Java中Unsafe类详解

java不能直接访问操作系统底层，而是通过本地方法来访问。Unsafe类提供了硬件级别的原子操作，主要提供了以下功能：

1、通过**Unsafe**类可以分配内存，可以释放内存；

类中提供的3个本地方法allocateMemory、reallocateMemory、freeMemory分别用于分配内存，扩充内存和释放内存，与C语言中的3个方法对应。

2、可以定位对象某字段的内存位置，也可以修改对象的字段值，即使它是私有的；

```
public native long allocateMemory(long l);
public native long reallocateMemory(long l, long l1);
public native void freeMemory(long l);
```

字段的定位：

JAVA中对象的字段的定位可能通过staticFieldOffset方法实现，该方法返回给定field的内存地址偏移量，这个值对于给定的filed是唯一的且是固定不变的。

getIntVolatile方法获取对象中offset偏移地址对应的整型field的值,支持volatile load语义。

getLong方法获取对象中offset偏移地址对应的long型field的值

数组元素定位：

Unsafe类中有很多以BASE_OFFSET结尾的常量，比如ARRAY_INT_BASE_OFFSET，ARRAY_BYTE_BASE_OFFSET等，这些常量值是通过arrayBaseOffset方法得到的。arrayBaseOffset方法是一个本地方法，可以获取数组第一个元素的偏移地址。Unsafe类中还有很多以INDEX_SCALE结尾的常量，比如 ARRAY_INT_INDEX_SCALE ， ARRAY_BYTE_INDEX_SCALE 等，这些常量值是通过arrayIndexScale方法得到的。arrayIndexScale方法也是一个本地方法，可以获取数组的转换因子，也就是数组中元素的增量地址。将arrayBaseOffset与arrayIndexScale配合使用，可以定位数组中每个元素在内存中的位置。

```
public final class Unsafe {
    public static final int ARRAY_INT_BASE_OFFSET;
    public static final int ARRAY_INT_INDEX_SCALE;

    public native long staticFieldOffset(Field field);
    public native int getIntVolatile(Object obj,long l);
    public native long getLong(Object obj, long l);
    public native int arrayBaseOffset(Class class1);
    public native int arrayIndexScale(Class class1);

    static
    {
        ARRAY_INT_BASE_OFFSET= theUnsafe.arrayBaseOffset([I);
        ARRAY_INT_INDEX_SCALE= theUnsafe.arrayIndexScale([I);
    }
}
```

3、挂起与恢复

将一个线程进行挂起是通过park方法实现的，调用 park后，线程将一直阻塞直到超时或者中断等条件出现。unpark可以终止一个挂起的线程，使其恢复正常。整个并发框架中对线程的挂起操作被封装在LockSupport类中，LockSupport类中有各种版本pack方法，但最终都调用了Unsafe.park()方法。

```
public class LockSupport {
    public static void unpark(Thread thread) {
        if (thread != null)
            unsafe.unpark(thread);
    }
```

```
    public static void park(Object blocker) {
        Thread t = Thread.currentThread();
        setBlocker(t, blocker);
        unsafe.park(false, 0L);
        setBlocker(t, null);
    }

    public static void parkNanos(Object blocker, long nanos) {
        if (nanos > 0) {
            Thread t = Thread.currentThread();
            setBlocker(t, blocker);
            unsafe.park(false, nanos);
            setBlocker(t, null);
        }
    }

    public static void parkUntil(Object blocker, long deadline) {
        Thread t = Thread.currentThread();
        setBlocker(t, blocker);
        unsafe.park(true, deadline);
        setBlocker(t, null);
    }

    public static void park() {
        unsafe.park(false, 0L);
    }

    public static void parkNanos(long nanos) {
        if (nanos > 0)
            unsafe.park(false, nanos);
    }

    public static void parkUntil(long deadline) {
        unsafe.park(true, deadline);
    }
}
```

