sun.misc.Unsafe详解和CAS底层实现

原创

lvbaolin123 于 2018-06-01 14:44:33 发布 3865 收藏 5

分类专栏： java 文章标签： Unsafe java cas实现 openJDK源码分析

本文链接：https://blog.csdn.net/lvbaolin123/article/details/80527598

最近在看jdk 1.8的源码，再看源码的过程中很多地方都用到了sun.misc.Unsafe, 然后对这个类做了一个详细的了解。

1. java 生态圈。 几乎每个使用 java开发的工具、软件基础设施、高性能开发库都在底层使用了 sun.misc.Unsafe 。
2. 这就是SUN未开源的sun.misc.Unsafe的类，该类功能很强大，涉及到类加载机制，其实例一般情况是获取不到的，源码中的设计是采用单例模式，不是系统加载初始化就会抛出SecurityException异常。
3. 查阅一些资料后发现，Unsafe类官方并不对外开放，因为Unsafe这个类提供了一些绕开JVM的更底层功能，基于它的实现可以提高效率。

   在jdk 1.9版本中对Unsafe提供了公开的API（之前受到过很多争议要不要去掉Unsafe类，stackoverflow帖子有很多说法是1.9版本会去掉这个类，移除 Unsafe 的一个主要论据是：使用它太容易让开发中犯错了。如果有完善的官方文档或许可以改善这一现状。）

   1. 下面详细分析这个类的实现和原理

1、Unsafe API的大部分方法都是native实现

分为下面几类：

Info：主要返回某些低级别的内存信息：

```
public native int addressSize();
public native int pageSize();
```

Objects：主要提供Object和它的域操纵方法

```
public native Object allocateInstance(Class<?> var1) throws InstantiationException;
public native long objectFieldOffset(Field var1);
```

Class：主要提供Class和它的静态域操纵方

```
public native long staticFieldOffset(Field var1);
public native Class<?> defineClass(String var1, byte[] var2, int var3, int var4, ClassLoader var5, Protectio
nDomain var6);
public native Class<?> defineAnonymousClass(Class<?> var1, byte[] var2, Object[] var3);
public native void ensureClassInitialized(Class<?> var1);
```

Arrays：数组操纵方

```
public native int arrayBaseOffset(Class<?> var1);
public native int arrayIndexScale(Class<?> var1);
```

Synchronization：主要提供低级别同步原语

```
/** @deprecated */
@Deprecated
public native void monitorEnter(Object var1);

/** @deprecated */
@Deprecated
public native void monitorExit(Object var1);

public final native boolean compareAndSwapInt(Object var1, long var2, int var4, int var5);

public native void putOrderedInt(Object var1, long var2, int var4);
```

Memory：直接内存访问方法（绕过JVM堆直接操纵本地内存）

```
public native long allocateMemory(long var1);
public native long reallocateMemory(long var1, long var3);
public native void setMemory(Object var1, long var2, long var4, byte var6);
public native void copyMemory(Object var1, long var2, Object var4, long var5, long var7);
```

2、Unsafe类实例的获取

- Unsafe类设计只提供给JVM信任的启动类加载器所使用，是一个典型的单例模式类

```
    private Unsafe() {
    }

    @CallerSensitive
    public static Unsafe getUnsafe() {
        Class var0 = Reflection.getCallerClass();
        if(!VM.isSystemDomainLoader(var0.getClassLoader())) {
            throw new SecurityException("Unsafe");
        } else {
            return theUnsafe;
        }
    }
```

- 可以通过反射技术暴力获取Unsafe对象，下面做一个cas算法的测试

```
package com.lv.study.unsafe;

import sun.misc.Unsafe;
import java.lang.reflect.Field;

/**
 * Created by lvbaolin on 2018/5/31.
 */
public class UnsafeCASTest {
    public static void main(String[] args) throws Exception {
        // 通过反射实例化Unsafe
        Field f = Unsafe.class.getDeclaredField("theUnsafe");
        f.setAccessible(true);
        Unsafe unsafe = (Unsafe) f.get(null);

        // 实例化Player
        Player player = (Player) unsafe.allocateInstance(Player.class);
        player.setAge(18);
        player.setName("li lei");
        for (Field field : Player.class.getDeclaredFields()) {
            System.out.println(field.getName() + ":对应的内存偏移地址" + unsafe.objectFieldOffset(field))
        }

        System.out.println("--------------------");
        // unsafe.compareAndSwapInt(arg0, arg1, arg2, arg3)
        // arg0, arg1, arg2, arg3 分别是目标对象实例，目标对象属性偏移量，当前预期值，要设的值

        int ageOffset = 12;
        // 修改内存偏移地址为12的值（age）,返回true,说明通过内存偏移地址修改age的值成功
        System.out.println(unsafe.compareAndSwapInt(player, ageOffset, 18, 20));
        System.out.println("age修改后的值：" + player.getAge());
        System.out.println("--------------------");

        // 修改内存偏移地址为12的值，但是修改后不保证立马能被其他的线程看到。
        unsafe.putOrderedInt(player, 12, 33);
        System.out.println("age修改后的值：" + player.getAge());
        System.out.println("--------------------");

        // 修改内存偏移地址为16的值，volatile修饰，修改能立马对其他线程可见
        unsafe.putObjectVolatile(player, 16, "han mei");
        System.out.println("name修改后的值：" + unsafe.getObjectVolatile(player, 16));
    }
}
```

```
class Player {
    private int age;
    private String name;
    private Player() {
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }

}
```

