XML JAXB

Anotaciones XML JAXB

@XmlRootElement: Defines the XML root element. Root Java classes must be registered with the JAXB context when created.

@XmlAccessorType: Defines the fields and properties of your Java classes that the JAXB engine uses for binding. It has four values: PUBLIC_MEMBER, FIELD, PROPERTY and NONE.

@XmlAccessorOrder: Defines the sequential order of the children.

@XmlType: Defines the type name and order of its children.

@XmlElement: Maps a field or property to an XML elementx.

@XmlAttribute: Maps a field or property to an XML attribute.

@XmlTransient: Prevents mapping a field or property to the XML Schema

@XmlValue: Maps a field or property to the text value on an XML tag.

@XmlList: Maps a collection to a list of values separated by space.

@XmlElementWrapper: Maps a Java collection to an XML-wrapped collection.

@XmlEnum: Maps an enum type Enum to XML representation. This annotation, together with **XmlEnumValue** provides a mapping of enum type to XML representation.

@ XmlEnumValue: Maps an enum constant in Enum type to XML representation.

@XmlJavaTypeAdapter: Use an adapter that implements XmlAdapter for custom marshaling and unmarsharling.

Adaptadores XML

Extender de la clase abstracta:

XmlAdapter <String, ClaseAAdaptar>

Leer y Escribir ficheros XML

Escribir Fichero XML:

Usar objetos de las clases JAXBContext y Marshaller para crear el contexto y el parseador o marshaller.

Crear contexto:

JAXBContext context = JAXBContext.newInstance(raiz.getClass());

Donde raiz representa el objeto a escribir en el fichero XML.

Crear marshaller, objeto que se encarga de escribir el XML:

Marshaller ms = context.createMarshaller();

Poner propiedades al marshaller:

ms.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);

Escribir fichero XML:

ms.marshal(raiz, writer);

Donde writer representa un objeto de la clase Writer que apunta al fichero a guardar.

Leer fichero XML:

Usar objetos de las clases JAXBContext y UnMarshaller para crear el contexto y el serializador o unmarshaller.

Crear contexto:

JAXBContext context = JAXBContext.newInstance(claseRaiz);

Donde claseRaiz representa el objeto de la clase Class que representa al objeto a leer del fichero XML.

Crear UnMarshaller:

Unmarshaller ums = context.createUnmarshaller();

Leer el fichero XML y guardarlo en un objeto de la clase ClaseRaiz:

ClaseRaiz raiz = (ClaseRaiz) ums.unmarshal(file);

Donde *file* representa un objeto de la clase *File* que apunta al fichero a leer.

JSON-GSON

Anotaciones JSON GSON

@SerializedName: Indicates this member should be serialized to JSON with the provided name value as its field name.

@JsonAdapter : An annotation that indicates the Gson TypeAdapter to use with a class or field.

Adaptadores JSON

Extender la clase abstracta:

TypeAdapter <ClaseAAdaptar>

Leer y Escribir ficheros JSON

Escribir ficheros JSON:

Crear BuilderGSON:

GsonBuilder creadorGson = new GsonBuilder();

Poner propiedades:

creadorGson.setPrettyPrinting();

Registrar los adaptadores:

creadorGson.registerTypeAdapter (ClaseAAdaptar.class, objetoAdaptador);

Donde ClaseAAdaptar representa la clase que se adaptar y objetoAdaptador representa un objeto de la clase adaptadora.

Crear objeto Gson:

Gson gson = creadorGson.create();

Escribir objeto o colección de objetos en fichero JSON:

gson.toJson (src, fichero);

Donde src representa el objeto o colección de objetos a guardar y fichero representa un objeto de la clase FileWriter que referencia al fichero a guardar.

Leer fichero JSON

Crear BuilderGSON:

GsonBuilder creadorGson = new GsonBuilder();

Registrar los adaptadores:

creadorGson.registerTypeAdapter (ClaseAAdaptar.class, objetoAdaptador);

Donde ClaseAAdaptar representa la clase que se adaptar y objetoAdaptador representa un objeto de la clase adaptadora.

