StampedLock

# StampedLock

## 继承关系

public class **StampedLock** extends Object implements **Serializable**

**since：1.8**

## 功能介绍

A capability-based lock with three modes for controlling read/write access. The state of a StampedLock consists of a version and mode. Lock acquisition methods return a stamp that represents and controls access with respect to a lock state; "try" versions of these methods may instead return the special value zero to represent failure to acquire access. Lock release and conversion methods require stamps as arguments, and fail if they do not match the state of the lock. The three modes are:

Writing. Method writeLock() possibly blocks waiting for exclusive access, returning a stamp that can be used in method unlockWrite(long) to release the lock. Untimed and timed versions of tryWriteLock are also provided. When the lock is held in write mode, no read locks may be obtained, and all optimistic read validations will fail.

Reading. Method readLock() possibly blocks waiting for non-exclusive access, returning a stamp that can be used in method unlockRead(long) to release the lock. Untimed and timed versions of tryReadLock are also provided.

Optimistic Reading. Method tryOptimisticRead() returns a non-zero stamp only if the lock is not currently held in write mode. Method validate(long) returns true if the lock has not been acquired in write mode since obtaining a given stamp. This mode can be thought of as an extremely weak version of a read-lock, that can be broken by a writer at any time. The use of optimistic mode for short read-only code segments often reduces contention and improves throughput. However, its use is inherently fragile. Optimistic read sections should only read fields and hold them in local variables for later use after validation. Fields read while in optimistic mode may be wildly inconsistent, so usage applies only when you are familiar enough with data representations to check consistency and/or repeatedly invoke method validate(). For example, such steps are typically required when first reading an object or array reference, and then accessing one of its fields, elements or methods.

This class also supports methods that conditionally provide conversions across the three modes. For example, method tryConvertToWriteLock(long) attempts to "upgrade" a mode, returning a valid write stamp if (1) already in writing mode (2) in reading mode and there are no other readers or (3) in optimistic mode and the lock is available. The forms of these methods are designed to help reduce some of the code bloat that otherwise occurs in retry-based designs.

StampedLocks are designed for use as internal utilities in the development of thread-safe components. Their use relies on knowledge of the internal properties of the data, objects, and methods they are protecting. They are not reentrant, so locked bodies should not call other unknown methods that may try to re-acquire locks (although you may pass a stamp to other methods that can use or convert it). The use of read lock modes relies on the associated code sections being side-effect-free. Unvalidated optimistic read sections cannot call methods that are not known to tolerate potential inconsistencies. Stamps use finite representations, and are not cryptographically secure (i.e., a valid stamp may be guessable). Stamp values may recycle after (no sooner than) one year of continuous operation. A stamp held without use or validation for longer than this period may fail to validate correctly. StampedLocks are serializable, but always deserialize into initial unlocked state, so they are not useful for remote locking.

The scheduling policy of StampedLock does not consistently prefer readers over writers or vice versa. All "try" methods are best-effort and do not necessarily conform to any scheduling or fairness policy. A zero return from any "try" method for acquiring or converting locks does not carry any information about the state of the lock; a subsequent invocation may succeed.

Because it supports coordinated usage across multiple lock modes, this class does not directly implement the Lock or ReadWriteLock interfaces. However, a StampedLock may be viewed asReadLock(), **asWriteLock**(), or **asReadWriteLock**() in applications requiring only the associated set of functionality.

## 构造方法

**StampedLock() ： Creates a new lock, initially in unlocked state.**

## 示例

class Point {

private double x, y;

private final StampedLock sl = new StampedLock();

void move(double deltaX, double deltaY) { // an exclusively locked method

long stamp = sl.writeLock();

try {

x += deltaX;

y += deltaY;

} finally {

sl.unlockWrite(stamp);

}

}

double distanceFromOrigin() { // A read-only method

long stamp = sl.tryOptimisticRead();

double currentX = x, currentY = y;

if (!sl.validate(stamp)) {

stamp = sl.readLock();

try {

currentX = x;

currentY = y;

} finally {

sl.unlockRead(stamp);

}

}

return Math.sqrt(currentX \* currentX + currentY \* currentY);

}

void moveIfAtOrigin(double newX, double newY) { // upgrade

// Could instead start with optimistic, not read mode

long stamp = sl.readLock();

try {

while (x == 0.0 && y == 0.0) {

long ws = sl.tryConvertToWriteLock(stamp);

if (ws != 0L) {

stamp = ws;

x = newX;

y = newY;

break;

}

else {

sl.unlockRead(stamp);

stamp = sl.writeLock();

}

}

} finally {

sl.unlock(stamp);

}

}

}