- 输出的结果是：

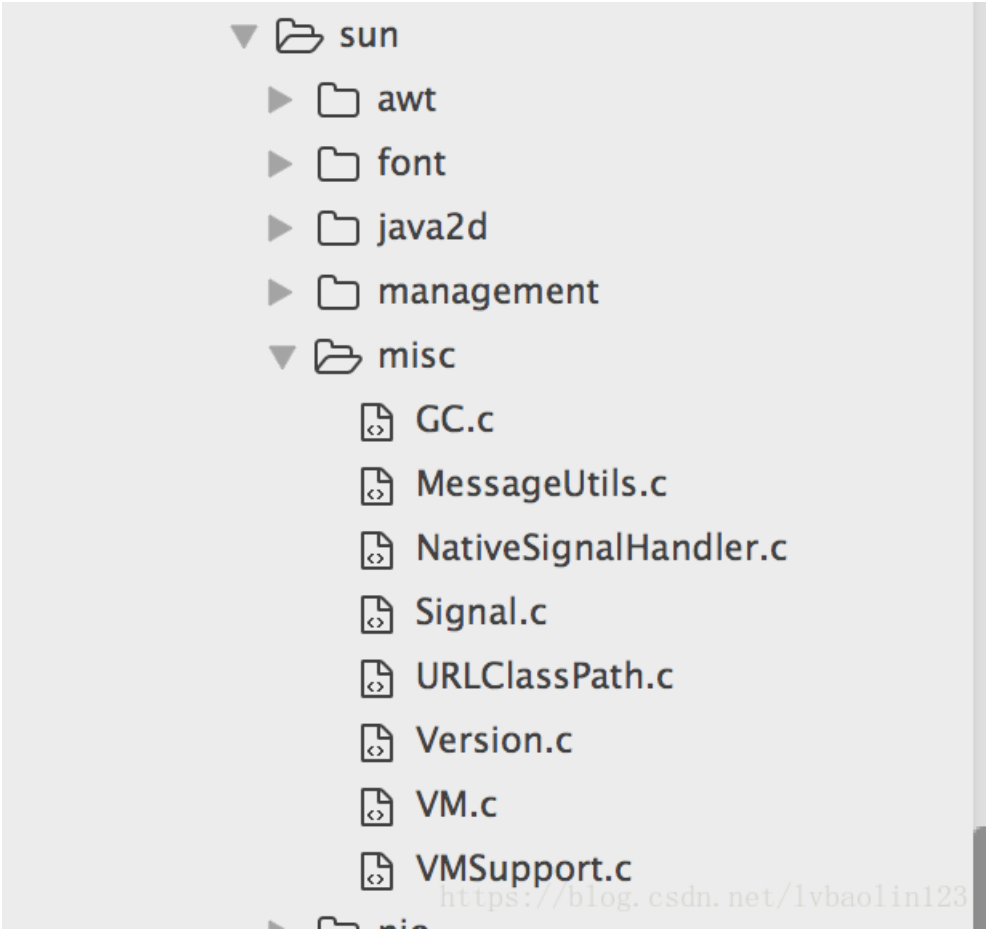
```
age:对应的内存偏移地址12
name:对应的内存偏移地址16
-------------------
true
age修改后的值：20
-------------------
age修改后的值：33
-------------------
name修改后的值：han mei
```

