(http://baeldung.com)

Fecha de Jackson

Última modificación: 19 de abril de 2018

por baeldung (http://www.baeldung.com/author/baeldung/)
(http://www.baeldung.com/author/baeldung/)

Jackson (http://www.baeldung.com/category/jackson/)

1. Información general

En este tutorial, vamos a serializar las fechas con Jackson. Comenzaremos serializando un simple java.util. *Fecha* , luego Joda-Time y Java 8 *DateTime* .

2. *Fecha de* serialización a Timestamp

Primero, veamos cómo serializar un simple ***java.util.Date* con Jackson** .

En el siguiente ejemplo, serializaremos una instancia de " *Evento* " que tiene un campo de *fecha* " *eventDate* ":

```
@Test
public void whenSerializingDateWithJackson_thenSerializedToTimestamp(
  throws JsonProcessingException, ParseException {

    SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy hh:mm");
    df.setTimeZone(TimeZone.getTimeZone("UTC"));

    Date date = df.parse("01-01-1970 01:00");
    Event event = new Event("party", date);

    ObjectMapper mapper = new ObjectMapper();
    mapper.writeValueAsString(event);
}
```

Lo que es importante aquí es que Jackson serializará la fecha en un formato de marca de tiempo por defecto (cantidad de milisegundos desde el 1 de enero de 1970, UTC).

El resultado real de la serialización de " *evento* " es:

```
{
   "name":"party",
   "eventDate":3600000
}
```

3. *Fecha de* serialización a ISO-8601

Serializar a este formato de sello de tiempo no es óptimo. Vamos a serializar ahora la *fecha* en el formato *ISO-8601* :

```
@Test
public void whenSerializingDateToISO8601_thenSerializedToText()
  throws JsonProcessingException, ParseException {

    SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy hh:mm");
    df.setTimeZone(TimeZone.getTimeZone("UTC"));

    String toParse = "01-01-1970 02:30";
    Date date = df.parse(toParse);
    Event event = new Event("party", date);

    ObjectMapper mapper = new ObjectMapper();
    mapper.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);
    mapper.setDateFormat(new ISO8601DateFormat());
    String result = mapper.writeValueAsString(event);
    assertThat(result, containsString("1970-01-01T02:30:00Z"));
}
```

Observe cómo la representación de la fecha ahora es mucho más legible.

4. Configure *ObjectMapper DateFormat*

Las soluciones anteriores aún carecen de la flexibilidad total de elegir el formato exacto para representar las instancias de *java.util.Date* .

Veamos ahora una configuración que nos permitirá **configurar nuestros formatos para representar fechas** :

```
@Test
public void whenSettingObjectMapperDateFormat_thenCorrect()
  throws JsonProcessingException, ParseException {

    SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy hh:mm");

    String toParse = "20-12-2014 02:30";
    Date date = df.parse(toParse);
    Event event = new Event("party", date);

    ObjectMapper mapper = new ObjectMapper();
    mapper.setDateFormat(df);

    String result = mapper.writeValueAsString(event);
    assertThat(result, containsString(toParse));
}
```

Tenga en cuenta que, aunque ahora somos más flexibles con respecto al formato de fecha, todavía estamos usando una configuración global en el nivel de todo el *ObjectMapper* .

5. Utilice *@JsonFormat* para formatear la *fecha*

A continuación, echemos un vistazo a la anotación *@JsonFormat* para **controlar el formato de fecha en clases individuales en** lugar de globalmente, para toda la aplicación:

```
public class Event {
    public String name;

    @JsonFormat
      (shape = JsonFormat.Shape.STRING, pattern = "dd-MM-yyyy hh:mm:ss
    public Date eventDate;
}
```

Ahora - vamos a probarlo:

```
@Test
public void whenUsingJsonFormatAnnotationToFormatDate_thenCorrect()
  throws JsonProcessingException, ParseException {

    SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy hh:mm:ss");
    df.setTimeZone(TimeZone.getTimeZone("UTC"));

    String toParse = "20-12-2014 02:30:00";
    Date date = df.parse(toParse);
    Event event = new Event("party", date);

    ObjectMapper mapper = new ObjectMapper();
    String result = mapper.writeValueAsString(event);
    assertThat(result, containsString(toParse));
}
```


