Lección 9 - Serialización de mensajes

Tecnologías del sector financiero

Asier Sampietro Alberdi

Práctica 1: JAXB

Con el XSD de referencia, se han añadido las correspondientes etiquetas a las clases ReferenceData e Instrument. De esta manera, los elementos extraídos del XML se han podido mapear al las clases de Java.

Después de la ejecución se ha obtenido el siguiente resultado:

```
19:11:52.147 [com.cnebrera.uc3.tech.lesson9.jaxb.JaxbTest.main()] DEBUG com.cnebrera.uc3.tech.lesson9.jaxb.JaxbTest
- [Practica 1] Size of referenceData instrument list: 5
19:11:52.151 [com.cnebrera.uc3.tech.lesson9.jaxb.JaxbTest.main()] DEBUG com.cnebrera.uc3.tech.lesson9.jaxb.JaxbTest
- [Practica 1] Algorithm identifier: TWAP
19:11:52.151 [com.cnebrera.uc3.tech.lesson9.jaxb.JaxbTest.main()] DEBUG com.cnebrera.uc3.tech.lesson9.jaxb.JaxbTest
- [Practica 1] Algorithm marketId: 1
```

Práctica 2: JSON

A las clases anteriores, les ha sido asignado más etiquetas para ser compatibles con los mapeadores de JSON.
En la ejecución, al tratarse de datos similares, se ha comparado el objeto obtenido desde el XML con el obtenido desde JSON, obteniendo el siguiente resultado:

```
19:16:33.648 [com.cnebrera.uc3.tech.lesson9.json.JsonTest.main()] DEBUG com.cnebrera.uc3.tech.lesson9.json.JsonTest
- [Practica 1] Size of referenceData instrument list: 5
19:16:33.651 [com.cnebrera.uc3.tech.lesson9.json.JsonTest.main()] DEBUG com.cnebrera.uc3.tech.lesson9.json.JsonTest
- [Practica 1] Algorithm identifier: TWAP
19:16:33.651 [com.cnebrera.uc3.tech.lesson9.json.JsonTest.main()] DEBUG com.cnebrera.uc3.tech.lesson9.json.JsonTest
- [Practica 1] Algorithm marketId: 1
19:16:33.702 [com.cnebrera.uc3.tech.lesson9.json.JsonTest.main()] DEBUG com.cnebrera.uc3.tech.lesson9.json.JsonTest
- [Practica 2] Json Serializer [true]
```

Práctica 3: Protobuf

Usando el framework de Google, se ha completado primero el archivo proto. Este archivo es el encargado de definir cómo serán la clases posteriormente generada, así que se le han asignado los mismos valores del XML.
Después de completar el proto, se ha compilado dicho archivo para obtener su equivalente en Java con el siguiente comando:

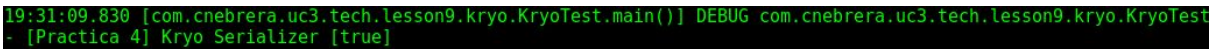
```
sampru@debian:/media/sampru/Datuak/eskola/tec-sec-fin/Tema9/protoc-3.0.0-linux-x
86_64/bin$ protoc --java_out=/media/sampru/Datuak/eskola/tec-sec-fin/MasterUC3Pr
actices/Lesson9/src/main/java/ /media/sampru/Datuak/eskola/tec-sec-fin/MasterUC3
Practices/Lesson9/src/main/resources/lesson9.proto --proto_path=/.
```

Habiendo obtenido la clase, se ha modificado el test para rellenar el proto con los valores extraídos desde el XML, y después se ha comparado con la deserialización de la serialización de este, asegurando que el proceso deja un objeto idéntico. El resultado es la siguiente traza:

```
19:25:32.590 [com.cnebrera.uc3.tech.lesson9.proto.ProtoTest.main()] DEBUG com.cnebrera.uc3.tech.lesson9.proto.ProtoT
est - [Practica 3] Proto Serializer [true]
```

