jdk源码&多线程&高并发-【阶段3、深入juc源码解析】

讲师简介

smart哥，互联网悍将，历经从传统软件公司到大型互联网公司的洗礼，入行即在中兴通讯等大型通信公司担任项目leader，后随着互联网的崛起，先后在美团支付等大型互联网公司担任架构师，公派旅美期间曾与并发包大神Doug Lea探讨java多线程等最底层的核心技术。对互联网架构底层技术有相当的研究和独特的见解，在多个领域有着丰富的实战经验。

一、Atomic相关的类

重点：unsafe，cas

1、unsafe类

- 获取Unsafe类的方式（3种方式）
  - 将应用达成jar，放置到bootstrap classload（启动类加载器）的搜索范围之内。
  - 通过构造方法来反射获取Unsafe
  - **通过unsafe属性反射获取Unsafe**
- unsafe方法分类
  - 对内存操作

    源码直接调用操作系统的函数

  - 对数组操作

  - 对对象进行操作
    - 重点：

```
putOrderedObject 方法
```

```
putObjectVolatile 方法
```

底层c++的源码都是一样的，但是jit会帮我们在汇编层面在putobjectVolatile方法上面加上storeload屏障

（**lock addl**）

-server
-XX:+UnlockDiagnosticVMOptions
-XX:+PrintAssembly
-XX:-Inline

- 揭秘putOrderedObject和**putObjectVolatile**方法（如果加上lock 指令前缀的？？）

  相同的代码生成不同的指令是如何做到的？？

  方法内联

1）vmsymbols.hpp

```
do_intrinsic(_putOrderedObject,          sun_misc_Unsafe,
 putOrderedObject_name, putOrderedObject_signature, F_RN) \
   do_name(     putOrderedObject_name,
"putOrderedObject")


do_intrinsic(_putObjectVolatile,         sun_misc_Unsafe,
 putObjectVolatile_name, putObject_signature,   F_RN)  \
```

根据不同的intrinsic id生成不同的指令集（差异就在指令集里面）

2）library_call.cpp

```
case vmIntrinsics::_putObjectVolatile:        return
inline_unsafe_access(!is_native_ptr,  is_store, T_OBJECT,
is_volatile);

 case vmIntrinsics::_putOrderedObject:        return
inline_unsafe_ordered_store(T_OBJECT);
```

比较inline_unsafe_access方法和inline_unsafe_ordered_store方法的区别

主要差别：

```
 if (is_volatile) {//如果是volatile操作
    if (!is_store) //非写即读
      insert_mem_bar(Op_MemBarAcquire); //读操作
    else
      insert_mem_bar(Op_MemBarVolatile); //写操作
  }

MemBarVolatile  在 x86_64.ad文件中定义了lock指令模板

instruct membar_volatile(rFlagsReg cr) %{
  match(MemBarVolatile);
  effect(KILL cr);
  ins_cost(400);

  format %{
    $$template
    if (os::is_MP()) {
      $$emit$$"lock addl [rsp + #0], 0\t! membar_volatile" //这里就是lock
指令前缀嵌入
    } else {
      $$emit$$"MEMBAR-volatile ! (empty encoding)"
    }
  %}
  ins_encode %{
```

```
    __ membar(Assembler::StoreLoad);
  %}
  ins_pipe(pipe_slow);
%}
```

- 对线程调度的操作
  - monitorenter， monitorExit， tryMonitorEnter
  - park，unpark
- 对Class的操作
  - defineClass
  - defineAnonymousClass
  - staticFieldBase
  - staticFieldOffset
- 对内存屏障的操作
  - loadFence
  - storeFence
  - fullFence
- 获取系统信息
  - addressSize
  - pageSize
- 对cas的操作

  全称就是compare and swap

二、Locks相关类（AQS+CAS）

三、配合并发的工具类（CountDownLatch，samephone）

四、线程安全的集合（map，list）

五、阻塞队列

六、Executors框架和Future异步框架