无锁

1. 无锁类的原理详解

1.1. CAS

CAS算法的过程是这样:它包含3个参数CAS(V,E,N)。V表示要更新的变量，E表示预期值，N表示新值。仅当V值等于E值时，才会将V的值设为N，如果V值和E值不同，则说明已经有其他线程做了更新，则当前线程什么都不做。最后，CAS返回当前V的真实值。CAS操作是抱着乐观的态度进行的，它总是认为自己可以成功完成操作。当多个线程同时使用CAS操作一个变量时，只有一个会胜出，并成功更新，其余均会失败。失败的线程不会被挂起，仅是被告知失败，并且允许再次尝试，当然也允许失败的线程放弃操作。基于这样的原理，CAS操作即时没有锁，也可以发现其他线程对当前线程的干扰，并进行恰当的处理。

1.2. CPU指令

```
cmpxchg
/*
accumulator = AL, AX, or EAX, depending on whether
a byte, word, or doubleword comparison is being performed
*/
if(accumulator == Destination) {
    ZF = 1;
    Destination = Source;
}
else {
    ZF = 0;
    accumulator = Destination;
}
```

2. 无锁类的使用

2.1. AtomicInteger

2.1.1. 概述

Number

2.1.2. 主要接口

public final int get()　　//取得当前值

public final void set(int newValue)　　//设置当前值

public final int getAndSet(int newValue)　　//设置新值，并返回旧值

```
public final boolean compareAndSet(int expect, int u)
  //如果当前值为expect，则设置为u
public final int getAndIncrement()          //当前值加1，返回旧值
public final int getAndDecrement()          //当前值减1，返回旧值
public final int getAndAdd(int delta)         //当前值增加delta，返回旧值
public final int incrementAndGet()          //当前值加1，返回新值
public final int decrementAndGet()          //当前值减1，返回新值
public final int addAndGet(int delta)         //当前值增加delta，返回新值
```

2.1.3. 主要接口的实现

2.2. Unsafe

2.2.1. 概述

非安全的操作，比如：
根据偏移量设置值
park()
底层的CAS操作
非公开API，在不同版本的JDK中，可能有较大差异

2.2.2. 主要接口

```
//获得给定对象偏移量上的int值
public native int getInt(Object o, long offset);
//设置给定对象偏移量上的int值
public native void putInt(Object o, long offset, int x);
//获得字段在对象中的偏移量
public native long objectFieldOffset(Field f);
//设置给定对象的int值，使用volatile语义
public native void putIntVolatile(Object o, long offset, int x);
//获得给定对象对象的int值，使用volatile语义
public native int    getIntVolatile(Object o, long offset);
//和putIntVolatile()一样，但是它要求被操作字段就是volatile类型的
public native void putOrderedInt(Object o, long offset, int x);
```

2.3. AtomicReference

2.3.1. 概述

对引用进行修改

是一个模板类，抽象化了数据类型

2.3.2. 主要接口

get()
set(V)
compareAndSet()
getAndSet(V)

2.4. AtomicStampedReference

2.4.1. 概述

ABA问题

2.4.2. 主要接口

//比较设置 参数依次为：期望值 写入新值 期望时间戳 新时间戳
public boolean compareAndSet(V expectedReference,V newReference,int expectedStamp,int newStamp)
//获得当前对象引用
public V getReference()
//获得当前时间戳
public int getStamp()
//设置当前对象引用和时间戳
public void set(V newReference, int newStamp)

2.5. AtomicIntegerArray

2.5.1. 概述

支持无锁的数组

2.5.2. 主要接口

//获得数组第i个下标的元素
public final int get(int i)
//获得数组的长度
public final int length()
//将数组第i个下标设置为newValue，并返回旧的值
public final int getAndSet(int i, int newValue)
//进行CAS操作，如果第i个下标的元素等于expect，则设置为update，设置成功返回true
public final boolean compareAndSet(int i, int expect, int update)
//将第i个下标的元素加1
public final int getAndIncrement(int i)
//将第i个下标的元素减1
public final int getAndDecrement(int i)
//将第i个下标的元素增加delta（delta可以是负数）
public final int getAndAdd(int i, int delta)

2.6. AtomicIntegerFieldUpdater

2.6.1. 概述

让普通变量也享受原子操作

2.6.2. 主要接口

AtomicIntegerFieldUpdater.newUpdater()
incrementAndGet()

2.6.3. 小说明

1.
Updater只能修改它可见范围内的变量。因为Updater使用反射得到这个变量。如果变量不可见，就会出错。比如如果score申明为private，就是不可行的。
2.
为了确保变量被正确的读取，它必须是volatile类型的。如果我们原有代码中未申明这个类型，那么简单得申明一下就行，这不会引起什么问题。

3. 由于CAS操作会通过对象实例中的偏移量直接进行赋值，因此，它不支持static字段（Unsafe.objectFieldOffset()不支持静态变量）。

3. 无锁算法详解

3.1. 无锁的Vector实现