4、CAS操作

是通过compareAndSwapXXX方法实现的

```
/**
* 比较obj的offset处内存位置中的值和期望的值，如果相同则更新。此更新是不可中断的。
*
* @param obj 需要更新的对象
* @param offset obj中整型field的偏移量
* @param expect 希望field中存在的值
* @param update 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 如果field的值被更改返回true
*/
public native boolean compareAndSwapInt(Object obj, long offset, int expect, int update);
```

CAS操作有3个操作数，内存值M，预期值E，新值U，如果M==E，则将内存值修改为B，否则啥都不做。

参考资料：

Unsafe.h

源码剖析之sun.misc.Unsafe

注：转自 http://blog.csdn.net/aesop_wubo/article/details/7537278

首先介绍一下什么是Compare And Swap(CAS)？简单的说就是比较并交换。

CAS 操作包含三个操作数 —— 内存位置（V）、预期原值（A）和新值(B)。如果内存位置的值与预期原值相匹配，那么处理器会自动将该位置值更新为新值。否则，处理器不做任何操作。无论哪种情况，它都会在 CAS 指令之前返回该位置的值。CAS 有效地说明了“我认为位置 V 应该包含值 A；如果包含该值，则将 B 放到这个位置；否则，不要更改该位置，只告诉我这个位置现在的值即可。” Java并发包(java.util.concurrent)中大量使用了CAS操作,涉及到并发的地方都调用了sun.misc.Unsafe类方法进行CAS操作。

在看一下volatile, Volatile修饰的成员变量在每次被线程访问时，都强迫从共享内存中重读该成员变量的值。而且，当成员变量发生变化时，强迫线程将变化值回写到共享内存。这样在任何时刻，两个不同的线程总是看到某个成员变量的值是相同的，更简单一点理解就是volatile修饰的变量值发生变化时对于另外的线程是可见的。

如何正确使用volatile可以参考下面这篇文章：

http://www.ibm.com/developerworks/cn/java/j-jtp06197.html Java 理论与实践：正确使用 Volatile 变量

下面来看看java中具体的CAS操作类sun.misc.Unsafe。Unsafe类提供了硬件级别的原子操作，Java无法直接访问到操作系统底层（如系统硬件等），为此Java使用native方法来扩展Java程序的功能。具体实现使用c++,详见文件sun.misc.natUnsafe.cc();sun.misc包的源代码可以在这里找到：