6. Serializador de *fecha* personalizado

A continuación, para obtener un control total sobre el resultado, aprovecharemos un serializador personalizado para Fechas:

```
public class CustomDateSerializer extends StdSerializer<Date> {

    private SimpleDateFormat formatter
      = new SimpleDateFormat("dd-MM-yyyy hh:mm:ss");

    public CustomDateSerializer() {
        this(null);
    }

    public CustomDateSerializer(Class t) {
        super(t);
    }

    @Override
    public void serialize (Date value, JsonGenerator gen, SerializerF
      throws IOException, JsonProcessingException {
        gen.writeString(formatter.format(value));
    }
}
```

A continuación, *utilicémoslo* como el serializador de nuestro campo " *eventDate* ":

```
public class Event {
    public String name;

    @JsonSerialize(using = CustomDateSerializer.class)
    public Date eventDate;
}
```

Finalmente, probemos:

```
@Test
public void whenUsingCustomDateSerializer_thenCorrect()
  throws JsonProcessingException, ParseException {

    SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy hh:mm:ss");

    String toParse = "20-12-2014 02:30:00";
    Date date = df.parse(toParse);
    Event event = new Event("party", date);

    ObjectMapper mapper = new ObjectMapper();
    String result = mapper.writeValueAsString(event);
    assertThat(result, containsString(toParse));
}
```

Otras lecturas:

Cómo serializar los enums como objetos JSON con Jackson (http://www.baeldung.com/jackson-serialize-enums)

Cómo serializar un Enum como un objeto JSON usando Jackson 2.

Leer más (http://www.baeldung.com/jackson-serialize-enums) →

Jackson - Serializador personalizado (http://www.baeldung.com/jackson-custom-serialization)

Controle su salida JSON con Jackson 2 mediante el uso de un serializador personalizado.

Leer más (http://www.baeldung.com/jackson-custom-serialization) →

Comenzar con la deserialización personalizada en Jackson (http://www.baeldung.com/jackson-deserialization)

Use Jackson para mapear JSON personalizado a cualquier gráfico de entidad java con control total sobre el proceso de deserialización.

Leer más (http://www.baeldung.com/jackson-deserialization) →

7. Serializar Joda-Time con Jackson

Dates aren't always an instance of *java.util.Date*; actually – they're more and more represented by some other class – and a common one is, of course, the *DateTime* implementation from the Joda-Time library.

Let's see how we can **serialize *DateTime* with Jackson**.

We'll make use of the ***jackson-datatype-joda*** module for out of the box Joda-Time support:

```
<dependency>
  <groupId>com.fasterxml.jackson.datatype</groupId>
  <artifactId>jackson-datatype-joda</artifactId>
  <version>2.4.0</version>
</dependency>
```

And now we can simply register the *JodaModule* and be done:

```
@Test
public void whenSerializingJodaTime_thenCorrect()
  throws JsonProcessingException {
    DateTime date = new DateTime(2014, 12, 20, 2, 30,
      DateTimeZone.forID("Europe/London"));

    ObjectMapper mapper = new ObjectMapper();
    mapper.registerModule(new JodaModule());
    mapper.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);

    String result = mapper.writeValueAsString(date);
    assertThat(result, containsString("2014-12-20T02:30:00.000Z"));
}
```

8. Serialize Joda *DateTime* with Custom Serializer

If we don't want the extra Joda-Time Jackson dependency – we can also make use of **a custom serializer** (similar to the earlier examples) to get *DateTime* instances serialized cleanly:

```
public class CustomDateTimeSerializer extends StdSerializer<DateTime>

    private static DateTimeFormatter formatter =
      DateTimeFormat.forPattern("yyyy-MM-dd HH:mm");

    public CustomDateTimeSerializer() {
        this(null);
    }

     public CustomDateTimeSerializer(Class<DateTime> t) {
         super(t);
     }

    @Override
    public void serialize
      (DateTime value, JsonGenerator gen, SerializerProvider arg2)
      throws IOException, JsonProcessingException {
        gen.writeString(formatter.print(value));
    }
}
```

Next – let's use it as our property "*eventDate*" serializer:

```
public class Event {
    public String name;

    @JsonSerialize(using = CustomDateTimeSerializer.class)
    public DateTime eventDate;
}
```

Finally – let's put everything together and test it:

```
@Test
public void whenSerializingJodaTimeWithJackson_thenCorrect()
  throws JsonProcessingException {

    DateTime date = new DateTime(2014, 12, 20, 2, 30);
    Event event = new Event("party", date);

    ObjectMapper mapper = new ObjectMapper();
    String result = mapper.writeValueAsString(event);
    assertThat(result, containsString("2014-12-20 02:30"));
}
```


