JVM上弱内存模型案例分享

www.huawei.com

内存模型

强内存模型 —— x86

只会存在一种重排序的情况：storeload

弱内存模型 —— aarch64

四种类型的内存重排序都可能发生：loadload、loadstore、storeload、storestore

代码中如何保证实际运行时的内存访问顺序？

- 插入平台相关的memory barrier
- 在两次内存访问操作之间建立依赖

弱内存模型

CPU1

```
data->a1 = data1;
data->a2 = data2;
data->a3 = data3;
...
write_memory_barriar();
data->ready = true;
```

CPU2

```
ready = data->ready;
read_memory_barriar();

if (ready) {
    mydata1 = data->a1;
    ...
}
```

aarch64的memory barrier

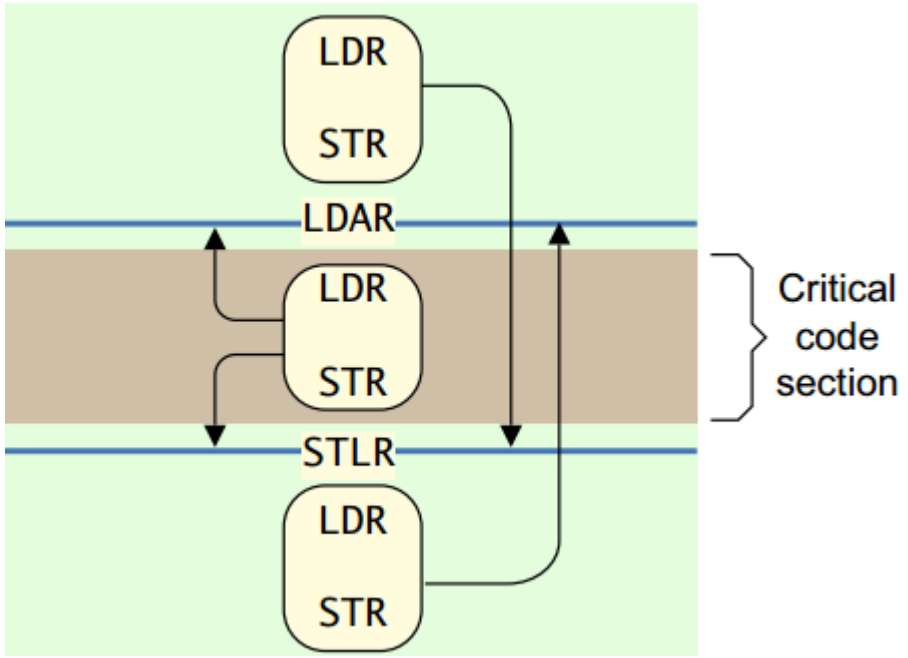
常用的几条memory barrier

指令	保证的访问顺序
DMB ISHLD	LoadLoad LoadStore
DMB ISHST	StoreStore
DMB ISH	LoadLoad LoadStore StoreStore StoreLoad
LDAR r0, [r1]	LoadLoad LoadStore
STLR r0, [r1]	StoreStore StoreLoad

ARM Architecture Reference Manual for ARMv8-A

aarch64的memory barrier

LDAR和STLR的限制图示



HUAWEI

aarch64的依赖规则

可以保证内存访问顺序的依赖

- 地址依赖：读指令返回的值，如果直接/间接参与到后续读写指令的内存地址计算中

```
char *a = “hello”;
char b = *(a + 1);
```

```
ldr r1, [r0]
ldr r2, [r1]
```

```
ldr r1, [r0]
and r1, r1, #0
ldr r2, [r3, r1]
```

无法保证内存访问顺序的依赖

- 条件依赖：读指令返回的值，如果用作条件标志，来决定后续分支走向

```
a = x;
if (a == 1)
    b = 2;
else
    c = 3;
```

```
ldr r1, [r0]
b r1
cmp r1, #55
ldr.ne r2,[r3]
```