http://www.oschina.net/code/explore/gcc-4.5.2/libjava/sun/misc

```
//下面是sun.misc.Unsafe.java类源码
package sun.misc;
import java.lang.reflect.Field;
/***
 * This class should provide access to low-level operations and its
 * use should be limited to trusted code.  Fields can be accessed using
 * memory addresses, with undefined behaviour occurring if invalid memory
 * addresses are given.
 * 这个类提供了一个更底层的操作并且应该在受信任的代码中使用。可以通过内存地址
 * 存取fields,如果给出的内存地址是无效的那么会有一个不确定的运行表现。
 *
 * @author Tom Tromey (tromey@redhat.com)
 * @author Andrew John Hughes (gnu_andrew@member.fsf.org)
 */
public class Unsafe
{
  // Singleton class.
  private static Unsafe unsafe = new Unsafe();
  /***
   * Private default constructor to prevent creation of an arbitrary
   * number of instances.
   * 使用私有默认构造器防止创建多个实例
   */
  private Unsafe()
  {
  }
  /***
   * Retrieve the singleton instance of <code>Unsafe</code>.  The calling
   * method should guard this instance from untrusted code, as it provides
   * access to low-level operations such as direct memory access.
   * 获取<code>Unsafe</code>的单例,这个方法调用应该防止在不可信的代码中实例,
   * 因为unsafe类提供了一个低级别的操作，例如直接内存存取。
   *
   * @throws SecurityException if a security manager exists and prevents
   *                           access to the system properties.
   *                          如果安全管理器不存在或者禁止访问系统属性
   */
  public static Unsafe getUnsafe()
  {
    SecurityManager sm = System.getSecurityManager();
    if (sm != null)
      sm.checkPropertiesAccess();
    return unsafe;
  }

  /***
   * Returns the memory address offset of the given static field.
   * The offset is merely used as a means to access a particular field
   * in the other methods of this class.  The value is unique to the given
   * field and the same value should be returned on each subsequent call.
   * 返回指定静态field的内存地址偏移量,在这个类的其他方法中这个值只是被用作一个访问
   * 特定field的一个方式。这个值对于 给定的field是唯一的，并且后续对该方法的调用都应该
   * 返回相同的值。
   *
   * @param field the field whose offset should be returned.
   *              需要返回偏移量的field
   * @return the offset of the given field.
   *         指定field的偏移量
   */
  public native long objectFieldOffset(Field field);
  /***
   * 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);
    /**
     * Compares the value of the long 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 a long field.
     * 在obj的offset位置比较long 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long 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 long field within <code>obj</code>.
     *               <code>obj</code>中long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 compareAndSwapLong(Object obj,long offset,
                                             long expect, long update);
    /**
     * Compares the value of the object 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 object field.
     * 在obj的offset位置比较object 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object 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 object field within <code>obj</code>.
     *         <code>obj</code>中object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 compareAndSwapObject(Object obj,long offset,
                                               Object expect, Object update);
    /**
     * Sets the value of the integer field at the specified offset in the
     * supplied object to the given value.  This is an ordered or lazy
     * version of <code>putIntVolatile(Object,long,int)</code>, which
     * doesn't guarantee the immediate visibility of the change to other
     * threads.  It is only really useful where the integer field is
     * <code>volatile</code>, and is thus expected to change unexpectedly.
     * 设置obj对象中offset偏移地址对应的整型field的值为指定值。这是一个有序或者
     * 有延迟的<code>putIntVolatile</cdoe>方法，并且不保证值的改变被其他线程立
     * 即看到。只有在field被<code>volatile</code>修饰并且期望被意外修改的时候
     * 使用才有用。
     *
     * @param obj the object containing the field to modify.
     *    包含需要修改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 value the new value of the field.
     *      field将被设置的新值
     * @see #putIntVolatile(Object,long,int)
     */
    public native void putOrderedInt(Object obj,long offset, int value);
    /**
     * Sets the value of the long field at the specified offset in the
     * supplied object to the given value.  This is an ordered or lazy
     * version of <code>putLongVolatile(Object,long,long)</code>, which
```

```
     * doesn't guarantee the immediate visibility of the change to other
     * threads.  It is only really useful where the long field is
     * <code>volatile</code>, and is thus expected to change unexpectedly.
     * 设置obj对象中offset偏移地址对应的long型field的值为指定值。这是一个有序或者
     * 有延迟的<code>putLongVolatile</cdoe>方法，并且不保证值的改变被其他线程立
     * 即看到。只有在field被<code>volatile</code>修饰并且期望被意外修改的时候
     * 使用才有用。
     *
     * @param obj the object containing the field to modify.
