Evaluating the Pros & Cons of Java Readers-Writer Locks

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

• Recognize the intent of readers-writer locks
• Note a human known use of readers-writer locks
• Appreciate the pros & cons of readers-writer locks in general
Learning Objectives in this Part of the Lesson

- Recognize the intent of readers-writer locks
- Note a human known use of readers-writer locks
- Appreciate the pros & cons of readers-writer locks in general
- & also briefly outlines the pros & cons of Java’s ReentrantReadWriteLock & StampedLock

Class StampedLock

```java
java.lang.Object
java.util.concurrent.locks.StampedLock
```

All Implemented Interfaces:

- Serializable

```java
public class StampedLock
extends Object
implements Serializable
```

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

Class ReentrantReadWriteLock

```java
java.lang.Object
java.util.concurrent.locks.ReentrantReadWriteLock
```

All Implemented Interfaces:

- Serializable, ReadWriteLock

```java
public class ReentrantReadWriteLock
extends Object
implements ReadWriteLock, Serializable
```
Pros & Cons of Readers-Writer Locks
Pros

- Readers-writer locks *may* help improve performance
Pros & Cons of Readers-Writer Locks

Pros

- Readers-writer locks *may* help improve performance
- e.g., when resources are *read* from much more often than they are *written to*
Pros

• Readers-writer locks *may* help improve performance
  • e.g., when resources are *read* from much more often than they are *written to*
  • Especially on multi-core platforms that have a high level of inherent parallelism

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReadWriteLock.html
Cons

- Readers-writer locks can be problematic in practice
Pros & Cons of Readers-Writer Locks

Cons

- Readers-writer locks can be problematic in practice, e.g.
  - Can lead to starvation
  - Giving preference to either readers or writers can yield problems due to unfairness

See www.javaspecialists.eu/archive/Issue165.html
Pros & Cons of Readers-Writer Locks

Cons

- Readers-writer locks can be problematic in practice, e.g.
- Can lead to starvation
- Can be hard to program
- Due to features like lock upgrading & downgrading

```java
public class SimpleAtomicLong {
    ...
    public long incrementAndGet() {
        long value = 0;
        Lock lock = mRWLock.writeLock();
        lock.lock();
        try {
            mValue++;
            final Lock readLock =
                mRWLock.readLock();
            readLock.lock(); // Downgrade
            try {
                lock.unlock();
                value = mValue;
            } finally { lock = readLock; }
        } finally {
            lock.unlock();
        }
        return value;
    }
}
```

See word-bits.flurg.com/safely-downgrading-a-write-lock-with-readwritelock
Pros & Cons of Readers-Writer Locks

Cons

- Readers-writer locks can be problematic in practice, e.g.
  - Can lead to starvation
  - Can be hard to program
  - May be significantly slower than other synchronizers
  - Due to complexities of implementing the readers-writer protocol in software

Pros & Cons of Readers-Writer Locks

- Upcoming lessons evaluate the pros & cons of Java StampedLock & ReentrantReadWriteLock in more detail.

**Class ReentrantReadWriteLock**

```java
java.lang.Object
    java.util.concurrent.locks.ReentrantReadWriteLock
```

**All Implemented Interfaces:**
- Serializable, ReadWriteLock

```java
public class ReentrantReadWriteLock extends Object
    implements ReadWriteLock, Serializable
```

An implementation of ReadWriteLock supporting similar semantics to ReentrantLock.

This class has the following properties:

- **Acquisition order**
  
  This class does not impose a reader or writer preference ordering for lock access. However, it does support an optional fairness policy.

**Class StampedLock**

```java
java.lang.Object
    java.util.concurrent.locks.StampedLock
```

**All Implemented Interfaces:**
- Serializable

```java
public class StampedLock
```

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
Pros & Cons of Readers-Writer Locks

• Upcoming lessons evaluate the pros & cons of Java StampedLock & ReentrantReadWriteLock in more detail.

**Class ReentrantReadWriteLock**

```java
java.lang.Object
    java.util.concurrent.locks.ReentrantReadWriteLock

All Implemented Interfaces:
    Serializable, ReadWriteLock
```

```java
public class ReentrantReadWriteLock
extends Object
implements ReadWriteLock, Serializable
```

An implementation of `ReadWriteLock` supporting similar semantics to `ReentrantLock`.

This class has the following properties:

- **Acquisition order**
  
  This class does not impose a reader or writer preference ordering for lock access. However, it does support an optional fairness policy.

**Pros:** Portable to earlier versions of Java & relatively simple to program

**Cons:** Inefficient & don’t scale well
Pros & Cons of Readers-Writer Locks

- Upcoming lessons evaluate the pros & cons of Java StampedLock & ReentrantReadWriteLock in more detail.

**Pros:** More scalable & efficient

**Cons:** Complex & not portable to earlier versions of Java
End of Evaluating the Pros & Cons of Java Readers-Writer Locks