Runtime Infrastructure - Feature #3789

rewrite writeObject and readObject to rely on NativeTypeSerializer or a similar approach

11/07/2018 11:33 AM - Constantin Asofiei

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Igor Skornyakov	**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**version:**	
vendor_id:	GCD		

Description

Related issues:

Related to Runtime Infrastructure - Feature #4026: ensure all objects transmi...	**Closed**

History

#1 - 11/07/2018 11:45 AM - Constantin Asofiei

Created task branch 3789a from trunk rev 11289.

This includes an initial approach of using NativeTypeSerializer to send native arrays over the socket.

Some other cases which can be improved are:

- String serialization - many cases use readObject and writeObject so that a null value can be handled. This can be replaced with a NativeTypeSerializer API which relies on readUTF and writeUTF (see bellow how to handle null values).
- null values - even NativeTypeSerializer relies on readObject and writeObject to handle null values - this can be replaced by a custom NativeTypeSerialized sub-class which writes only its marker, and it will know to assume a null value when reading the data, instead of serializing the entire object.
- BaseDataType instances - there are cases where read/writeObject is used to serialize a BDT (see PutField.writeExternal for an example); this can be solved writing a specific sub-class of NativeTypeSerializer, which writes its custom marker first, the class name to be instantiated, and after that relies on read/writeExternal to serialize the BDT.

The idea behind this performance optimization is that writing and reading fully serialized objects is expensive in Java - we should rely on the Externalizable approach always; write/readObject will at some point use the write/readExternal, if the object is a Externalizable, but cutting down this overhead should help.

This performance issue was initially observed in #3741, where a testcase relies on lots of PUT statements (10s of thousands).

#2 - 11/07/2018 11:50 AM - Constantin Asofiei

The native array serializing changes are in 3789a rev 11290.

#3 - 11/07/2018 01:24 PM - Constantin Asofiei

Another part to think about is the read/writeObject in Protocol$Reader and Writer classes - this uses standard object serialization, because it doesn't know what kind of objects are sent as messages.

I'm not sure why this approach was taken, but in our case messages are taken via Queue.dequeueOutbound, and this always returns a Message instance. In turn, Message instance is Externalizable, but it contains a Serializable payload - which relies on read/WriteObject.

I think instead of a BDT specific serializer we should write a generic serializer for objects:

- if the instance is a Externalizable, it will work almost the same: write the marker, a flag that this is Externalizable, the class name to be instantiated and read/write the data using the read/writeExternal APIs.
- otherwise, write the marker, and a false flag which will inform that we are not an Externalizable, and rely on read/writeObject to get the object.

#4 - 04/01/2019 09:53 PM - Eric Faulhaber

- Related to Feature #4026: ensure all objects transmitted over the DAP implement Externalizable added

#5 - 07/19/2020 03:19 PM - Greg Shah

- Assignee set to Igor Skornyakov

#6 - 07/21/2020 10:06 AM - Igor Skornyakov

A pure "hand-made" serialization (even with simple helper classes such as NativeTypeSerializer) has at least two disadvantages:

- It required a lot of coding.
- Such serialization is difficult to maintain.

Maybe it is worth considering one of the Java serialization libraries? See a comparison of some at https://github.com/eishay/jvm-serializers/wiki. I had an experience with one such library in the past - Kryo (https://github.com/iskorn/kryo). It was an application where the performance of the (de)serialization was very important and there were a lot of objects which required this. The library proved to be really very efficient.
Please note that a "shaded" version of Kryo is already indirectly used by FWD (it is a part or the gremlin-shaded-3.2.3.jar).

#7 - 07/21/2020 10:14 AM - Constantin Asofiei

Igor, how would Message.payload be changed to use Kryo?

#8 - 07/21/2020 10:27 AM - Igor Skornyakov

Constantin Asofiei wrote:

> Igor, how would Message.payload be changed to use Kryo?

Constantin,
I've not analysed Message.payload yet. The Kryo (or other library) can be used to simplify implementation of the Externalizable interface. In this case there still will be some overhead in using writeObject for the Message.payload serialization but it will be just the names of the classes if the payload is an array of Object if all objects are serialized efficiently.