Ian Partridge <i.partridge@uk.ibm.com> – Java L3 Service
23rd March 2011

Class loading and debugging class loader memory leaks in WebSphere Application Server

Agenda

- Understanding classloaders
  - How classes are loaded
  - Memory usage of classes
  - Strong references
  - How classes are unloaded
- Interactive session with MAT
- Common classloader leaks

Java classloading

- Every object in a Java program is an instance of a class
- Every class in a Java program is loaded by a classloader

- Classloading can be:
  - Explicit e.g. java/lang/Class.forName() from Java code
  - Implicit e.g. loading dependent classes like superclasses, interfaces etc.

- java.lang.Classloader has 2 constructors
  - Classloader(Classloader parent) Use the given classloader as parent
  - Classloader() Use the system classloader as parent
- Classloaders are a tree, with the system/bootstrap classloader at the top

Common classloaders

- System/Bootstrap classloader
  - Loads most of the Java standard libraries, e.g. java.lang.* , java.io.* etc.
  - com.ibm.oti.vm.BootstrapClassLoader
- Extension classloader
  - JARs in the jre/lib/ext directory
  - sun.misc.Launcher$ExtClassLoader
- Application classloader
  - Loads classes from your classpath
  - sun.misc.Launcher$AppClassLoader
- WAS uses OSGi to manage classloading in the WAS runtime
  - org.eclipse.osgi.internal.baseadaptor.DefaultClassLoader
- Each WAS application is loaded in its own classloader
  - com.ibm.ws.classloader.CompoundClassLoader

How does classloading work?

- Classloading uses *parent-first delegation*

- When java/lang/ClassLoader.loadClass() is called, it:

  - Invokes findLoadedClass() to check if the class is already loaded by this classloader
    - If yes, just return the class
    - If no...
  - Invokes loadClass() on its *parent* classloader
    - Parent-first means each classloader up the tree is asked whether they have already loaded the class
    - If no...
  - Invokes findClass() to find and load the class

Why won't my class load?

- ClassNotFoundException
  - The given class could not be found.
- NoClassDefFoundError
  - A ClassNotFoundException was generated when loading a dependent class
- ClassCircularityError
  - For example, if loading a superclass calls defineClass() for the original class
- ClassFormatError
  - Bad bytes in your .class file, e.g. no CAFEBABE
- UnsupportedClassVersionError
  - Java code compiled using javac from Java 6 but you're running on Java 5?
- UnsatisfiedLinkError
  - Native library cannot be loaded, or a JNI method is called but the symbol is unknown
- VerifyError
  - Bytecodes are not valid according to the Java specification

Useful classloading options

- -verbose:class

```
class load: java/util/zip/ZipEntry
class load: java/util/jar/JarEntry
class load: java/util/jar/JarFile$JarFileEntry
class load: java/net/URLConnection
```

- -verbose:dynload

```
<Loaded java/lang/Object from C:\Program Files\IBM\Java60\jre\lib\vm.jar>
<  Class size 1555; ROM size 1688; debug size 0>
<  Read time 67 usec; Load time 54 usec; Translate time 57 usec>
```

- -Dibm.cl.verbose=* (only sees the ExtClassloader downwards)