     *    包含需要修改field的对象
     * @param offset the offset of the long field within <code>obj</code>.
     *       <code>obj</code>中long型field的偏移量
     * @param value the new value of the field.
     *      field将被设置的新值
     * @see #putLongVolatile(Object,long,long)
     */
    public native void putOrderedLong(Object obj,long offset, long value);
    /**
     * Sets the value of the object field at the specified offset in the
     * supplied object to the given value.  This is an ordered or lazy
     * version of <code>putObjectVolatile(Object,long,Object)</code>, which
     * doesn't guarantee the immediate visibility of the change to other
     * threads.  It is only really useful where the object field is
     * <code>volatile</code>, and is thus expected to change unexpectedly.
     * 设置obj对象中offset偏移地址对应的object型field的值为指定值。这是一个有序或者
     * 有延迟的<code>putObjectVolatile</cdoe>方法，并且不保证值的改变被其他线程立
     * 即看到。只有在field被<code>volatile</code>修饰并且期望被意外修改的时候
     * 使用才有用。
     *
     * @param obj the object containing the field to modify.
     *    包含需要修改field的对象
     * @param offset the offset of the object field within <code>obj</code>.
     *       <code>obj</code>中long型field的偏移量
     * @param value the new value of the field.
     *      field将被设置的新值
     */
    public native void putOrderedObject(Object obj,long offset, Object value);
    /**
     * Sets the value of the integer field at the specified offset in the
     * supplied object to the given value, with volatile store semantics.
     * 设置obj对象中offset偏移地址对应的整型field的值为指定值。支持volatile store语义
     *
     * @param obj the object containing the field to modify.
     *    包含需要修改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 value the new value of the field.
     *       field将被设置的新值
     */
    public native void putIntVolatile(Object obj,long offset, int value);
    /**
     * Retrieves the value of the integer field at the specified offset in the
     * supplied object with volatile load semantics.
     * 获取obj对象中offset偏移地址对应的整型field的值,支持volatile load语义。
     *
     * @param obj the object containing the field to read.
     *    包含需要去读取的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
     */
    public native int getIntVolatile(Object obj,long offset);
    /**
     * Sets the value of the long field at the specified offset in the
     * supplied object to the given value, with volatile store semantics.
     * 设置obj对象中offset偏移地址对应的long型field的值为指定值。支持volatile store语义
     *
     * @param obj the object containing the field to modify.
     *            包含需要修改field的对象
     * @param offset the offset of the long field within <code>obj</code>.
     *               <code>obj</code>中long型field的偏移量
     * @param value the new value of the field.
     *              field将被设置的新值
     * @see #putLong(Object,long,long)
     */
    public native void putLongVolatile(Object obj,long offset, long value);
    /**
     * Sets the value of the long field at the specified offset in the
     * supplied object to the given value.
     * 设置obj对象中offset偏移地址对应的long型field的值为指定值。
     *
```

```
     * @param obj the object containing the field to modify.
     *     包含需要修改field的对象
     * @param offset the offset of the long field within <code>obj</code>.
     *     <code>obj</code>中long型field的偏移量
     * @param value the new value of the field.
     *     field将被设置的新值
     * @see #putLongVolatile(Object,long,long)
     */
    public native void putLong(Object obj, long offset, long value);
    /**
     * Retrieves the value of the long field at the specified offset in the
     * supplied object with volatile load semantics.
     * 获取obj对象中offset偏移地址对应的long型field的值,支持volatile load语义。
     *
     * @param obj the object containing the field to read.
     *    包含需要去读取的field的对象
     * @param offset the offset of the long field within <code>obj</code>.
     *       <code>obj</code>中long型field的偏移量
     * @see #getLong(Object,long)
     */
    public native long getLongVolatile(Object obj,long offset);
    /**
     * Retrieves the value of the long field at the specified offset in the
     * supplied object.
     * 获取obj对象中offset偏移地址对应的long型field的值
     *
     * @param obj the object containing the field to read.
     *    包含需要去读取的field的对象
     * @param offset the offset of the long field within <code>obj</code>.
     *       <code>obj</code>中long型field的偏移量
     * @see #getLongVolatile(Object,long)
     */
    public native long getLong(Object obj, long offset);
    /**
     * Sets the value of the object field at the specified offset in the
     * supplied object to the given value, with volatile store semantics.
     * 设置obj对象中offset偏移地址对应的object型field的值为指定值。支持volatile store语义
     *
     * @param obj the object containing the field to modify.
     *    包含需要修改field的对象
