| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/ReadWriteLock.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/locks/ReadWriteLock.html)    [**NO FRAMES**](http://docs.google.com/ReadWriteLock.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#2et92p0) |

## **java.util.concurrent.locks**

Interface ReadWriteLock

**All Known Implementing Classes:** [ReentrantReadWriteLock](http://docs.google.com/java/util/concurrent/locks/ReentrantReadWriteLock.html)

public interface **ReadWriteLock**

A ReadWriteLock maintains a pair of associated [locks](http://docs.google.com/java/util/concurrent/locks/Lock.html), one for read-only operations and one for writing. The [read lock](http://docs.google.com/java/util/concurrent/locks/ReadWriteLock.html#readLock()) may be held simultaneously by multiple reader threads, so long as there are no writers. The [write lock](http://docs.google.com/java/util/concurrent/locks/ReadWriteLock.html#writeLock()) is exclusive.

All ReadWriteLock implementations must guarantee that the memory synchronization effects of writeLock operations (as specified in the [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) interface) also hold with respect to the associated readLock. That is, a thread successfully acquiring the read lock will see all updates made upon previous release of the write lock.

A read-write lock allows for a greater level of concurrency in accessing shared data than that permitted by a mutual exclusion lock. It exploits the fact that while only a single thread at a time (a *writer* thread) can modify the shared data, in many cases any number of threads can concurrently read the data (hence *reader* threads). In theory, the increase in concurrency permitted by the use of a read-write lock will lead to performance improvements over the use of a mutual exclusion lock. In practice this increase in concurrency will only be fully realized on a multi-processor, and then only if the access patterns for the shared data are suitable.

Whether or not a read-write lock will improve performance over the use of a mutual exclusion lock depends on the frequency that the data is read compared to being modified, the duration of the read and write operations, and the contention for the data - that is, the number of threads that will try to read or write the data at the same time. For example, a collection that is initially populated with data and thereafter infrequently modified, while being frequently searched (such as a directory of some kind) is an ideal candidate for the use of a read-write lock. However, if updates become frequent then the data spends most of its time being exclusively locked and there is little, if any increase in concurrency. Further, if the read operations are too short the overhead of the read-write lock implementation (which is inherently more complex than a mutual exclusion lock) can dominate the execution cost, particularly as many read-write lock implementations still serialize all threads through a small section of code. Ultimately, only profiling and measurement will establish whether the use of a read-write lock is suitable for your application.

Although the basic operation of a read-write lock is straight-forward, there are many policy decisions that an implementation must make, which may affect the effectiveness of the read-write lock in a given application. Examples of these policies include:

* Determining whether to grant the read lock or the write lock, when both readers and writers are waiting, at the time that a writer releases the write lock. Writer preference is common, as writes are expected to be short and infrequent. Reader preference is less common as it can lead to lengthy delays for a write if the readers are frequent and long-lived as expected. Fair, or "in-order" implementations are also possible.
* Determining whether readers that request the read lock while a reader is active and a writer is waiting, are granted the read lock. Preference to the reader can delay the writer indefinitely, while preference to the writer can reduce the potential for concurrency.
* Determining whether the locks are reentrant: can a thread with the write lock reacquire it? Can it acquire a read lock while holding the write lock? Is the read lock itself reentrant?
* Can the write lock be downgraded to a read lock without allowing an intervening writer? Can a read lock be upgraded to a write lock, in preference to other waiting readers or writers?

You should consider all of these things when evaluating the suitability of a given implementation for your application.

**Since:** 1.5 **See Also:**[ReentrantReadWriteLock](http://docs.google.com/java/util/concurrent/locks/ReentrantReadWriteLock.html), [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html), [ReentrantLock](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html)

| **Method Summary** | |
| --- | --- |
| [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) | [**readLock**](http://docs.google.com/java/util/concurrent/locks/ReadWriteLock.html#readLock())()            Returns the lock used for reading. |
| [Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) | [**writeLock**](http://docs.google.com/java/util/concurrent/locks/ReadWriteLock.html#writeLock())()            Returns the lock used for writing. |

| **Method Detail** |
| --- |

### readLock

[Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) **readLock**()

Returns the lock used for reading.

**Returns:**the lock used for reading.

### writeLock

[Lock](http://docs.google.com/java/util/concurrent/locks/Lock.html) **writeLock**()

Returns the lock used for writing.

**Returns:**the lock used for writing.

| | [**Overview**](http://docs.google.com/overview-summary.html) | [**Package**](http://docs.google.com/package-summary.html) | **Class** | [**Use**](http://docs.google.com/class-use/ReadWriteLock.html) | [**Tree**](http://docs.google.com/package-tree.html) | [**Deprecated**](http://docs.google.com/deprecated-list.html) | [**Index**](http://docs.google.com/index-files/index-1.html) | [**Help**](http://docs.google.com/help-doc.html) | | --- | --- | --- | --- | --- | --- | --- | --- | | | ***Java™ Platform***  ***Standard Ed. 6*** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [**PREV CLASS**](http://docs.google.com/java/util/concurrent/locks/LockSupport.html)   [**NEXT CLASS**](http://docs.google.com/java/util/concurrent/locks/ReentrantLock.html) | [**FRAMES**](http://docs.google.com/index.html?java/util/concurrent/locks/ReadWriteLock.html)    [**NO FRAMES**](http://docs.google.com/ReadWriteLock.html)     [**All Classes**](http://docs.google.com/allclasses-noframe.html) |
| SUMMARY: NESTED | FIELD | CONSTR | [METHOD](#3znysh7) | DETAIL: FIELD | CONSTR | [METHOD](#2et92p0) |

[Submit a bug or feature](http://bugs.sun.com/services/bugreport/index.jsp)

For further API reference and developer documentation, see [Java SE Developer Documentation](http://docs.google.com/webnotes/devdocs-vs-specs.html). That documentation contains more detailed, developer-targeted descriptions, with conceptual overviews, definitions of terms, workarounds, and working code examples.

Copyright 2006 Sun Microsystems, Inc. All rights reserved. Use is subject to [license terms](http://docs.google.com/legal/license.html). Also see the [documentation redistribution policy](http://java.sun.com/docs/redist.html).