AvailableLockFactories

When applying changes to an index the IndexWriter needs to acquire an exclusive lock on the Directory.

While both F{{`S}}`Directory and RAM`Directory have sensible defaults you might want to consider an alternative for finetuning of performance, or because you are storing the index on some multiple-node shared resource having issues with file based locking.

⚠️ Using an inappropriate LockFactory will have your index corrupted! Make sure you understood the Lock design.

Core LockFactory implementations

NativeFSLockFactory

Qualified ClassName: `org.apache.lucene.store.NativeFSLockFactory`

This implementation is similar to Simple{{`F}}`SLockFactory, they both extend FSLockFactory and rely on a lock file. This version relies on NIO and the lock files will be properly removed (by the OS) if the JVM has an abnormal exit, but should be avoided on NF`S: make sure you read the javadocs for updated details. Because the lock is obtained for the current process different applications in the same J{{`V}}`M might not be excluded properly; Lucene makes an attempt to prevent this issue using a static reference to owned locks, but when loaded in an isolated classloader or when reloading the same web application this workaround is not effective.

NoLockFactory

Qualified ClassName: `org.apache.lucene.store.NoLockFactory`

As the name reminds, this LockFactory will not do anything. This is useful when you're sure that only one thread at a time will ever attempt to apply changes to the index. Using it could be reasonable when your application is designed in such a way, and you can also guarantee that no other processes will attempt to apply changes. If you're not sure to meet these rare requirements, avoid it.

SimpleFSLockFactory

Qualified ClassName: `org.apache.lucene.store.SimpleFSLockFactory`

This implementation is similar to Native{{`F}}`SLockFactory, they both extend FSLockFactory and rely on a lock file. This is the default implementation of FSDirectory: it will write a lockfile to filesystem when the lock is acquired and delete the file when it's released, making it a safe choice for almost all configurations. The drawback of this implementations is that it might happen when the JV`M holding the lock crashes that you have to manually remove the stale lock file.

SingleInstanceLockFactory

Qualified ClassName: `org.apache.lucene.store.SingleInstanceLockFactory`

Uses an object instance to represent the lock, so this is usefull when you are certain that all modifications to the a given index are running against a single shared in-process Directory instance. This is currently the default locking for R{{`A}}`MDirectory, but it could also make sense on a FS`Directory provided the other processes use the index in read-only.

VerifyingLockFactory

Qualified ClassName: `org.apache.lucene.store.VerifyingLockFactory`

This Lock{{`Factory was designed to test other implementations, it's recommended to read it's javadocs and make use of it when you want to implement your own Lock}}`Factory.

Additional implementations of interest

Infinispan LockFactory

Qualified ClassName: `org.infinispan.lucene.locking.LuceneLockFactory`

Infinispan is an open source data grid platform written in Java: http://www.jboss.org/infinispan

It includes a distributed Directory implementation for Lucene and a Lock{{`Factory. While this Lock}}`Factory implementation was designed initially to be used with Infinispan Directory it might also be useful combined with other Directory implementations.

This Lock implementation uses Infinispan and J{{`Groups behind the scenes to provide auto-discovery at network level, making it interesting for both network shared Directories or local Directories shared by applications in multiple J}}V`Ms to have a reliable lock independent from the file-system; when combined with it's sister Directory you also get nice dynamic reconfiguration features, as nodes will find each-other without having to maintain configuration files.

For more information refer to Infinispan's documentation, especially this wiki page.

⚠️ :TODO: add other implementations here ⚠️