Práctica 4: Kryo

Para esta práctica no se han necesitado modificaciones, simplemente se ha ejecutado el simple código que deja esta librería para ver que la serialización es sencilla con librerías dedicadas.
El output stream después de la ejecución ha sido similar al de protocol buffer, ya que hacen lo mismo:



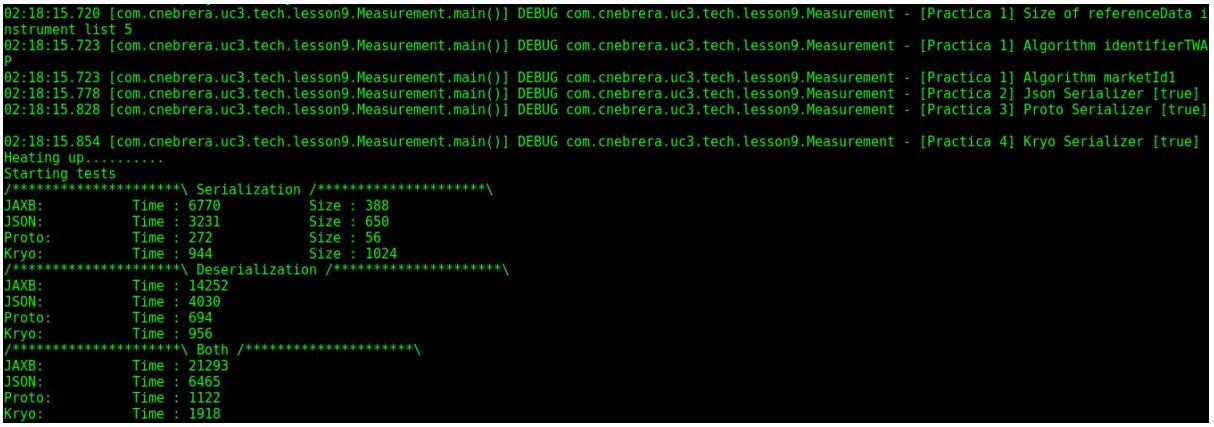
Práctica 5: Comparación de rendimiento

Teniendo claro cómo se serializan las clases en diferentes formas, se ha procedido a realizar una medición del rendimiento de estos cuatro métodos.
Para la creación de la “test suite”, primero se han se han ejecutado los cuatro casos anteriores para tener los objetos y binarios intermedios. Después se ha hecho una ejecución de calentamiento, para que la compilación a tiempo real optimice las operaciones y se obtengan resultados arbitrarios (ya que sin ‘warm up’ inicial, la primera fase hacia como tal, obteniendo unos resultados desconcertantes).
Las mediciones se han hecho sobre la media del tiempo de ejecución en nanosegundos de 10000 ejecuciones de serialización, deserialización y combinación de ambas. En el caso de la serialización, también se ha medido el tamaño que obtendría el binario, para hacerse a la idea de lo que puede tardar el paquete mientras viaja.

Los resultados han sido los siguientes:

	Serialización	**Deserialización**	**Ambas**	**Tamaño binario**
JAXB	6770 ns	14252 ns	21293 ns	388
JSON	3231 ns	4030 ns	6465 ns	650
Proto	272 ns	694 ns	1122 ns	56
Kryo	944 ns	956 ns	1918 ns	1024

Se adjunta también una captura de la ejecución:



Conclusiones

Excluyendo protocol buffer de estas, todas las librerías pecan de tiempo o de tamaño de paquete. El precio a pagar por la rapidez es el coste de un binario pesado, y de la misma manera, para simplificar el binario se incrementan los costes de tiempo.
Excluido de este grupo queda el método de Google, que muestra un desempeño superior y un binario mínimo, debido a su implementación. Además de esto, cabe destacar que es portable a varios lenguajes, así que en estos casos ofrece una serie de ventajas que ayudan a decantarse por esta librería.