章节我们知道，所有节点值的获取都是通过 SpelNodeImpl 实现类的 getValue() 方法实现。Bean 内部属性的应用，通过推测是 PropertyOrFieldReference 的 getValueInternal() 方法实现的。



为了验证这个猜测，可以在 PropertyOrFieldReference 的 getValueInternal() 方法打出调用链接。

```
@Override
public TypedValue getValueInternal(ExpressionState state) throws EvaluationException {
    StackUtils.getStack();  // 打印调用链程序
    TypedValue tv = getValueInternal(state.getActiveContextObject(), state.getEvaluationContext(),
            state.getConfiguration().isAutoGrowNullReferences());
    PropertyAccessor accessorToUse = this.cachedReadAccessor;
    if (accessorToUse instanceof CompilablePropertyAccessor) {
        CompilablePropertyAccessor accessor = (CompilablePropertyAccessor) accessorToUse;
        setExitTypeDescriptor(CodeFlow.toDescriptor(accessor.getPropertyType()));
    }
    return tv;
}
```

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-1;i>0;i--) {
                 StackTraceElement s=ste[i];
                 System.out.println("调用序号: "+cnt+"  调用类和方法 "+s.getClassName()+"$"+s.getMethodName());
                 cnt++;
             }
    }

}
```

此时打印出完整的调用链，如下所示。PropertyOrFieldReference.java#getValueInternal()

> 调用序号：1 调用类和方法 com.davidwang456.test.SpELTest$main
>
> 调用序号：2 调用类和方法 org.springframework.context.support.ClassPathXmlApplicationContext$

调用序号：3 调用类和方法 org.springframework.context.support.ClassPathXmlApplicationContext$

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$40/606548741getObject

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

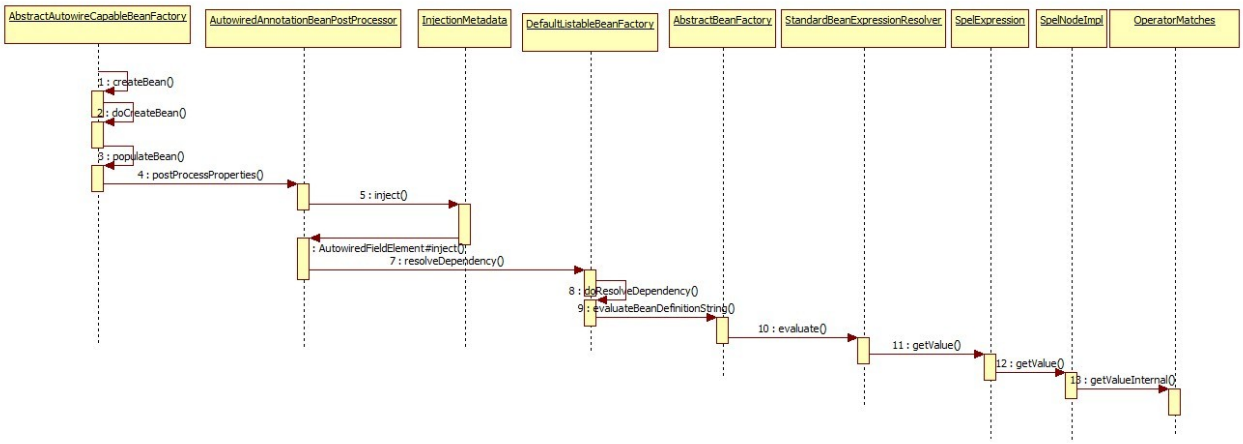
调用序号：18 调用类和方法 org.springframework.beans.factory.support.DefaultListableBeanFactory$resolveDependency

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$getValue
>
> 调用序号：24 调用类和方法 org.springframework.expression.spel.ast.CompoundExpression$getValueInternal
>
> 调用序号：25 调用类和方法 org.springframework.expression.spel.ast.CompoundExpression$getValueRef
>
> 调用序号：26 调用类和方法 org.springframework.expression.spel.ast.PropertyOrFieldReference$getValueInternal

为了方便查看，整理出完整流程的时序图如下：



我们发现入口程序是在 AbstractBeanFactory.java#evaluateBeanDefinitionString() 方法。

总结

如果我们的应用是以集群的方式部署，或者我们希望在运行期间能够动态调整引用的某些配置值，这时，就必须将配置信息放到数据库。因为这样不但方便集中管理，而且可以通过应用系统的管理界面对其进行动态维护，从而可以有效地增强应用系统的可维护性 。

Spring3+ 中，我们可以通过 #{beanName.propertyName} 的方式来引用另外一个 Bean 的属性值 。

> **Tips**：格式中是 #{xxx}，而不是 ${xxx}。

在基于注解或基于 Java 类的 Bean 配置中，可以通过 @Value("#{beanName.propertyName}") 的注解形式来引用其它 Bean 的属性值 。

}