Guide to JAXB Annotations

Lokesh Gupta February 7, 2023

JAXB

Learn about **JAXB annotations** in detail along with their usage during marshalling and unmarshalling operations on Java beans.

1. JAXB Annotations

Annotation	Scope	Description
`@XmlRootElement`	Class, Enum	Defines the XML root element. Root Java classes must be registered with th when created.
`@XmlAccessorType`	Package, Class	Defines the fields and properties of your Java classes that the JAXB engine has four values: `PUBLIC_MEMBER`, `FIELD`, `PROPERTY` and `NONE`.
`@XmlAccessorOrder`	Package, Class	Defines the sequential order of the children.
`@XmlType`	Class, Enum	Maps a Java class to a schema type. It defines the type name and order of i
`@XmlElement`	Field	Maps a field or property to an XML element
`@XmlAttribute`	Field	Maps a field or property to an XML attribute
`@XmlTransient`	Field	Prevents mapping a field or property to the XML Schema
`@XmlValue`	Field	Maps a field or property to the text value on an XML tag.
`@XmlList`	Field, Parameter	Maps a collection to a list of values separated by space.
`@XmlElementWrapper`	Field	Maps a Java collection to an XML-wrapped collection

1.1. *@XmlRootElement*

This **maps a class or an enum type to an XML root element**. When a top-level class or an enum type is annotated with the `@XmlRootElement` annotation, then its value is represented as XML element in an XML document.

```
@XmlRootElement(name = "employee")
@XmlAccessorType(XmlAccessType.PROPERTY)
public class Employee implements Serializable
{
    //More code
}
```

The above will result into:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employee>
  //....
</employee>
```

1.2. *@XmlAccessorType*

It **defines the class fields that the JAXB engine uses for including into generated XML**. It has four possible values.

- *`FIELD`* – Every non-static, non-transient field in a JAXB-bound class will be automatically bound to XML, unless annotated by `XmlTransient`.
- *`NONE`* – None of the fields or properties is bound to XML unless they are specifically annotated with some of the JAXB annotations.
- *`PROPERTY`* – Every getter/setter pair in a JAXB-bound class will be automatically bound to XML, unless annotated by `XmlTransient`.
- *`PUBLIC_MEMBER`* – Every public getter/setter pair and every public field will be automatically bound to XML, unless annotated by `XmlTransient`.
- **Default value is** `PUBLIC_MEMBER`.

```
@XmlRootElement(name = "employee")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee implements Serializable
{
    private Integer id;
    private String firstName;
    private String lastName;
}
```

The above will result into:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employee>
  <firstName>Lokesh</firstName>
    <id>1</id>
    <lastName>Gupta</lastName>
</employee>
```

1.3. *@XmlAccessorOrder*

It **controls the ordering of fields and properties in the generated XML**. We can have predefined values ALPHABETICAL or UNDEFINED.

```
@XmlRootElement(name = "employee")
@XmlAccessorOrder(AccessorOrder.ALPHABETICAL)
public class Employee implements Serializable
{
  private Integer id;
  private String firstName;
  private String lastName;
  private Department department;
}
```

The above will result into:

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
   <department>
      <id>101</id>
      <name>IT</name>
   </department>
   <firstName>Lokesh</firstName>
   <id>1</id>
   <lastName>Gupta</lastName>
</employee>
```

1.4. *@XmlType*

It **maps a Java class or enum type to a schema type**. It defines the type name, namespace and order of its children. It is used to match the element in the schema to the element in the model.

```
@XmlRootElement(name = "employee")
@XmlType(propOrder={"id", "firstName" , "lastName", "department" })
public class Employee implements Serializable
{
  private Integer id;
  private String firstName;
  private String lastName;
  private Department department;
}
```

1.5. *@XmlElement*

It maps a JavaBean property to an XML element derived from the property name.