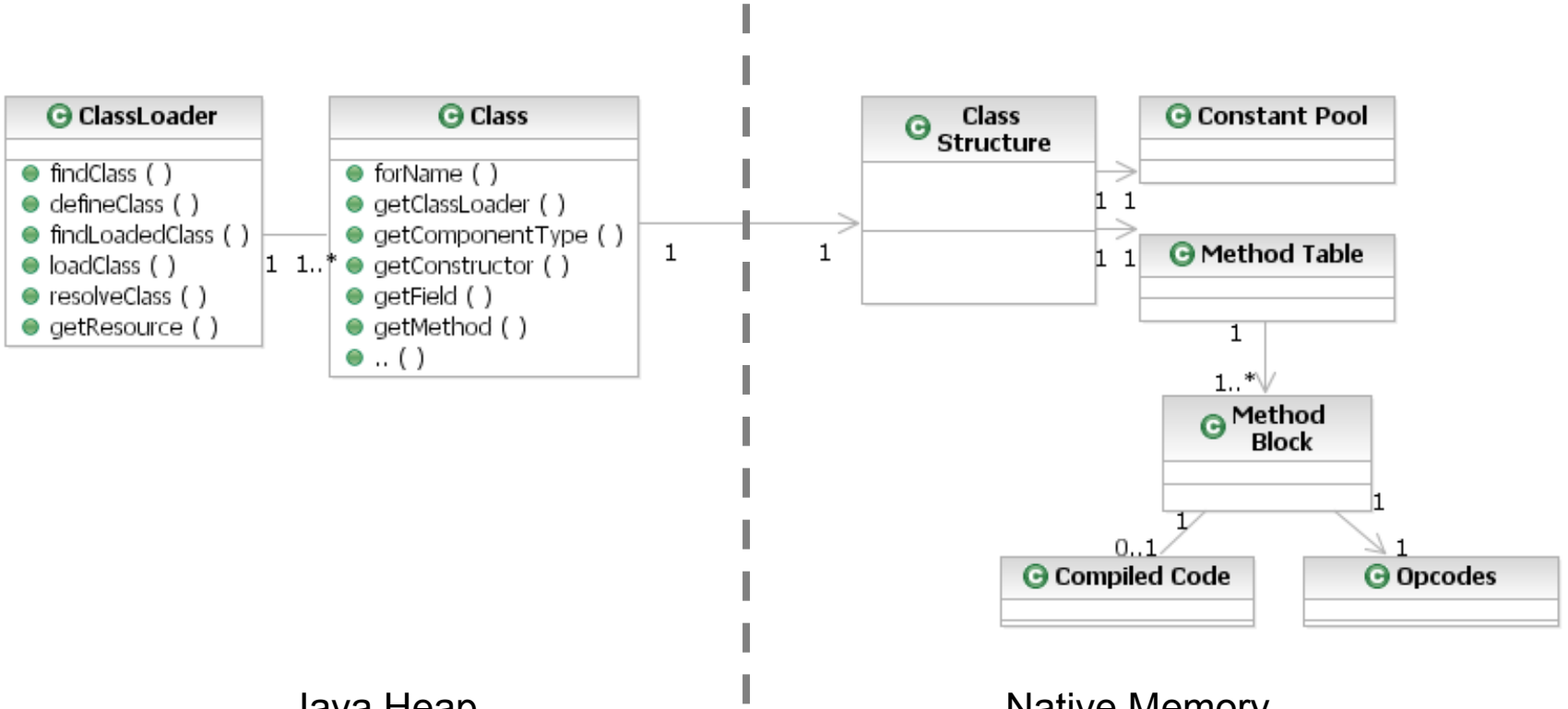
```
ExtClassLoader attempting to find MyClass
ExtClassLoader using classpath [.....]
ExtClassLoader could not find MyClass.class in C:\Program%20Files\IBM\Java60\jre\lib\ext\dtfj.jar
[.....]
ExtClassLoader could not find MyClass

AppClassLoader attempting to find MyClass
AppClassLoader using classpath C:\Users\Ian
AppClassLoader found MyClass.class in C:\Users\Ian
AppClassLoader found MyClass
```

Memory usage of classes

- Classes use both native and Java heap memory
- Java heap
  - The instance of java.lang.Class itself
  - All instances of the class
- Native heap
  - Bytecodes
  - Constant pool
- Native class memory is allocated in segments, visible in javacore files

```
1STSEGTYPE      Class Memory
NULL            segment           start             alloc             end               type     bytes
1STSEGMENT      00002AAC43B360F8 00002AAC4297C888 00002AAC4297CB78 00002AAC4297CB78 00010040 2f4
1STSEGMENT      00002AAC43B36038 00002AAC4130E648 00002AAC4130EA10 00002AAC4130FCA8 00020040 1660
1STSEGMENT      00002AAC43B35F78 00002AAC42BCF0E8 00002AAC42BCF3D0 00002AAC42BCF3D0 00010040 2ec
1STSEGMENT      00002AAC43B35EB8 00002AAC429C34A8 00002AAC429C3868 00002AAC429C4AA8 00020040 1600
1STSEGMENT      00002AAC43B35978 00002AAC3F96AA78 00002AAC3F96AD60 00002AAC3F96AD60 00010040 2ec
…
```

Class memory usage



Classloader references

- A Java object has a strong reference to its class — object.getClass()
- A class has a strong reference to its classloader — class.getClassloader()
- A classloader has a strong reference to every class it has loaded — classloader.findLoadedClass()

- All these references are strong!