aarch64上写法示例

写法1

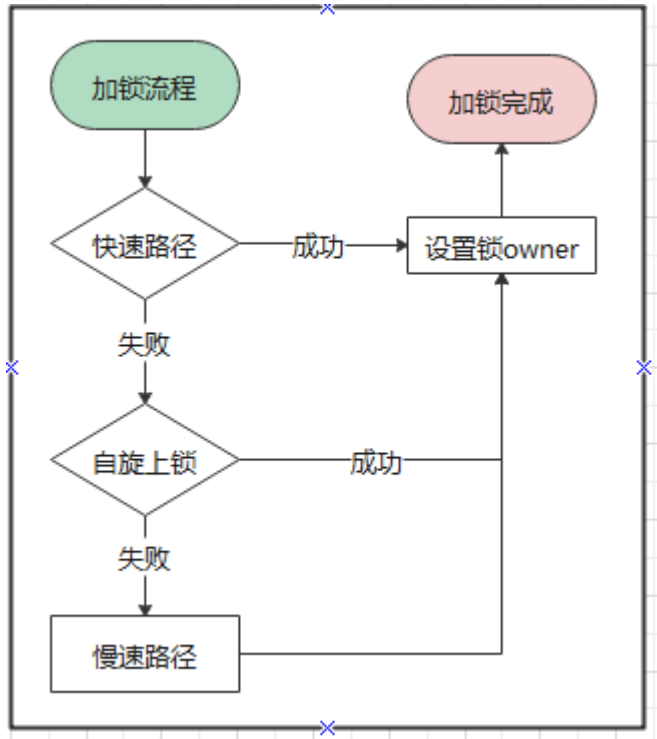
```
P1
str r5, [r1]
stlr r0, [r2]

P2
wait.
      ldar r12, [r2]
      cmp r12, #1
      b.ne <wait>
      ldr r5, [r1]
```

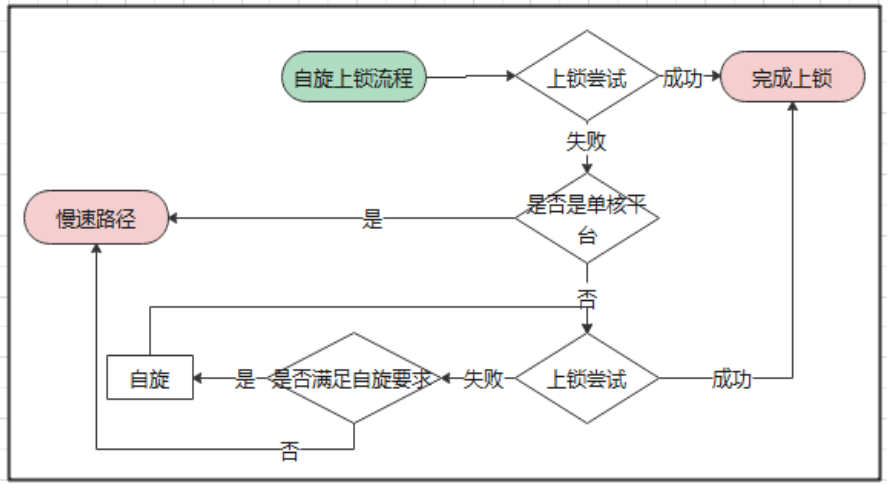
写法2

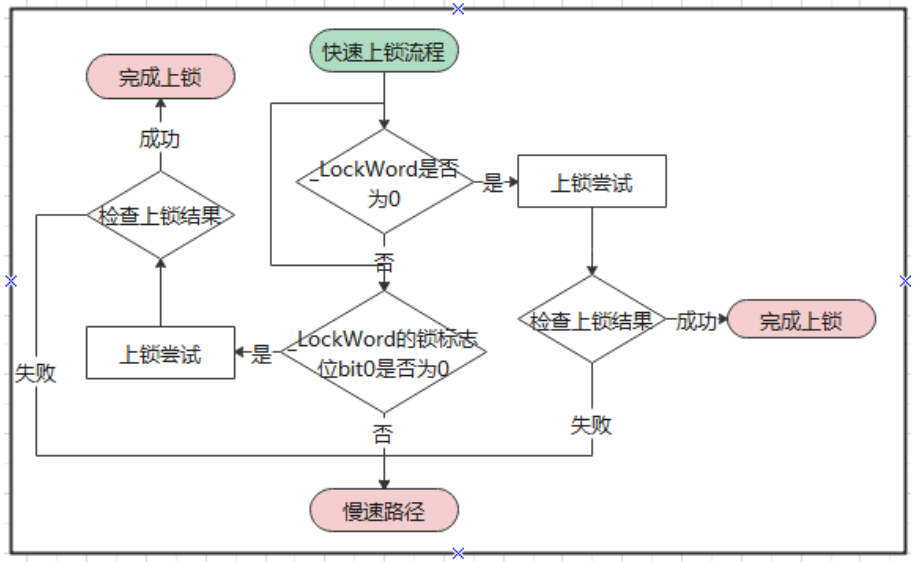
```
P1
str r5, [r1]
stlr r0, [r2]

P2
wait.
      ldr r12, [r2]
      cmp r12, #1
      b.ne <wait>
      and r12, r12, #0
      ldr r5, [r1, r12]
```