```
@XmlRootElement(name = "employee")
public class Employee implements Serializable
{
  @XmlElement(name=employeeId)
  private Integer id;

  @XmlElement
  private String firstName;

  private String lastName;
  private Department department;
}
```

The above will result into:

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
  <employeeId>1</employeeId>
    <firstName>Lokesh</firstName>
</employee>
```

1.6. *@XmlAttribute*

It maps a JavaBean property to an XML attribute.

```
@XmlRootElement(name = "employee")
public class Employee implements Serializable
```

```
{
  @XmlAttribute
  private Integer id;

  private String firstName;
  private String lastName;
  private Department department;
}
```

The above will result into:

```
<?xml version="1.0" encoding="UTF-8"?>
<employee id="1">
   <department>
      <id>101</id>
      <name>IT</name>
   </department>
   <firstName>Lokesh</firstName>
   <lastName>Gupta</lastName>
</employee>
```

1.7. *@XmlTransient*

It prevents the mapping of a JavaBean property/type to XML representation. When placed on a class, it indicates that it shouldn't be mapped to XML. Properties on such class will be mapped to XML along with its derived classes as if the class is inlined.

`@XmlTransient` is mutually exclusive with all other JAXB-defined annotations.

```
@XmlRootElement(name = "employee")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee implements Serializable
{
  @XmlTransient
  private Integer id;

  private String firstName;
  private String lastName;
  private Department department;
}
```

The above will result into:

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
   <firstName>Lokesh</firstName>
   <lastName>Gupta</lastName>
   <department>
      <id>101</id>
      <name>IT</name>
   </department>
</employee>
```

1.8. *@XmlValue*

It enables mapping a class to an XML Schema complex type with simple content or an XML Schema simple type. It's more related to schema mapping to model mapping.

1.9. *@XmlList*

It is used to **map a property to a *List* simple type**. It allows multiple values to be represented as **whitespace-separated tokens** in a single element.

```
@XmlRootElement(name = "employee")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee implements Serializable
{
  private List<String> hobbies;
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
  <hobbies>Swimming</hobbies>
    <hobbies>Playing</hobbies>
    <hobbies>Karate</hobbies>
</employee>
```

After using `@XmlList`, observe the output.

```
@XmlRootElement(name = "employee")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee implements Serializable
{
```

```
  @XmlList
  private List<String> hobbies;
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
  <hobbies>Swimming Playing Karate</hobbies>
</employee>
```

1.10. *@XmlElementWrapper*

Generates a wrapper element around XML representation. This is primarily intended to be used to produce a wrapper XML element around collections. So, it must be used with collection property.

```
@XmlRootElement(name = "employee")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee implements Serializable
{
  @XmlElementWrapper(name="hobbies")
  @XmlElement(name="hobby")
  private List<String> hobbies;
}
```

The above will result into:

```
<?xml version="1.0" encoding="UTF-8"?>
<employee>
  <hobbies>
        <hobby>Swimming</hobby>
        <hobby>Playing</hobby>
        <hobby>Karate</hobby>
    </hobbies>
</employee>
```

2. JAXB Annotation Example

Learn to apply JAXB annotations on model classes and then marshal the object into the XML file.

```
import java.io.Serializable;
import java.util.List;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElementWrapper;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "employee")
@XmlAccessorType(XmlAccessType.FIELD)
public class Employee implements Serializable {

  private Integer id;
  private String firstName;
  private String lastName;

  private Department department;

  @XmlElementWrapper(name="hobbies")
  @XmlElement(name="hobby")
  private List<String> hobbies;


  //Constructors, Setters and Getters
}
```

The following code marshals an instance of the *Employee* class.

```
import java.io.File;
import java.util.Arrays;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
```

```
import javax.xml.bind.Marshaller;
import com.howtodoinjava.demo.model.Department;
import com.howtodoinjava.demo.model.Employee;

public class JaxbExample
{
  public static void main(String[] args)
  {
    Employee employee = new Employee(1, "Lokesh", "Gupta", new Department(101, "IT"));

    employee.setHobbies(Arrays.asList("Swimming","Playing", "Karate"));

    jaxbObjectToXML(employee);
  }

  private static void jaxbObjectToXML(Employee employee)
  {
      try {
          JAXBContext jaxbContext = JAXBContext.newInstance(Employee.class);
          Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

          jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE); //

          //Print XML String to Console
          jaxbMarshaller.marshal(employee, new File("employee.xml"));

      } catch (JAXBException e) {
          e.printStackTrace();
      }
  }
}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<employee>
    <id>1</id>
    <firstName>Lokesh</firstName>
    <lastName>Gupta</lastName>
    <department>
        <id>101</id>
        <name>IT</name>
    </department>
    <hobbies>
        <hobby>Swimming</hobby>
        <hobby>Playing</hobby>
        <hobby>Karate</hobby>
    </hobbies>
</employee>
```

D