Java classunloading

- Java classes are loaded *per-class*, but unloaded *per-classloader*

- The garbage collector decides when to run classunloading
  - In “gencon”, only occurs on a global GC
  - Classunloading activity is shown in -verbose:gc

- Classunloading can be denied for three reasons:
  1) Live references to the classloader
  2) Live references to a class loaded by the classloader
  3) Live references to objects of classes loaded by the classloader

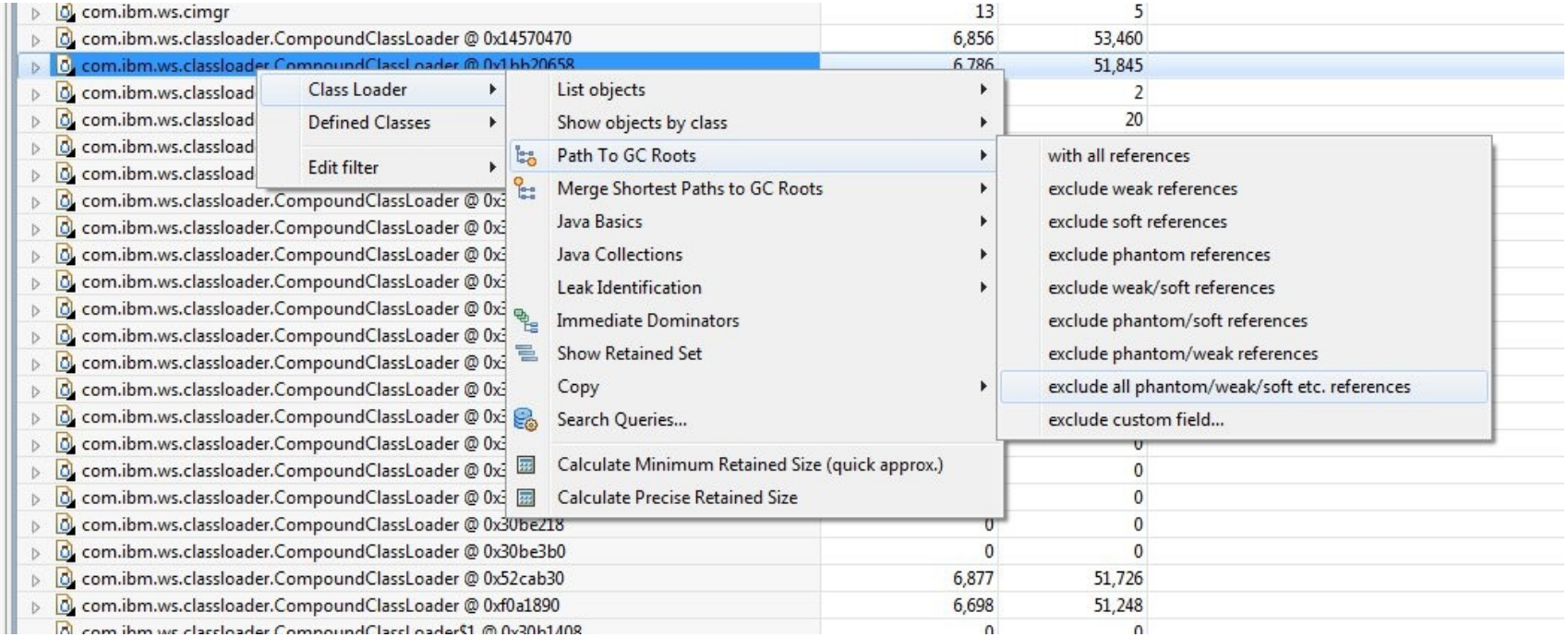
- References can come from anywhere:
  - Other objects
  - Other classes/classloaders
  - Thread stacks
  - Thread variables
  - JNI global references
  - Finalizer queue entries

