Date with Java, SQL and XML

Rodrigo García Carmona (v1.2.1)

Since there are several ways in which one can store time and date information using Java, it's important to use the most appropriate one in order to be able to store and retrieve this information from the database. We'll use several classes:

- To work with dates in Java we'll use the *LocalDate* and *LocalDateTime* classes.
- To persist to and read from the database we'll use the *java.sql.Date* class. It's not the same as the *java.util.Date* class!
- To marshall and unmarshall to a XML file we'll turn the *LocalDate* and *LocalDateTime* classes into *Strings*.

Create a date

With time:

```
LocalDateTime january1st2014WithTime = LocalDateTime.of(2014, Month.JANUARY, 1, 12, 30);
```

Without time:

```
LocalDate january1st2014 = LocalDate.of(2014, Month.JANUARY, 1);
```

Print a date to the screen

It can't be easier:

```
january1st2014.toString();
```

Create a date from a String

With time:

```
String withTime = "2014-01-01 12:30";
DateTimeFormatter formatterWithTime = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
LocalDateTime january1st2014WithTime = LocalDateTime.parse(withTime, formatterWithTime);
```

Without time:

```
String withoutTime = "2014-01-01";
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");
LocalDate january1st2014 = LocalDate.parse(withoutTime, formatter);
```

Note that we can (and should) reuse the *DateTimeFormatter* in several parts of our code.

Transform from and to java.sql.Date

From *java.sql.Date* to *java.time.LocalDate* (or to *java.time.LocalDateTime*):

```
date.toLocalDate();
date.toLocalDateTime();
```

From *java.time.LocalDate* (or from *java.time.LocalDateTime*) to *java.sql.Date*:

```
Date.valueOf(localDate);
```

Show correctly the Date stored in SQLite

Using java.sql.Date we store the date in the database as the number of miliseconds since 1970.
If we want to see it correctly using SQL without Java we will need to use the Date and Time Functions of SQLite

In this case it will be:

```
date(dateNumber/1000, 'unixepoch','localtime')
```

(Un)marshall a java.sql.Date from/to XML

To properly (un)marshall a date we must define what serializing strategy will be used. We do that by creating a class that extends *XmlAdapter*. Such class must implement a *marshall* method that turns a *java.sql.Date* into a *String*, and an *unmarshall* method that turns a *String* into a *java.sql.Date*.

The following class is one possible implementation of this:

```
public class SQLDateAdapter extends XmlAdapter<String, Date> {

    private DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd");

    @Override
    public String marshal(Date sqlDate) throws Exception {
        return sqlDate.toLocalDate().format(formatter);
    }

    @Override
    public Date unmarshal(String string) throws Exception {
        LocalDate localDate = LocalDate.parse(string, formatter);
        return Date.valueOf(localDate);
    }

}
```

Here we've used *LocalDate* to get the desired *String* representation of the date.

On top of that, we must also annotate the java.sql.Date attributes to indicate that they will be using the adapter class we just created:

```
@XmlElement
@XmlJavaTypeAdapter(SQLDateAdapter.class)
private Date date;
```