9. Serialize Java 8 *Date* with Jackson

Next – let's see how to serialize Java 8 *DateTime* – in this example, ***LocalDateTime* – using Jackson**. We can make use of the ***jackson-datatype-jsr310*** module:

```
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
    <version>2.4.0</version>
</dependency>
```

Now, all we need to do is register the *JSR310Module* and Jackson will take care of the rest:

```
@Test
public void whenSerializingJava8Date_thenCorrect()
  throws JsonProcessingException {
    LocalDateTime date = LocalDateTime.of(2014, 12, 20, 2, 30);

    ObjectMapper mapper = new ObjectMapper();
    mapper.registerModule(new JSR310Module());
    mapper.disable(SerializationFeature.WRITE_DATES_AS_TIMESTAMPS);

    String result = mapper.writeValueAsString(date);
    assertThat(result, containsString("2014-12-20T02:30"));
}
```

10. Serialize Java 8 *Date* Without any Extra Dependency

If you don't want the extra dependency, you can always use **a custom serializer to write out the Java 8 *DateTime*.** Cuál de

```
public class CustomLocalDateTimeSerializer
  extends StdSerializer<LocalDateTime> {

    private static DateTimeFormatter formatter =
      DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");

    public CustomLocalDateTimeSerializer() {
        this(null);
    }

    public CustomLocalDateTimeSerializer(Class<LocalDateTime> t) {
        super(t);
    }

    @Override
    public void serialize(
      LocalDateTime value,
      JsonGenerator gen,
      SerializerProvider arg2)
      throws IOException, JsonProcessingException {

        gen.writeString(formatter.format(value));
    }
}
```

Next – let's use the serializer for our "*eventDate*" field:

```
public class Event {
    public String name;

    @JsonSerialize(using = CustomLocalDateTimeSerializer.class)
    public LocalDateTime eventDate;
}
```

Now – let's test it:

```
@Test
public void whenSerializingJava8DateWithCustomSerializer_thenCorrect()
  throws JsonProcessingException {

    LocalDateTime date = LocalDateTime.of(2014, 12, 20, 2, 30);
    Event event = new Event("party", date);

    ObjectMapper mapper = new ObjectMapper();
    String result = mapper.writeValueAsString(event);
    assertThat(result, containsString("2014-12-20 02:30"));
}
```

11. Deserialize *Date*

Next – let's see how to deserialize a *Date* with **Jackson**. In the following example – we deserialize an "*Event*" instance containing a date:

```
@Test
public void whenDeserializingDateWithJackson_thenCorrect()
  throws JsonProcessingException, IOException {

    String json = "{"name":"party","eventDate":"20-12-2014 02:30:00"}

    SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy hh:mm:ss");
    ObjectMapper mapper = new ObjectMapper();
    mapper.setDateFormat(df);

    Event event = mapper.readerFor(Event.class).readValue(json);
    assertEquals("20-12-2014 02:30:00", df.format(event.eventDate));
}
```

12. Deserialize Joda *ZonedDateTime* with Time Zone Preserved

In its default configuration, Jackson adjusts the time zone of a Joda *ZonedDateTime* to the time zone of the local context. As, by default, the time zone of the local context is not set and has to be configured manually, Jackson adjusts the time zone to GMT:

```
@Test
public void whenDeserialisingZonedDateTimeWithDefaults_thenNotCorrect
  throws IOException {
    ObjectMapper objectMapper = new ObjectMapper();
    objectMapper.findAndRegisterModules();
    objectMapper.disable(SerializationFeature.WRITE_DATES_AS_TIMESTA
    ZonedDateTime now = ZonedDateTime.now(ZoneId.of("Europe/Berlin"))
    String converted = objectMapper.writeValueAsString(now);

    ZonedDateTime restored = objectMapper.readValue(converted, ZonedD
    System.out.println("serialized: " + now);
    System.out.println("restored: " + restored);
    assertThat(now, is(restored));
}
```

This test case will fail with output:

```
serialized: 2017-08-14T13:52:22.071+02:00[Europe/Berlin]
restored: 2017-08-14T11:52:22.071Z[UTC]
```

Fortunately, there is a quick and simple fix for this odd default-behavior: **we just have to tell Jackson, not to adjust the time zone.**

This can be done by adding the below line of code to the above test case:

```
objectMapper.disable(DeserializationFeature.ADJUST_DATES_TO_CONTEXT_TI
```

Note that, to preserve time zone we also have to disable the default behavior of serializing the date to the timestamp.