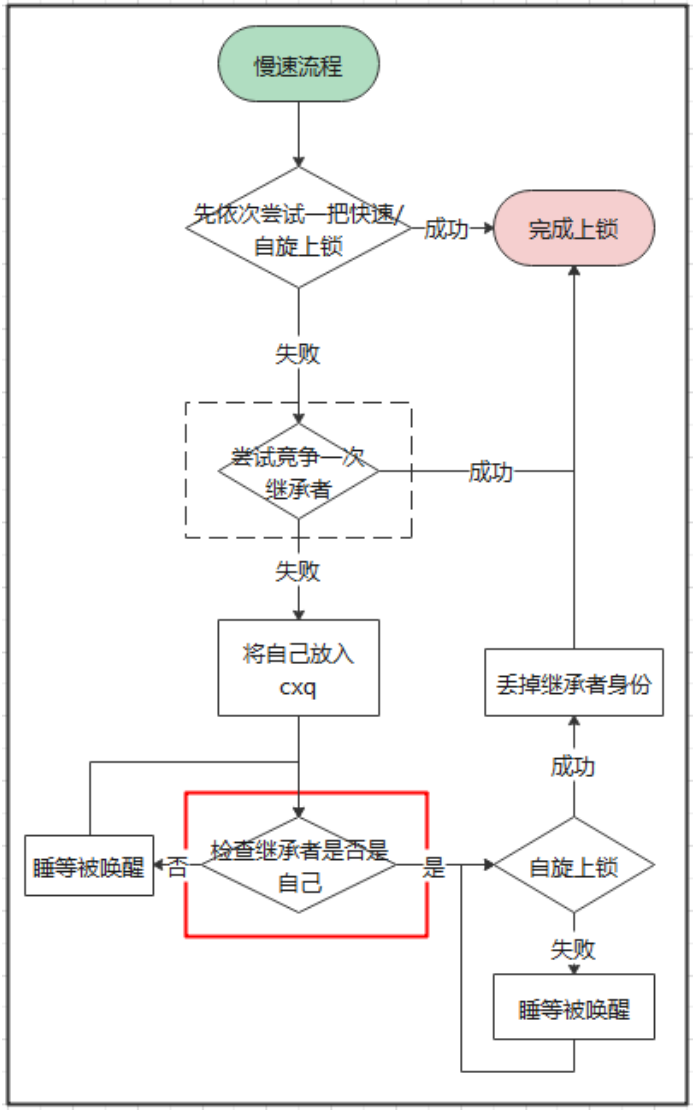
JVM案例：弱内存模型导致GC线程死循环

前置知识——JVM内部锁实现原理





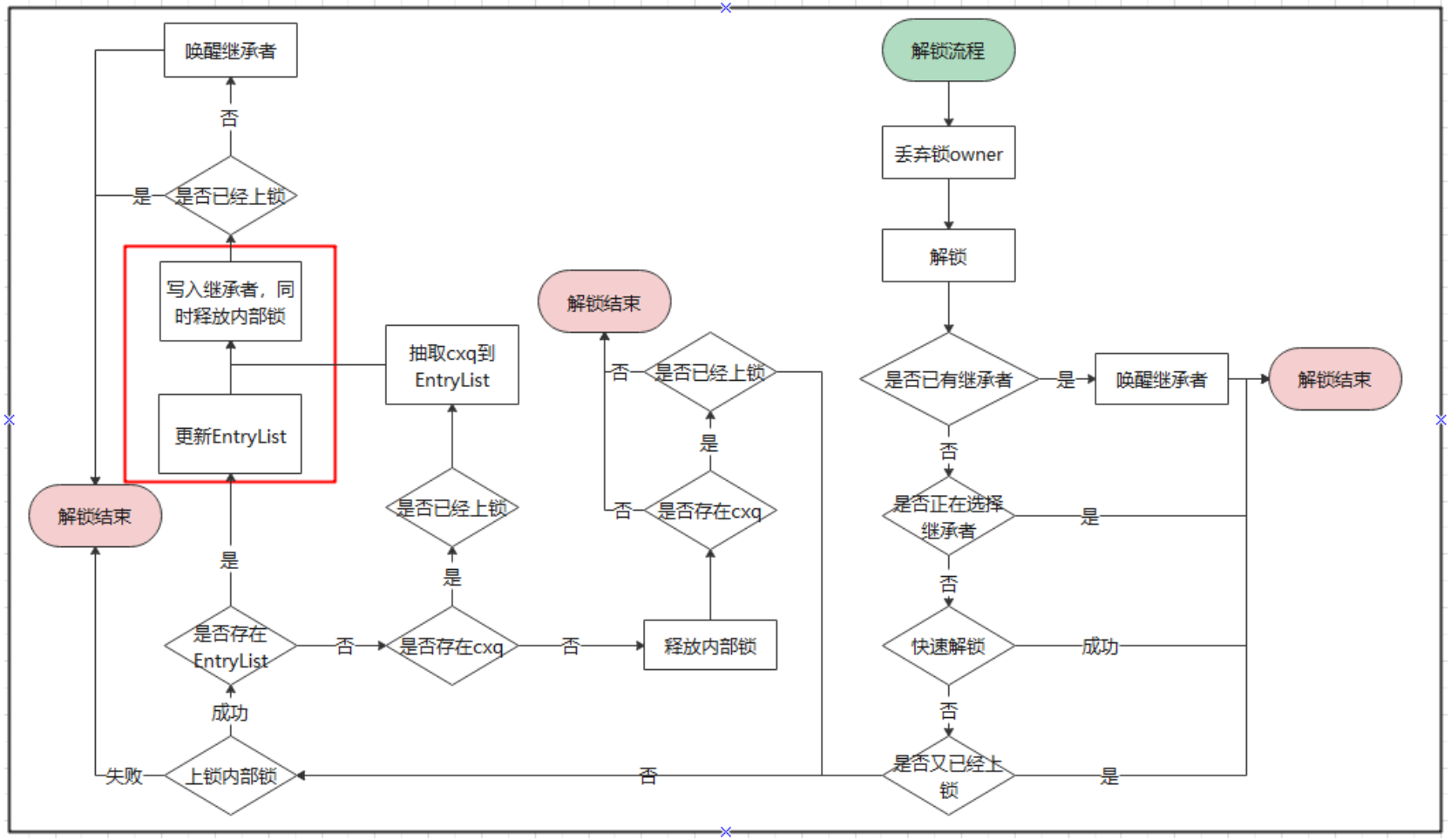




HUAWEI

JVM案例：弱内存模型导致GC线程死循环

前置知识——JVM内部锁实现原理



案例描述

部分ParNew线程在JVM内部Monitor上死锁

Monitor中的情况：

- cxq上积累了大量GC线程
- OnDeck记录的GC线程已经走了
- _owner为NULL

JVM案例：GC线程死循环

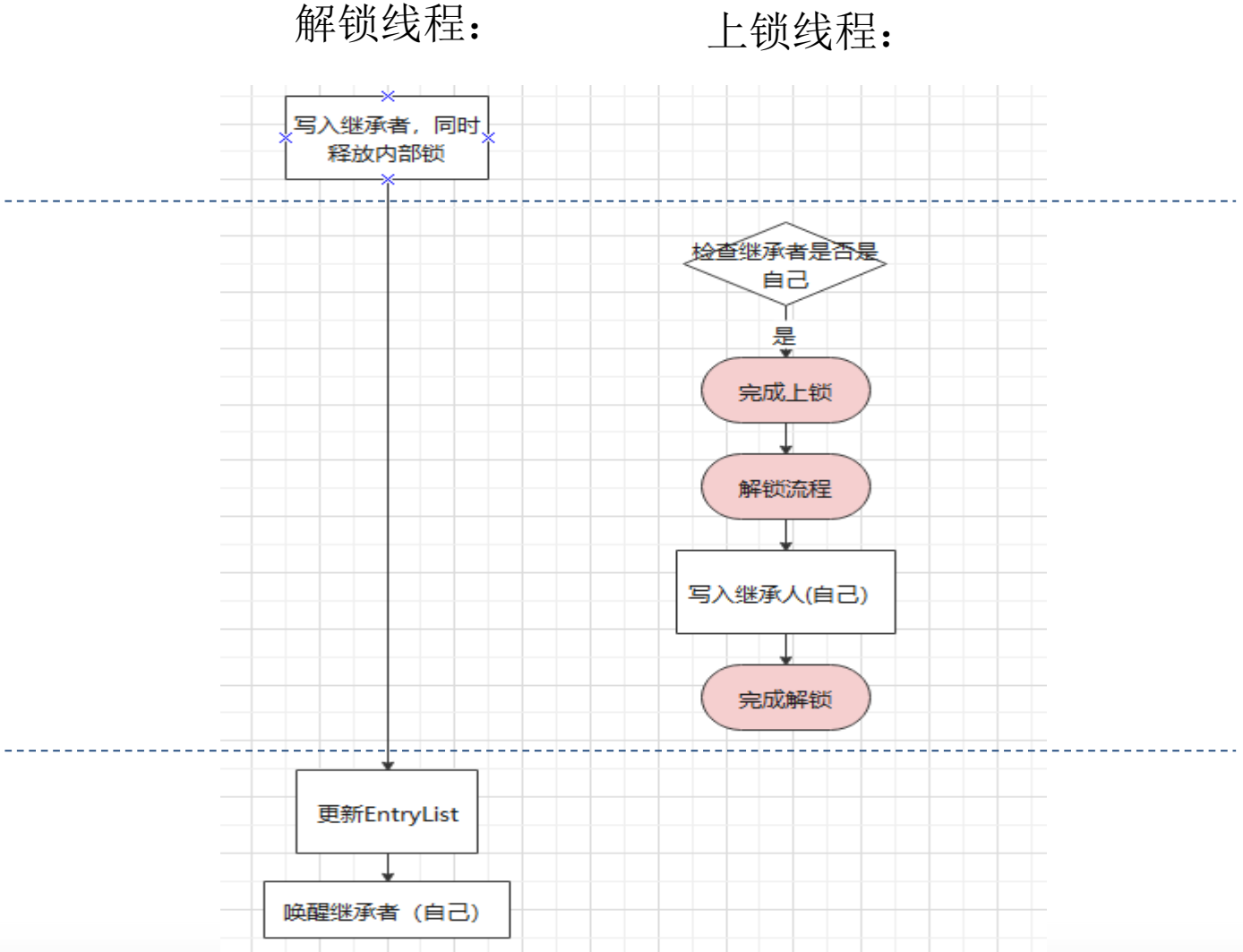
上锁线程：

```
    // At any given time there is at most one ondeck thread.
    // ondeck implies not resident on cxq and not resident on EntryList
-   // Only the OnDeck thread can try to acquire -- contended for -- the lock.
+   // Only the OnDeck thread can try to acquire -- contend for -- the lock.
