SPRING - INJECTING COLLECTION

You have seen how to configure primitive data type using **value** attribute and object references using **ref** attribute of the <property> tag in your Bean configuration file. Both the cases deal with passing singular value to a bean.

Now what about if you want to pass plural values like Java Collection types List, Set, Map, and Properties. To handle the situation, Spring offers four types of collection configuration elements which are as follows:

Element	Description
<list>	This helps in wiring ie injecting a list of values, allowing duplicates.
<set>	This helps in wiring a set of values but without any duplicates.
<map>	This can be used to inject a collection of name-value pairs where name and value can be of any type.
<props>	This can be used to inject a collection of name-value pairs where the name and value are both Strings.

You can use either <list> or <set> to wire any implementation of java.util.Collection or an **array**.

You will come across two situations *a* Passing direct values of the collection and *b* Passing a reference of a bean as one of the collection elements.

Example:

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 *SpringExample* and create a package *com.tutorialspoint* under the **src** folder in the created project.
2	Add required Spring libraries using *Add External JARs* option as explained in the *Spring Hello World Example* chapter.
3	Create Java classes *JavaCollection*, and *MainApp* under the *com.tutorialspoint* package.
4	Create Beans configuration file *Beans.xml* 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 **JavaCollection.java** file:

```java
package com.tutorialspoint;
import java.util.*;

public class JavaCollection {
   List addressList;
   Set  addressSet;
   Map  addressMap;
   Properties addressProp;

   // a setter method to set List
   public void setAddressList(List addressList) {
      this.addressList = addressList;
   }

   // prints and returns all the elements of the list.
   public List getAddressList() {
      System.out.println("List Elements :"  + addressList);
      return addressList;
   }

   // a setter method to set Set
   public void setAddressSet(Set addressSet) {
      this.addressSet = addressSet;
   }

   // prints and returns all the elements of the Set.
   public Set getAddressSet() {
      System.out.println("Set Elements :"  + addressSet);
      return addressSet;
   }

   // a setter method to set Map
   public void setAddressMap(Map addressMap) {
      this.addressMap = addressMap;
   }

   // prints and returns all the elements of the Map.
   public Map getAddressMap() {
      System.out.println("Map Elements :"  + addressMap);
      return addressMap;
   }

   // a setter method to set Property
   public void setAddressProp(Properties addressProp) {
      this.addressProp = addressProp;
   }

   // prints and returns all the elements of the Property.
   public Properties getAddressProp() {
      System.out.println("Property Elements :"  + addressProp);
      return addressProp;
   }
}
```

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");

      JavaCollection jc=(JavaCollection)context.getBean("javaCollection");

      jc.getAddressList();
      jc.getAddressSet();
      jc.getAddressMap();
      jc.getAddressProp();
   }
}
```

Following is the configuration file **Beans.xml** which has configuration for all the type of collections:

```
<?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 javaCollection -->
   <bean  >

      <!-- results in a setAddressList(java.util.List) call -->
      <property name="addressList">
         <list>
            <value>INDIA</value>
            <value>Pakistan</value>
            <value>USA</value>
            <value>USA</value>
         </list>
      </property>

      <!-- results in a setAddressSet(java.util.Set) call -->
      <property name="addressSet">
         <set>
            <value>INDIA</value>
            <value>Pakistan</value>
            <value>USA</value>
            <value>USA</value>
        </set>
      </property>

      <!-- results in a setAddressMap(java.util.Map) call -->
      <property name="addressMap">
         <map>
            <entry key="1" value="INDIA"/>
            <entry key="2" value="Pakistan"/>
            <entry key="3" value="USA"/>
            <entry key="4" value="USA"/>
         </map>
      </property>

      <!-- results in a setAddressProp(java.util.Properties) call -->
      <property name="addressProp">
         <props>
            <prop key="one">INDIA</prop>
            <prop key="two">Pakistan</prop>
            <prop key="three">USA</prop>
            <prop key="four">USA</prop>
         </props>
      </property>

   </bean>

</beans>
```

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:

```
List Elements :[INDIA, Pakistan, USA, USA]
Set Elements :[INDIA, Pakistan, USA]
ap Elements :{1=INDIA, 2=Pakistan, 3=USA, 4=USA}
Property Elements :{two=Pakistan, one=INDIA, three=USA, four=USA}
```

Injecting Bean References:

Following Bean definition will help you understand how to inject bean references as one of the collection's element. Even you can mix references and values all together as shown below:

```
<?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 Definition to handle references and values -->
   <bean  >

      <!-- Passing bean reference  for java.util.List -->
      <property name="addressList">
         <list>
            <ref bean="address1"/>
            <ref bean="address2"/>
            <value>Pakistan</value>
         </list>
      </property>

      <!-- Passing bean reference  for java.util.Set -->
      <property name="addressSet">
         <set>
            <ref bean="address1"/>
            <ref bean="address2"/>
            <value>Pakistan</value>
         </set>
      </property>

      <!-- Passing bean reference  for java.util.Map -->
      <property name="addressMap">
         <map>
            <entry key="one" value="INDIA"/>
            <entry key ="two" value-ref="address1"/>
            <entry key ="three" value-ref="address2"/>
         </map>
      </property>

   </bean>

</beans>
```

To use above bean definition, you need to define your setter methods in such a way that they should be able to handle references as well.

Injecting null and empty string values

If you need to pass an empty string as a value then you can pass it as follows:

```
<bean  >
   <property name="email" value=""/>
</bean>
```

The preceding example is equivalent to the Java code: exampleBean.setEmail ""

If you need to pass an NULL value then you can pass it as follows:

```
<bean  >
   <property name="email"><null/></property>
</bean>
```

The preceding example is equivalent to the Java code: exampleBean.setEmail*null*