-verbose:gc example

```
<af type="tenured" id="8" timestamp="Oct 12 16:28:25 2010" intervalms="16316214.370">
  <minimum requested_bytes="32" />
  <time exclusiveaccessms="0.266" meanexclusiveaccessms="0.216" threads="1" lastthreadtid="0x0000000030B99100" />
  <refs soft="4659" weak="181586" phantom="7461" dynamicSoftReferenceThreshold="24" maxSoftReferenceThreshold="32" />
  <nursery freebytes="5105205712" totalbytes="5973037056" percent="85" />
  <tenured freebytes="0" totalbytes="4026028032" percent="0" >
    <soa freebytes="0" totalbytes="4026028032" percent="0" />
    <loa freebytes="0" totalbytes="0" percent="0" />
  </tenured>
  <gc type="global" id="98" totalid="11383" intervalms="2392662.819">
    <classunloading classloaders="1391" classes="1391" timevmquiescems="0.000" timetakenms="229.237" />
    <finalization objectsqueued="1538" />
    <timesms mark="716.356" sweep="14.879" compact="0.000" total="1522.929" />
    <nursery freebytes="5138270304" totalbytes="5973037056" percent="86" />
    <tenured freebytes="3361335616" totalbytes="4026028032" percent="83" >
      <soa freebytes="3361335616" totalbytes="4026028032" percent="83" />
      <loa freebytes="0" totalbytes="0" percent="0" />
    </tenured>
  </gc>
  <nursery freebytes="5138270304" totalbytes="5973037056" percent="86" />
  <tenured freebytes="3361335584" totalbytes="4026028032" percent="83" >
    <soa freebytes="3361335584" totalbytes="4026028032" percent="83" />
    <loa freebytes="0" totalbytes="0" percent="0" />
  </tenured>
  <refs soft="4466" weak="99084" phantom="4741" dynamicSoftReferenceThreshold="26" maxSoftReferenceThreshold="32" />
  <time totalms="1524.051" />
</af>
```

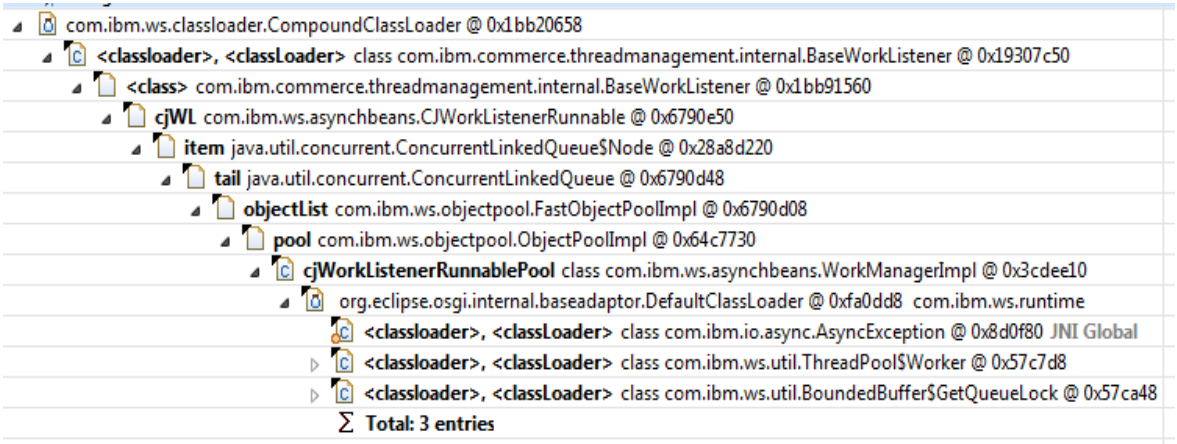
Identifying an application classloader leak

- Most common first symptom is OutOfMemoryError
  - Can be either Java heap or native!
  - Collect a system core and open in MAT
- Use the “Classloader explorer” and find the application classloaders whose localClassPath is not set
- For each, run “Class Loader -> Path to GC Roots -> exclude all phantom/weak/soft etc. references”