Crear objeto Gson:

Gson gson = creadorGson.create();

Leer fichero json que representa un objeto:

ClaseElemento elemento = (ClaseElemento) gson.fromJson(reader,ClaseElemento.class)

Donde ClaseElemento representa la clase que recoge la información guardada en el fichero y reader un objeto de la clase FileReader que apunta al fichero a leer.

Leer fichero Json que representa una colección de objetos:

Type listaDeObjetos;
List<ClaseElemento> objetos;
listaDeObjetos = TypeToken.getParameterized(List.class, ClaseElemento.class).getType();
objetos = gson.fromJson(reader, listaDeObjetos);

Donde ClaseElemento representa la clase que recoge la información guardada en el fichero y reader un objeto de la clase FileReader que apunta al fichero a leer.

CSV OPENCSV

Anotaciones CSV

@CsvBindByPosition*(position = n)* : Specifies a binding between a column number of the CSV input and a field in a bean (n starts with 0)

@CsvDate(formato): This annotation indicates that the destination field is an expression of time. Formato indica el formato de la fecha ("yyyy-MM-dd").

@CsvBindAndSplitByPosition*(position = n,*

```
                         elementType= Elemento.class, //Tipo Elem. Colecc
                         splitOn = ";", // separador de elementos
                         converter = ClaseConvertidora.class // Clase convertidora
                         extends AbstractCsvConverter
                         ,writeDelimiter = ";")
```

this annotation interprets one field of the input as a collection that will be split up into its components and assigned to a collection-based bean field.

@CsvCustomBindByPosition*(position=n,*

```
                         converter = ClaseAdaptadora.class) // extends
                         AbstractBeanField<String, ElementoAAdaptar>
```

Allows us to specify a class that will perform the translation from source to destination. For special needs, we can implement a class that takes the source field from the CSV and translates it into a form of our choice.

Leer fichero csv

Leer elementos de un fichero csv:

```
List<Elemento>beans = new CsvToBeanBuilder(new fileReader()))
                          .withType(Elemento.class)
                          .withSeparator(‘’)
                          .build()
                          .parse();// stream()
```

Leer elementos de un String con el csv (Adaptadores):

```
StringReader stringReader = new StringReader(value);
CSVParser icsvParser = new CSVParserBuilder().withSeparator('').build();
CSVReader csvReader = new
CSVReaderBuilder(stringReader).withCSVParser(icsvParser).build();
Stream <Elemento> beans = new CsvToBeanBuilder(csvReader)

                                  .withType(Elemento.class)

                                  .build()

                                  .stream();

return beans.findFirst().orElseGet(Elemento::new);
```

Escribir fichero csv

```
StatefulBeanToCsv beanToCsv = new StatefulBeanToCsvBuilder(writer).

                                  withSeparator(':').
                                  withApplyQuotesToAll(false).
                                  build();
                                  beanToCsv.write(arrayList);
```

Donde writer indica un objeto de la clase Writer que representa el lugar donde escribir el csv y arrayList indica un objeto que representa la lista elementos a escribir en el fichero csv**.** **No olvidar cerrar writer**.

LOMBOK

Anotaciones Lombok

@Getter - @Setter : generate the default getter/setter automatically.

@NoArgsConstructor: generate a constructor with no parameters

@AllArgsConstructor: generates a constructor with 1 parameter for each field in your class

@RequiredArgsConstructor: generates a constructor with 1 parameter for each field that requires special handling. All non-initialized final fields get a parameter, as well as any fields that are marked as @NonNull that aren't initialized where they are declared. For those fields marked with @NonNull, an explicit null check is also generated. The constructor will throw a NullPointerException if any of the parameters intended for the fields marked with @NonNull contain null. The order of the parameters match the order in which the fields appear in your class.

@ToString: generate an implementation of the toString() method.

@Data: is a convenient shortcut annotation that bundles the features of @ToString, @EqualsAndHashCode, @Getter / @Setter and @RequiredArgsConstructor

@Builder: produces complex builder APIs for your classes.

@Singular: treat that builder node as a collection, and it generates 2 'adder' methods instead of a 'setter' method. One which adds a single element to the collection, and one which adds all elements of another collection to the collection. No setter to just set the collection (replacing whatever was already added) will be generated.

@EqualsAndHashCode: generate implementations of the equals(Object other) and hashCode() methods.

@NonNull: generate a null-check statement