Spring 基于构造函数的依赖注入

Spring 基于构造函数的依赖注入

当容器调用带有一组参数的类构造函数时，基于构造函数的 DI 就完成了，其中每个参数代表一个对其他类的依赖。

接下来，我们将通过示例来理解 Spring 基于构造函数的依赖注入。

示例：

下面的例子显示了一个类 TextEditor，只能用构造函数注入来实现依赖注入。

让我们用 Eclipse IDE 适当地工作，并按照以下步骤创建一个 Spring 应用程序。

步骤	描述
1	创建一个名为 *SpringExample* 的项目，并在创建的项目中的 src 文件夹下创
2	使用 *Add External JARs* 选项添加必需的 Spring 库，解释见 *Spring Hello*
3	在 *com.tutorialspoint* 包下创建 Java 类 *TextEditor*，*SpellChecker* 和
4	在 src 文件夹下创建 Beans 的配置文件 *Beans.xml* 。
5	最后一步是创建所有 Java 文件和 Bean 配置文件的内容并按照如下所示的方法运

这是 **TextEditor.java** 文件的内容：

```
package com.tutorialspoint;
public class TextEditor {
   private SpellChecker spellChecker;
   public TextEditor(SpellChecker spellChecker) {
      System.out.println("Inside TextEditor constructor." );
      this.spellChecker = spellChecker;
   }
   public void spellCheck() {
      spellChecker.checkSpelling();
   }
}
```

下面是另一个依赖类文件 **SpellChecker.java** 的内容：

```
package com.tutorialspoint;
public class SpellChecker {
   public SpellChecker(){
      System.out.println("Inside SpellChecker constructor." );
   }
   public void checkSpelling() {
      System.out.println("Inside checkSpelling." );
   }
}
```

以下是 **MainApp.java** 文件的内容：

```
package com.tutorialspoint;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class MainApp {
   public static void main(String[] args) {
      ApplicationContext context =
             new ClassPathXmlApplicationContext("Beans.xml");
      TextEditor te = (TextEditor) context.getBean("textEditor");
      te.spellCheck();
   }
}
```

下面是配置文件 **Beans.xml** 的内容，它有基于构造函数注入的配置：

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

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

   <!-- Definition for textEditor bean -->
   <bean id="textEditor" class="com.tutorialspoint.TextEditor">
      <constructor-arg ref="spellChecker"/>
   </bean>

   <!-- Definition for spellChecker bean -->
   <bean id="spellChecker" class="com.tutorialspoint.SpellChecker">
   </bean>
```

```
</beans>
```

当你完成了创建源和 bean 配置文件后，让我们开始运行应用程序。如果你的应用程序运行顺利的话，那么将会输出下述所示消息：

```
Inside SpellChecker constructor.
Inside TextEditor constructor.
Inside checkSpelling.
```

构造函数参数解析:

注释：上面这个例子里，将依赖类 SpellChecker.java 注入到 TextEditor.java 文件。

如此，便称为依赖注入。

如果存在不止一个参数时，当把参数传递给构造函数时，可能会存在歧义。要解决这个问题，那么构造函数的参数在 bean 定义中的顺序就是把这些参数提供给适当的构造函数的顺序就可以了。

考虑下面的类:

```
package x.y;
public class Foo {
   public Foo(Bar bar, Baz baz) {
      // ...
   }
}
```

下述配置文件工作顺利：

```
<beans>
   <bean id="foo" class="x.y.Foo">
      <constructor-arg ref="bar"/>
      <constructor-arg ref="baz"/>
   </bean>

   <bean id="bar" class="x.y.Bar"/>
   <bean id="baz" class="x.y.Baz"/>
```

```
</beans>
```

让我们再检查一下我们传递给构造函数不同类型的位置。考虑下面的类：

```
package x.y;
public class Foo {
   public Foo(int year, String name) {
      // ...
   }
}
```

如果你使用 `type` 属性显式的指定了构造函数参数的类型，容器也可以使用与简单类型匹配的类型。例如：

```
<beans>

   <bean id="exampleBean" class="examples.ExampleBean">
      <constructor-arg type="int" value="2001"/>
      <constructor-arg type="java.lang.String" value="Zara"/>
   </bean>

</beans>
```

最后并且也是最好的传递构造函数参数的方式，使用 `index` 属性来显式的指定构造函数参数的索引。下面是基于索引为 0 的例子，如下所示：

```
<beans>

   <bean id="exampleBean" class="examples.ExampleBean">
      <constructor-arg index="0" value="2001"/>
      <constructor-arg index="1" value="Zara"/>
   </bean>

</beans>
```

最后，如果你想要向一个对象传递一个引用，你需要使用 标签的 **ref** 属性，如果你想要直接传递值，那么你应该使用如上所示的 **value** 属性。