Debugging CDS DeterministicDump.java test failures

We want the JDK build to be "**reproducible**". This means that when building the JDK from the same source code, we want to (as much as possible) generate the exact same set of binaries that are bit-by-bit identical. The is useful in validating the JDK build process.

In CDS, after JDK-8241071 (Generation of classes.jsa with -Xshare:dump is not deterministic), running "java -Xshare:dump" twice should generate the exact same classes.jsa.

However, if you make changes in CDS, sometimes reproducibility may be broken, causing the DeterministicDump.java may fail. The usual causes are:

- You didn't sort objects before writing. See some notes here.
- The objects contains random data that varies from run to run of the JVM.

To debug the failure, use the following script and gdb:

```
#! /bin/bash
#
# Run "java -Xshare:dump" twice. They should generate the exact same classes.jsa.
# See https://bugs.openjdk.java.net/browse/JDK-8241071
#
# Usage: bash compare_cds_dump.sh $JAVA_HOME
#
# If "xxd" is not available, use "hexdump" instead.
#
# "xxd -g 8" is used because most of the time we have mismatched (64-bit) pointers.
# you may want to change to "-g 4", etc, if the mismatch is caused by fields
# of other sizes.

JAVA=$1/bin/java
shift

ARGS="-Xshare:dump -Xlog:cds=debug -Xmx256m"
$JAVA $ARGS -XX:SharedArchiveFile=1.jsa $* | egrep " crc " | cut -b 9- | tee 1.txt
cksum 1.jsa | tee -a 1.txt
xxd -e -g 8 1.jsa >> 1.txt
#hexdump -C 1.jsa >> 1.txt

$JAVA $ARGS -XX:SharedArchiveFile=2.jsa $* | egrep " crc " | cut -b 9- | tee 2.txt
cksum 2.jsa | tee -a 2.txt
xxd -e -g 8 2.jsa >> 2.txt
#hexdump -C 2.jsa >> 2.txt

(set -x; diff 1.txt 2.txt | wc)
```

Example

Often time we break reproducibility by writing an unprocessed native pointer into the archive. For example, **InstanceKlass::_package_entry** should be set to NULL before the InstanceKlass is copied. We can disable this by:

```
diff -r ef14216b4fc6 src/hotspot/share/oops/instanceKlass.cpp
--- a/src/hotspot/share/oops/instanceKlass.cpp        Fri Jul 24 13:56:45 2020 -0700
+++ b/src/hotspot/share/oops/instanceKlass.cpp        Sat Jul 25 14:23:41 2020 -0700
@@ -2541,7 +2541,7 @@
   _oop_map_cache = NULL;
   // clear _nest_host to ensure re-load at runtime
   _nest_host = NULL;
-  _package_entry = NULL;
+  //_package_entry = NULL;
   _dep_context_last_cleaned = 0;
 }
```

Running compare_cds_dump.sh shows the following output:

```
$ bash compare_cds_dump.sh linux-x64-slowdebug/images/jdk
[debug][cds] Shared file region (mc )  0:    25168 bytes, addr 0x0000000800000000 file offset 0x00001000 crc
0x47b1156b
[debug][cds] Shared file region (rw )  1:  4417824 bytes, addr 0x0000000800007000 file offset 0x00008000 crc
0x9f715f77
[debug][cds] Shared file region (ro )  2:  7688712 bytes, addr 0x000000080043e000 file offset 0x0043f000 crc
0xdee411d1
[debug][cds] Shared file region (bm )  3:   216008 bytes, addr 0x0000000000000000 file offset 0x00b95000 crc
0x79d7f089
[debug][cds] Shared file region (ca0)  4:   507904 bytes, addr 0x00000000fff00000 file offset 0x00bca000 crc
0x611e7b9a
[debug][cds] Shared file region (oa0)  6:   335872 bytes, addr 0x00000000ffe00000 file offset 0x00c46000 crc
0x1356a695
1030720421 13205504 1.jsa
[debug][cds] Shared file region (mc )  0:    25168 bytes, addr 0x0000000800000000 file offset 0x00001000 crc
0x47b1156b
[debug][cds] Shared file region (rw )  1:  4417824 bytes, addr 0x0000000800007000 file offset 0x00008000 crc
0x17aaa251
[debug][cds] Shared file region (ro )  2:  7688712 bytes, addr 0x000000080043e000 file offset 0x0043f000 crc
0xdee411d1
[debug][cds] Shared file region (bm )  3:   216008 bytes, addr 0x0000000000000000 file offset 0x00b95000 crc
0x79d7f089
[debug][cds] Shared file region (ca0)  4:   507904 bytes, addr 0x00000000fff00000 file offset 0x00bca000 crc
0x611e7b9a
[debug][cds] Shared file region (oa0)  6:   335872 bytes, addr 0x00000000ffe00000 file offset 0x00c46000 crc
0x1356a695
2074642362 13205504 2.jsa
+ diff 1.txt 2.txt
+ wc
   4766   14564  173338
$ diff 1.txt 2.txt | less
2c2
< [debug][cds] Shared file region (rw )  1:  4417824 bytes, addr 0x0000000800007000 file offset 0x00008000 crc
0x9f715f77
---
> [debug][cds] Shared file region (rw )  1:  4417824 bytes, addr 0x0000000800007000 file offset 0x00008000 crc
0x17aaa251
7,8c7,8
< 1030720421 13205504 1.jsa
< 00000000: cc91928ff00baba2 000000000000000a  ................
---
> 2074642362 13205504 2.jsa
> 00000000: 8be26863f00baba2 000000000000000a  ....ch..........
13c13
< 00000050: 0000000000000000 000000009f715f77  ........w_q.....
---
> 00000050: 0000000000000000 0000000017aaa251  ........Q.......
2103c2103
< 000082f0: 00007fe2bc17ce90 0000000000000000  ................
---
> 000082f0: 00007fe1d017ce90 0000000000000000  ................
2137c2137
< 00008510: 00007fe2bc17ce90 0000000800007878  ........xx......
---
> 00008510: 00007fe1d017ce90 0000000800007878  ........xx......
[...... snip ......]
```