13. Custom *Date* Deserializer

Let's also see how to use **a custom *Date* deserializer**; we'll write a custom deserializer for the property "*eventDate*":

```
public class CustomDateDeserializer extends StdDeserializer<Date> {

    private SimpleDateFormat formatter =
      new SimpleDateFormat("dd-MM-yyyy hh:mm:ss");

    public CustomDateDeserializer() {
        this(null);
    }

    public CustomDateDeserializer(Class<?> vc) {
        super(vc);
    }

    @Override
    public Date deserialize(JsonParser jsonparser, DeserializationCon
      throws IOException, JsonProcessingException {
        String date = jsonparser.getText();
        try {
            return formatter.parse(date);
        } catch (ParseException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Next – let's use it as the "*eventDate*" deserializer:

```
public class Event {
    public String name;

    @JsonDeserialize(using = CustomDateDeserializer.class)
    public Date eventDate;
}
```

And finally – let's test it:

```
@Test
public void whenDeserializingDateUsingCustomDeserializer_thenCorrect(
  throws JsonProcessingException, IOException {

    String json = "{"name":"party","eventDate":"20-12-2014 02:30:00"}

    SimpleDateFormat df = new SimpleDateFormat("dd-MM-yyyy hh:mm:ss");
    ObjectMapper mapper = new ObjectMapper();

