Non-Portable Classes

About Terracotta Documentation

This documentation is about Terracotta DSO, an advanced distributed-computing technology aimed at meeting special clustering requirements.

Terracotta products without the overhead and complexity of DSO meet the needs of almost all use cases and clustering requirements. To learn how to migrate from Terracotta DSO to standard Terracotta products, see Migrating From Terracotta DSO. To find documentation on non-DSO (standard) Terracotta products, see Terracotta Documentation. Terracotta release information, such as release notes and platform compatibility, is found in Product Information.

Release: 3.6
Publish Date: November, 2011 **Documentation Archive »**

Non-Portable Classes

Introduction

Non-portable classes are not shareable in Terracotta.

There are two types of non-portable classes in Terracotta:

- **Never-Portable** - these classes represent local resources or other non-shareable classes and are never shared.
- **Unsupported** - currently these classes are not supported by Terracotta but may be supported in the future.

Further Reading:
See the Concept and Architecture Guide for more information on portability in Terracotta.

Never-Portable Classes

The following list shows classes that are never shareable. Generally, these classes are not shareable because they are intrinsically associated with a local resource such as a socket, a file descriptor, or a thread:

```
java.awt.Component
java.lang.Thread
java.lang.ThreadGroup
java.lang.Process
java.lang.ClassLoader
java.lang.Runtime
java.io.FileReader
java.io.FileWriter
java.io.FileDescriptor
java.io.FileInputStream
java.io.FileOutputStream
java.net.DatagramSocket
java.net.DatagramSocketImpl
java.net.MulticastSocket
java.net.ServerSocket
java.net.Socket
java.net.SocketImpl
java.nio.channels.DatagramChannel
java.nio.channels.FileChannel
java.nio.channels.FileLock
java.nio.channels.ServerSocketChannel
java.nio.channels.SocketChannel
java.util.logging.FileHandler
java.util.logging.SocketHandler
javax.servlet.GenericServlet (and all subclasses)
```

If your current code base contains references to any of these classes, use the Terracotta setting `honor-transient`. This mechanism specifies whether Terracotta should or should not share a field in a class that is marked with the transient keyword.

Further Reading:
See the Concept and Architecture Guide for more information on transience in Terracotta.
See the Configuration Guide and Reference for more information on configuring transience in Terracotta.

Current List of Unsupported Classes

The following list shows classes that are not shareable in the current and in earlier versions of Terracotta.

```
java.util.concurrent.AbstractExecutorService
java.util.concurrent.ArrayBlockingQueue
java.util.concurrent.ConcurrentLinkedQueue
java.util.concurrent.ConcurrentSkipListSet
java.util.concurrent.ConcurrentSkipListMap
java.util.concurrent.CountDownLatch
java.util.concurrent.DelayQueue
java.util.concurrent.Exchanger
java.util.concurrent.ExecutorCompletionService
java.util.concurrent.Executors
java.util.concurrent.LinkedBlockingDeque (NOTE: Deque not Queue)
java.util.concurrent.PriorityBlockingQueue
java.util.concurrent.ScheduledThreadPoolExecutor
java.util.concurrent.Semaphore
java.util.concurrent.SynchronousQueue
java.util.concurrent.ThreadPoolExecutor
java.util.concurrent.ThreadPoolExecutor.AbortPolicy
java.util.concurrent.ThreadPoolExecutor.CallerRunsPolicy
java.util.concurrent.ThreadPoolExecutor.DiscardOldestPolicy
java.util.concurrent.ThreadPoolExecutor.DiscardPolicy
java.util.concurrent.atomic.AtomicBoolean
java.util.concurrent.atomic.AtomicIntegerArray
java.util.concurrent.atomic.AtomicIntegerFieldUpdater
java.util.concurrent.atomic.AtomicLongArray
java.util.concurrent.atomic.AtomicLongFieldUpdater
java.util.concurrent.atomic.AtomicMarkableReference
java.util.concurrent.atomic.AtomicReferenceArray
java.util.concurrent.atomic.ReferenceFieldUpdater
java.util.concurrent.atomic.AtomicStampedReference
java.util.concurrent.locks.AbstractOwnableSynchronizer
java.util.concurrent.locks.AbstractQueuedLongSynchronizer
java.util.concurrent.locks.AbstractQueuedSynchronizer
java.util.concurrent.locks.LockSupport
java.util.WeakHashMap
java.lang.ref.SoftReference
java.lang.ref.WeakReference
java.lang.ref.PhantomReference
java.lang.ref.Reference
java.lang.ref.ReferenceQueue
```


In many cases, the `java.util.concurrent.*` classes can be replaced by an equivalent class from the oswego concurrent class library.