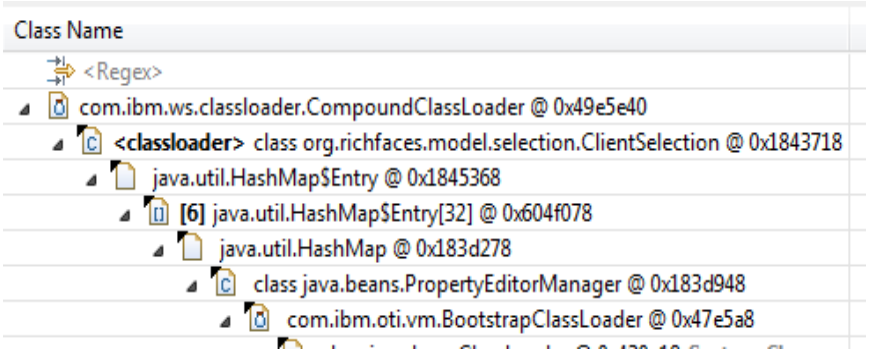
Unwanted reference to an object whose class was loaded by the application classloader – example 1

- Here, a CompoundClassLoader is kept alive because an instance of an object whose class was loaded by it has been cached
- Question to be answered:
  - Who adds and removes entries to “cjWorkListenerRunnablePool” in WorkManagerImpl?
- Notes:
  - “cjWorkListenerRunnablePool” is a static field in WorkManagerImpl
  - BaseWorkListener is loaded by the application classloader
- This is APAR PM25457 – fixed in WAS 7.0.0.17



Unwanted reference to a class which was loaded by the application classloader – example 2

- Here, a CompoundClassLoader is kept alive because a class it loaded – org.richfaces.model.selection.ClientSelection – has been stored inside a HashMap in the system class java.beans.PropertyEditorManager
- Because this is a system class, it has javadoc!
- Looks like a RichFaces bug...
  - https://jira.jboss.org/browse/RF-7911
    - "OutOfMemory when redeploying - ClientSelection not unregistered from PropertyEditorManager"



Method Detail

registerEditor

```
public static void registerEditor(Class<?> targetType,
                                  Class<?> editorClass)
```

Register an editor class to be used to edit values of a given target class.

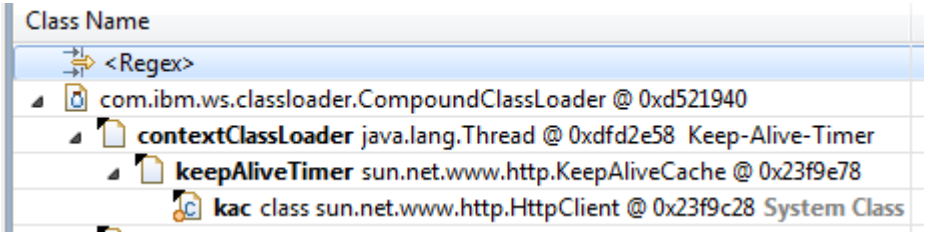
First, if there is a security manager, its `checkPropertiesAccess` method is called. This could result in a SecurityException.

Parameters:

`targetType` - the Class object of the type to be edited
`editorClass` - the Class object of the editor class. If this is null, then any existing definition will be removed.

Unwanted reference to application classloader – example 3

- Here, we have a CompoundClassLoader which is kept alive because a thread named “Keep-Alive-Timer” has its contextClassLoader set to it
- “Keep-Alive-Timer” is a daemon thread
  - Spawned by the classlibraries
  - Daemon threads live until the JVM ends
- Threads inherit contextclassloader
  - From their parent
- This is a Java classlibrary bug
  - Being raised with Oracle
  - Fix is simple:
    - thread.setContextClassLoader(null);



Statics | Attributes | Class Hierarchy

Type	Name	Value
long	threadRef	0
long	stackSize	0
boolean	started	true
ref	name	Keep-Alive-Timer
int	priority	8
boolean	isDaemon	true

Common leaks

- ThreadLocal problems
  - Custom class extending ThreadLocal?
- Threads' contextClassLoaders
  - Daemon threads started by a servlet
  - Careless use of java.util.Timer
  - Daemon threads started by 3rd party libraries, shared between two applications
- Bean introspection
  - If you introspect (call getBeanInfo()) on a Bean loaded by the app classloader, you must call flushfromCaches(beanClass) on app shutdown
- JMX MBeans and NotificationListeners
  - Must be unregistered when the application stops

Questions?