

SPRING SETTER-BASED DEPENDENCY INJECTION

http://www.tutorialspoint.com/spring/setter_based_dependency_injection.htm

Copyright © tutorialspoint.com

Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

Example:

The following example shows a class *TextEditor* that can only be dependency-injected using pure setter-based injection.

Let us have working Eclipse IDE in place and follow the following steps to create a Spring application:

Step	Description
1	Create a project with a name <i>SpringExample</i> and create a package <i>com.tutorialspoint</i> under the src folder in the created project.
2	Add required Spring libraries using <i>Add External JARs</i> option as explained in the <i>Spring Hello World Example</i> chapter.
3	Create Java classes <i>TextEditor</i> , <i>SpellChecker</i> and <i>MainApp</i> under the <i>com.tutorialspoint</i> package.
4	Create Beans configuration file <i>Beans.xml</i> under the src folder.
5	The final step is to create the content of all the Java files and Bean Configuration file and run the application as explained below.

Here is the content of **TextEditor.java** file:

```
package com.tutorialspoint;

public class TextEditor {
    private SpellChecker spellChecker;

    // a setter method to inject the dependency.
    public void setSpellChecker(SpellChecker spellChecker) {
        System.out.println("Inside setSpellChecker." );
        this.spellChecker = spellChecker;
    }

    // a getter method to return spellChecker
    public SpellChecker getSpellChecker() {
        return spellChecker;
    }

    public void spellCheck() {
        spellChecker.checkSpelling();
    }
}
```

Here you need to check naming convention of the setter methods. To set a variable **spellChecker** we are using **setSpellChecker** method which is very similar to Java POJO classes. Let us create the content of another dependent class file **SpellChecker.java**:

```
package com.tutorialspoint;

public class SpellChecker {
    public SpellChecker(){
        System.out.println("Inside SpellChecker constructor." );
    }
}
```

```

    }

    public void checkSpelling() {
        System.out.println("Inside checkSpelling." );
    }
}

```

Following is the content of the **MainApp.java** file:

```

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();
    }
}

```

Following is the configuration file **Beans.xml** which has configuration for the setter-based injection:

```

<?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 >
        <property name="spellChecker" ref="spellChecker"/>
    </bean>

    <!-- Definition for spellChecker bean -->
    <bean >
    </bean>

</beans>

```

You should note the difference in Beans.xml file defined in constructor-based injection and setter-based injection. The only difference is inside the `<bean>` element where we have used `<constructor-arg>` tags for constructor-based injection and `<property>` tags for setter-based injection.

Second important point to note is that in case you are passing a reference to an object, you need to use **ref** attribute of `<property>` tag and if you are passing a value directly then you should use **value** attribute.

Once you are done with creating source and bean configuration files, let us run the application. If everything is fine with your application, this will print the following message:

```

Inside SpellChecker constructor.
Inside setSpellChecker.
Inside checkSpelling.

```

XML Configuration using p-namespace:

If you have many setter methods then it is convenient to use **p-namespace** in the XML

configuration file. Let us check the difference:

Let us take the example of a standard XML configuration file with <property> tags:

```
<?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">

  <bean >
    <property name="name" value="John Doe"/>
    <property name="spouse" ref="jane"/>
  </bean>

  <bean name="jane" >
    <property name="name" value="John Doe"/>
  </bean>

</beans>
```

Above XML configuration can be re-written in a cleaner way using **p-namespace** as follows:

```
<?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:p="http://www.springframework.org/schema/p"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

  <bean
    p:name="John Doe"
    p:spouse-ref="jane"/>
</bean>

  <bean name="jane"
    p:name="John Doe"/>
</bean>

</beans>
```

Here you should not the difference in specifying primitive values and object references with p-namespace. The **-ref** part indicates that this is not a straight value but rather a reference to another bean

Loading [MathJax]/jax/output/HTML-CSS/jax.js