3、绕开JVM的更底层源码分析

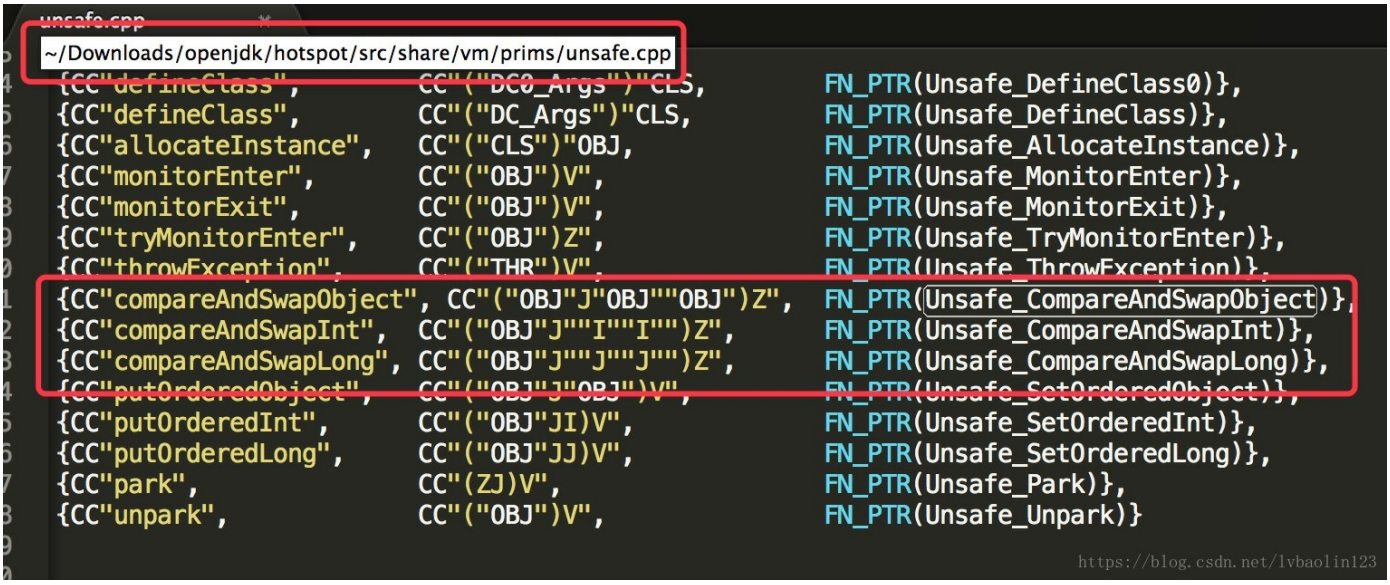
- 在Unsafe类中的方法

```
    /**
     * Compares the value of the integer field at the specified offset
     * in the supplied object with the given expected value, and updates
     * it if they match.  The operation of this method should be atomic,
     * thus providing an uninterruptible way of updating an integer field.
     * 在obj的offset位置比较integer field和期望的值，如果相同则更新。这个方法
     * 的操作应该是原子的，因此提供了一种不可中断的方式更新integer field。
     *
     * @param obj the object containing the field to modify.
     *            包含要修改field的对象
     * @param offset the offset of the integer field within <code>obj</code>.
     *               <code>obj</code>中整型field的偏移量
     * @param expect the expected value of the field.
     *               希望field中存在的值
     * @param update the new value of the field if it equals <code>expect</code>.
     *           如果期望值expect与field的当前值相同，设置filed的值为这个新值
     * @return true if the field was changed.
     *                             如果field的值被更改
     */
    public native boolean compareAndSwapInt(Object obj, long offset,int expect, int update);
```

- 在openJDK中在java/sun/misc目录下并没有Unsafe.c文件，说明这个实现不是通过jvm。



在/hotspot/src/share/vm/prims目录下可以找到Unsafe.cpp文件



分析CAS(compareAndSwapInt)源码实现

```
UNSAFE_ENTRY(jboolean, Unsafe_CompareAndSwapInt(JNIEnv *env, jobject unsafe, jobject obj, jlong offset,
  UnsafeWrapper("Unsafe_CompareAndSwapInt");
  oop p = JNIHandles::resolve(obj);
  //获取对象的变量的地址
  jint* addr = (jint *) index_oop_from_field_offset_long(p, offset);
  //调用Atomic操作
  //进入atomic.hpp,大意就是先去获取一次结果，如果结果和现在不同，就直接返回，因为有其他人修改了；否则会一直尝试去修改
  return (jint)(Atomic::cmpxchg(x, addr, e)) == e;
UNSAFE_END
```

参考文章：

http://mishadoff.com/blog/java-magic-part-4-sun-dot-misc-dot-unsafe/

https://blog.csdn.net/chenfenggang/article/details/76651273