     * @param offset the offset of the object field within <code>obj</code>.
     *     <code>obj</code>中object型field的偏移量
     * @param value the new value of the field.
     *       field将被设置的新值
     * @see #putObject(Object,long,Object)
     */
    public native void putObjectVolatile(Object obj,long offset, Object value);
    /**
     * Sets the value of the object field at the specified offset in the
     * supplied object to the given value.
     * 设置obj对象中offset偏移地址对应的object型field的值为指定值。
     *
     * @param obj the object containing the field to modify.
     *    包含需要修改field的对象
     * @param offset the offset of the object field within <code>obj</code>.
     *     <code>obj</code>中object型field的偏移量
     * @param value the new value of the field.
     *       field将被设置的新值
     * @see #putObjectVolatile(Object,long,Object)
     */
    public native void putObject(Object obj, long offset, Object value);
    /**
     * Retrieves the value of the object field at the specified offset in the
     * supplied object with volatile load semantics.
     * 获取obj对象中offset偏移地址对应的object型field的值,支持volatile load语义。
     *
     * @param obj the object containing the field to read.
     *    包含需要去读取的field的对象
     * @param offset the offset of the object field within <code>obj</code>.
     *       <code>obj</code>中object型field的偏移量
     */
    public native Object getObjectVolatile(Object obj,long offset);
    /**
     * Returns the offset of the first element for a given array class.
     * To access elements of the array class, this value may be used along with
     * with that returned by
     * <a href="#arrayIndexScale"><code>arrayIndexScale</code></a>,
     * if non-zero.
     * 获取给定数组中第一个元素的偏移地址。
     * 为了存取数组中的元素，这个偏移地址与<a href="#arrayIndexScale"><code>arrayIndexScale
```

```
     * </code></a>方法的非0返回值一起被使用。
     * @param arrayClass the class for which the first element's address should
     *                   be obtained.
     *                  第一个元素地址被获取的class
     * @return the offset of the first element of the array class.
     *    数组第一个元素 的偏移地址
     * @see arrayIndexScale(Class)
     */
    public native int arrayBaseOffset(Class arrayClass);
    /***
     * Returns the scale factor used for addressing elements of the supplied
     * array class.  Where a suitable scale factor can not be returned (e.g.
     * for primitive types), zero should be returned.  The returned value
     * can be used with
     * <a href="#arrayBaseOffset"><code>arrayBaseOffset</code></a>
     * to access elements of the class.
     * 获取用户给定数组寻址的换算因子.一个合适的换算因子不能返回的时候(例如：基本类型),
     * 返回0.这个返回值能够与<a href="#arrayBaseOffset"><code>arrayBaseOffset</code>
     * </a>一起使用去存取这个数组class中的元素
     *
     * @param arrayClass the class whose scale factor should be returned.
     * @return the scale factor, or zero if not supported for this array class.
     */
    public native int arrayIndexScale(Class arrayClass);

    /***
     * Releases the block on a thread created by
     * <a href="#park"><code>park</code></a>.  This method can also be used
     * to terminate a blockage caused by a prior call to <code>park</code>.
     * This operation is unsafe, as the thread must be guaranteed to be
     * live.  This is true of Java, but not native code.
     * 释放被<a href="#park"><code>park</code></a>创建的在一个线程上的阻塞.这个
     * 方法也可以被使用来终止一个先前调用<code>park</code>导致的阻塞.
     * 这个操作操作时不安全的,因此线程必须保证是活的.这是java代码不是native代码。
     * @param thread the thread to unblock.
     *           要解除阻塞的线程
     */
    public native void unpark(Thread thread);
    /***
     * Blocks the thread until a matching
     * <a href="#unpark"><code>unpark</code></a> occurs, the thread is
     * interrupted or the optional timeout expires.  If an <code>unpark</code>
     * call has already occurred, this also counts.  A timeout value of zero
     * is defined as no timeout.  When <code>isAbsolute</code> is
     * <code>true</code>, the timeout is in milliseconds relative to the
     * epoch.  Otherwise, the value is the number of nanoseconds which must
     * occur before timeout.  This call may also return spuriously (i.e.
     * for no apparent reason).
     * 阻塞一个线程直到<a href="#unpark"><code>unpark</code></a>出现、线程
     * 被中断或者timeout时间到期。如果一个<code>unpark</code>调用已经出现了，
     * 这里只计数。timeout为0表示永不过期.当<code>isAbsolute</code>为true时，
     * timeout是相对于新纪元之后的毫秒。否则这个值就是超时前的纳秒数。这个方法执行时
     * 也可能不合理地返回(没有具体原因)
     *
     * @param isAbsolute true if the timeout is specified in milliseconds from
     *                   the epoch.
     *                  如果为true timeout的值是一个相对于新纪元之后的毫秒数
     * @param time either the number of nanoseconds to wait, or a time in
     *             milliseconds from the epoch to wait for.
     *             可以是一个要等待的纳秒数，或者是一个相对于新纪元之后的毫秒数直到
     *             到达这个时间点
     */
    public native void park(boolean isAbsolute, long time);
}
```