    // CONSIDER: use Self->OnDeck instead of m->OnDeck.
    // Deschedule Self so that others may run.
-   while (_OnDeck != ESelf) {
+   while (OrderAccess::load_ptr_acquire(&_OnDeck) != ESelf) {
      ParkCommon (ESelf, 0) ;
    }
```

解锁线程：

```
    _EntryList = w->ListNext ;
    // as a diagnostic measure consider setting w->_ListNext = BAD
    assert (UNS( OnDeck) == _LBIT, "invariant") ;
-   _OnDeck = w ;           // pass OnDeck to w.
-                           // w will clear OnDeck once it acquires the outer lock
+
+   // Pass OnDeck role to w, ensuring that _EntryList has been set first.
+   // w will clear _OnDeck once it acquires the outer lock.
+   // Note that once we set _OnDeck that thread can acquire the mutex, proceed
+   // with its critical section and then enter this code to unlock the mutex. So
+   // you can have multiple threads active in IUnlock at the same time.
+   OrderAccess::release_store_ptr(&_OnDeck, w);
```

JVM案例：GC线程死循环



HUAWEI

内存模型

一些资源

http://www.puppetmastertrading.com/images/hwViewForSwHackers.pdf (偏硬件层面的内存模型解释)
https://en.cppreference.com/w/cpp/language/memory_model (C++11定义的内存模型规范)
http://gee.cs.oswego.edu/dl/jmm/cookbook.html (Java定义的内存模型规范)

Thank you

www.huawei.com