The first few lines are just differences in the CRC values. The first real difference appears at offset 0x000082f0 in the JSA file.

To find out what's written into this location, we first need to find out the address of the above offset. The can be found from these lines in 1.txt

```
[debug][cds] Shared file region (mc )  0:    25168 bytes, addr 0x0000000800000000 file offset 0x00001000 crc
0x47b1156b
[debug][cds] Shared file region (rw )  1:  4417824 bytes, addr 0x0000000800007000 file offset 0x00008000 crc
0x9f715f77
[debug][cds] Shared file region (ro )  2:  7688712 bytes, addr 0x000000080043e000 file offset 0x0043f000 crc
0xdee411d1
[debug][cds] Shared file region (bm )  3:   216008 bytes, addr 0x0000000000000000 file offset 0x00b95000 crc
0x79d7f089
[debug][cds] Shared file region (ca0)  4:   507904 bytes, addr 0x00000000fff00000 file offset 0x00bca000 crc
0x611e7b9a
[debug][cds] Shared file region (oa0)  6:   335872 bytes, addr 0x00000000ffe00000 file offset 0x00c46000 crc
0x1356a695
```

This basically means that the address for offset **X** is **0x0000000800000000 + X - 0x1000**. I.e., the address for offset **0x000082f0** is **0x800007 2f0.**

We can now run -Xshare:dump inside gdb:

```
$ gdb --args linux-x64-slow-debug/images/jdk/bin/java -Xshare:dump
(gdb) watch *(int**)0x8000072f0
Hardware watchpoint 1: *(int**)0x8000072f0
(gdb) r

Thread 8 "VM Thread" hit Hardware watchpoint 1: *(int**)0x8000072f0

Old value = (int *) 0x0
New value = (int *) 0x7ffff0205a90
(gdb) where
#0  __memmove_ssse3_back () at ../sysdeps/x86_64/multiarch/memcpy-ssse3-back.S:175
#1  0x00007ffff64fc59e in ArchiveCompactor::allocate (ref=0x7ffee00064b0, read_only=false)
#2  0x00007ffff64fc827 in ArchiveCompactor::ShallowCopier::do_unique_ref (this=0x7fffbc262740,
ref=0x7ffee00064b0, read_only=false)
#3  0x00007ffff64f2256 in UniqueMetaspaceClosure::do_ref (this=0x7fffbc262740, ref=0x7ffee00064b0,
read_only=false)
#4  0x00007ffff64f1fe4 in MetaspaceClosure::do_push (this=0x7fffbc262740, ref=0x7ffee00064b0)
....
```

So this happens when an object is being copied into the CDS archive:

```
  static void allocate(MetaspaceClosure::Ref* ref, bool read_only) {
    address obj = ref->obj();
    int bytes = ref->size() * BytesPerWord;
    ....
      p = _rw_region.allocate(bytes, alignment);
    ....
    memcpy(p, obj, bytes);
```

We can figure out the type of the object by doing this:

```
(gdb) p ref->msotype()
$2 = MetaspaceObj::ClassType
(gdb) p ((Klass*)obj)->_name
$3 = (Symbol *) 0x7ffee8000190
(gdb) p ((Klass*)obj)->_name->print()
Symbol: 'java/lang/Cloneable' count 65535$11 = void
```

p is the location of the copy of the InstanceKlass, and the offending address **0x8000072f0** is offset 224 into this copy, which is the as the field offset of **InstanceKlass::_package_entry**.

```
(gdb) p p
$5 = 0x800007210 ""
(gdb) p 0x8000072f0 - 0x800007210
$6 = 224
(gdb) p (int)&(((InstanceKlass*)0)->_package_entry)
$9 = 224
```

From the above, we can conclude that:

- The problem is caused by **InstanceKlass::_package_entry**
- The offending value (**0x7ffff0205a90**) looks like an object allocated from the C-HEAP on Linux. C-HEAP allocations vary from run to run of the JVM. That's why we get two different JSA files from two **-Xshare:dump** runs.