注：转自 http://blog.csdn.net/zgmzyr/article/details/8902683

下面这个例子演示了简单的修改一个byte[]的数据。

这个例子在eclipse里不能直接编译，要到项目的属性，Java Compiler，Errors/Warnings中Forbidden reference(access rules)中设置为warning。

另外，因为sun.misc.Unsafe包不能直接使用，所有代码里用反射的技巧得到了一个Unsafe的实例。

```
import java.lang.reflect.Field;
import java.util.Arrays;
import sun.misc.Unsafe;
```

```
public class Test {
    private static int byteArrayBaseOffset;

    public static void main(String[] args) throws SecurityException,
            NoSuchFieldException, IllegalArgumentException,
            IllegalAccessException {
        Field theUnsafe = Unsafe.class.getDeclaredField("theUnsafe");
        theUnsafe.setAccessible(true);
        Unsafe UNSAFE = (Unsafe) theUnsafe.get(null);
        System.out.println(UNSAFE);

        byte[] data = new byte[10];
        System.out.println(Arrays.toString(data));
        byteArrayBaseOffset = UNSAFE.arrayBaseOffset(byte[].class);

        System.out.println(byteArrayBaseOffset);
        UNSAFE.putByte(data, byteArrayBaseOffset, (byte) 1);
        UNSAFE.putByte(data, byteArrayBaseOffset + 5, (byte) 5);
        System.out.println(Arrays.toString(data));
    }
}
```

运行结果：

```
sun.misc.Unsafe@6af62373
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
24
[1, 0, 0, 0, 0, 5, 0, 0, 0, 0]
```