    Event event = mapper.readerFor(Event.class).readValue(json);
    assertEquals("20-12-2014 02:30:00", df.format(event.eventDate));
}
```

14. Conclusion

In this extensive *Date* article, we looked at most relevant ways **Jackson can help marshalli and unmarshall a date to JSON** using a sensible format we have control over.

The implementation of all these examples and code snippets **can be found in my GitHub project (https://github.com/eugenp/tutorials/tree/master/jackson#readme)** – this is a Maven-based project, so it should be easy to import and run as it is.

Subscribe ▾ ▲ newest ▲ oldest ▲ most voted

Guest

Bill

Eugen –

Again, great information! Something that I have been trying to get a definitive answer on for awhile (and the topic is serialization):

Spring MVC 4, I'm letting spring serialize my response body using @RestController. I'd like to use my own serialization. I see a lot of solutions from various sources for spring 3 but nothing for 4 and no consensus on the "right" solution. Would you be willing to weigh in?

Thanks,

Bill

+ 0 − 3 years ago

Guest

Eugen Paraschiv (http://www.baeldung.com/)

Hey Bill,
Sure – you can definitely use your own `ObjectMapper`, configured exactly how you want it. Check out the "Without Spring Boot" section here (https://spring.io/blog/2014/12/02/latest-jackson-integration-improvements-in-spring) – that will allow you to jump in and customize it fully. Hope it helps. Cheers,
Eugen.

+ 0 − 3 years ago

Guest

Bill

It does – thanks!

+ 0 −

Guest

Kisna

If I were to use the same entity for serialization and de-serialization, isn't there an elegant way to specify just one annotation?

I have also had use cases where customer would pass two different date formats (sometimes suppressing the time part or the timezone part) that needed to be handled gracefully?

What about “strictness” in formats in the above case? Default date formats are not strict at all.

+ 0 −

2 years ago

Guest

Eugen Paraschiv (http://www.baeldung.com/)

Hey Kisna, The answer to your first question depends on which of these solutions you actually use. If you’re thinking of a simple annotation, usually putting that annotation on the field itself will mean that Jackson is going to use it for both serialization and deserialization. Now, handling 2 different formats sounds like it would need the more flexible custom serializer / deserializer – and some manual checking of the format to get the right one. Finally, on the default date format, you definitely don’t have to use any default – you can specify your own and make it adhere… Read more »

+ 0 −

2 years ago

Guest

Kisna

Sorry, the first question was invalid. Rest makes sense, there is no clean option.

+ 0 −

2 years ago

Guest

Kisna

Is there a clean way to deserialize old (<7.x) Java Dates with JODA using JODAModule() without a custom serializer/deserializer?

+ 0 −

2 years ago

Guest

Eugen Paraschiv (http://www.baeldung.com/)

I haven’t tried to use the Joda module with Java 8 dates – and I don’t think it’s supported. That module is specifically meant for joda-time support. That being said, hopefully you’ll be able to deserialize dates with plain Jackson and not need that particular module. Cheers, Eugen.

+ 0 −

2 years ago

Guest

Kisna

Yes, for the older Java dates, a custom serializer that leverages the efficient JODA API by caching date formats. Is there a good way to inject this JODA based serializer for the old java.util.Date class only instead of mentioning the serializer annotation so many times? Just trying to figure out a best practice to use JODA for all Dates..

+ 0 −

2 years ago

Guest

Charles Li

It is very useful, thanks!~

+ 0 −

2 years ago

Guest

Ben Friedman

Eugen, thank you for these good examples. Any chance you can show your import statements? Or use fully qualified types? For example, JsonParser is a class in Jackson and Spring Boot. I presume this usage is for Jackson, not Spring Boot, yes?

+ 0 −

2 years ago

Guest

Eugen Paraschiv (http://www.baeldung.com/)

Hola Ben, sí, es la clase Jackson. Normalmente no incluyo las declaraciones de importación, pero en este caso, si hay confusión, tendría sentido. Pero hasta que lo haga, sí, esa es la importación correcta aquí.
Saludos,
Eugen.

+ 0 −

Hace 2 años

Cargar más comentarios

¿Cuál de estos es el más cercano a su trabajo / función actual?

Desarrollador

Desarrollador Senior

Desarrollador principal

Arquitecto

Gerente

CATEGORÍAS

PRIMAVERA (HTTP://WWW.BAELDUNG.COM/CATEGORY/SPRING/)

DESCANSO (HTTP://WWW.BAELDUNG.COM/CATEGORY/REST/)
JAVA (HTTP://WWW.BAELDUNG.COM/CATEGORY/JAVA/)
SEGURIDAD (HTTP://WWW.BAELDUNG.COM/CATEGORY/SECURITY-2/)
PERSISTENCIA (HTTP://WWW.BAELDUNG.COM/CATEGORY/PERSISTENCE/)
JACKSON (HTTP://WWW.BAELDUNG.COM/CATEGORY/JACKSON/)
HTTPCLIENT (HTTP://WWW.BAELDUNG.COM/CATEGORY/HTTP/)
KOTLIN (HTTP://WWW.BAELDUNG.COM/CATEGORY/KOTLIN/)

SERIE

TUTORIAL "VOLVER A LO BÁSICO" DE JAVA (HTTP://WWW.BAELDUNG.COM/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (HTTP://WWW.BAELDUNG.COM/JACKSON)
TUTORIAL DE HTTPCLIENT 4 (HTTP://WWW.BAELDUNG.COM/HTTPCLIENT-GUIDE)
REST CON SPRING TUTORIAL (HTTP://WWW.BAELDUNG.COM/REST-WITH-SPRING-SERIES/)
TUTORIAL DE SPRING PERSISTENCE (HTTP://WWW.BAELDUNG.COM/PERSISTENCE-WITH-SPRING-SERIES/)
SEGURIDAD CON SPRING (HTTP://WWW.BAELDUNG.COM/SECURITY-SPRING)

ACERCA DE

ACERCA DE BAELDUNG (HTTP://WWW.BAELDUNG.COM/ABOUT/)
LOS CURSOS (HTTP://COURSES.BAELDUNG.COM)
TRABAJO DE CONSULTORÍA (HTTP://WWW.BAELDUNG.COM/CONSULTING)
META BAELDUNG (HTTP://META.BAELDUNG.COM/)
EL ARCHIVO COMPLETO (HTTP://WWW.BAELDUNG.COM/FULL_ARCHIVE)
ESCRIBIR PARA BAELDUNG (HTTP://WWW.BAELDUNG.COM/CONTRIBUTION-GUIDELINES)
CONTACTO (HTTP://WWW.BAELDUNG.COM/CONTACT)
INFORMACIÓN DE LA COMPAÑÍA (HTTP://WWW.BAELDUNG.COM/BAELDUNG-COMPANY-INFO)
TÉRMINOS DE SERVICIO (HTTP://WWW.BAELDUNG.COM/TERMS-OF-SERVICE)
POLÍTICA DE PRIVACIDAD (HTTP://WWW.BAELDUNG.COM/PRIVACY-POLICY)
EDITORES (HTTP://WWW.BAELDUNG.COM/EDITORS)
KIT DE MEDIOS (PDF) (HTTPS://S3.AMAZONAWS.COM/BAELDUNG.COM/BAELDUNG+-+MEDIA+KIT.PDF)