注：转自 http://blog.csdn.net/hengyunabc/article/details/7657934

以下为Unsafe类涉及到的源码：

- natUnsafe.cc
- Service.h
- ServiceConfigurationError.h
- Unsafe.h
- Unsafe.java

```
/** Unsafe.java - Unsafe operations needed for concurrency
   Copyright (C) 2006 Free Software Foundation

This file is part of GNU Classpath.

GNU Classpath is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2, or (at your option)
any later version.

GNU Classpath is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
General Public License for more details.

You should have received a copy of the GNU General Public License
along with GNU Classpath; see the file COPYING.  If not, write to the
Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
02110-1301 USA.

Linking this library statically or dynamically with other modules is
making a combined work based on this library.  Thus, the terms and
conditions of the GNU General Public License cover the whole
combination.

As a special exception, the copyright holders of this library give you
permission to link this library with independent modules to produce an
executable, regardless of the license terms of these independent
modules, and to copy and distribute the resulting executable under
terms of your choice, provided that you also meet, for each linked
independent module, the terms and conditions of the license of that
module.  An independent module is a module which is not derived from
or based on this library.  If you modify this library, you may extend
this exception to your version of the library, but you are not
obligated to do so.  If you do not wish to do so, delete this
exception statement from your version.*/
```

```
package sun.misc;

import java.lang.reflect.Field;

/***
 * This class should provide access to low-level operations and its
 * use should be limited to trusted code.  Fields can be accessed using
 * memory addresses, with undefined behaviour occurring if invalid memory
 * addresses are given.
 *
 * @author Tom Tromey (tromey@redhat.com)
 * @author Andrew John Hughes (gnu_andrew@member.fsf.org)
 */
public class Unsafe
{
  // Singleton class.
  private static Unsafe unsafe = new Unsafe();

  /***
   * Private default constructor to prevent creation of an arbitrary
   * number of instances.
   */
  private Unsafe()
  {
  }

  /***
   * Retrieve the singleton instance of <code>Unsafe</code>.  The calling
   * method should guard this instance from untrusted code, as it provides
   * access to low-level operations such as direct memory access.
   *
   * @throws SecurityException if a security manager exists and prevents
   *                           access to the system properties.
   */
  public static Unsafe getUnsafe()
  {
    SecurityManager sm = System.getSecurityManager();
    if (sm != null)
      sm.checkPropertiesAccess();
    return unsafe;
  }

  /***
   * Returns the memory address offset of the given static field.
   * The offset is merely used as a means to access a particular field
   * in the other methods of this class.  The value is unique to the given
   * field and the same value should be returned on each subsequent call.
   *
   * @param field the field whose offset should be returned.
   * @return the offset of the given field.
   */
  public native long objectFieldOffset(Field field);

  /***
   * 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.
   *
   * @param obj the object containing the field to modify.
   * @param offset the offset of the integer field within <code>obj</code>.
   * @param expect the expected value of the field.
   * @param update the new value of the field if it equals <code>expect</code>.
   * @return true if the field was changed.
   */
  public native boolean compareAndSwapInt(Object obj,long offset,
                                          int expect, int update);

  /***
   * Compares the value of the long 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 a long field.
   *
   * @param obj the object containing the field to modify.
   * @param offset the offset of the long field within <code>obj</code>.
   * @param expect the expected value of the field.
   * @param update the new value of the field if it equals <code>expect</code>.
   * @return true if the field was changed.
   */
```

```
public native boolean compareAndSwapLong(Object obj,long offset,
                                         long expect, long update);

/***
 * Compares the value of the object 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 object field.
 *
 * @param obj the object containing the field to modify.
 * @param offset the offset of the object field within <code>obj</code>.
 * @param expect the expected value of the field.
 * @param update the new value of the field if it equals <code>expect</code>.
 * @return true if the field was changed.
 */
public native boolean compareAndSwapObject(Object obj,long offset,
                                           Object expect, Object update);

/***
 * Sets the value of the integer field at the specified offset in the
 * supplied object to the given value.  This is an ordered or lazy
 * version of <code>putIntVolatile(Object,long,int)</code>, which
 * doesn't guarantee the immediate visibility of the change to other
 * threads.  It is only really useful where the integer field is
 * <code>volatile</code>, and is thus expected to change unexpectedly.
 *
 * @param obj the object containing the field to modify.
 * @param offset the offset of the integer field within <code>obj</code>.
 * @param value the new value of the field.
 * @see #putIntVolatile(Object,long,int)
 */
public native void putOrderedInt(Object obj,long offset, int value);

/***
 * Sets the value of the long field at the specified offset in the
 * supplied object to the given value.  This is an ordered or lazy
 * version of <code>putLongVolatile(Object,long,long)</code>, which
 * doesn't guarantee the immediate visibility of the change to other
 * threads.  It is only really useful where the long field is
 * <code>volatile</code>, and is thus expected to change unexpectedly.
 *
 * @param obj the object containing the field to modify.
 * @param offset the offset of the long field within <code>obj</code>.
 * @param value the new value of the field.
 * @see #putLongVolatile(Object,long,long)
 */
public native void putOrderedLong(Object obj,long offset, long value);

/***
 * Sets the value of the object field at the specified offset in the
 * supplied object to the given value.  This is an ordered or lazy
 * version of <code>putObjectVolatile(Object,long,Object)</code>, which
 * doesn't guarantee the immediate visibility of the change to other
 * threads.  It is only really useful where the object field is
 * <code>volatile</code>, and is thus expected to change unexpectedly.
 *
 * @param obj the object containing the field to modify.
 * @param offset the offset of the object field within <code>obj</code>.
 * @param value the new value of the field.
 */
public native void putOrderedObject(Object obj,long offset, Object value);

/***
 * Sets the value of the integer field at the specified offset in the
 * supplied object to the given value, with volatile store semantics.
 *
 * @param obj the object containing the field to modify.
 * @param offset the offset of the integer field within <code>obj</code>.
 * @param value the new value of the field.
 */
public native void putIntVolatile(Object obj,long offset, int value);

/***
 * Retrieves the value of the integer field at the specified offset in the
 * supplied object with volatile load semantics.
 *
 * @param obj the object containing the field to read.
 * @param offset the offset of the integer field within <code>obj</code>.
 */
public native int getIntVolatile(Object obj,long offset);
```

```
/**
 * Sets the value of the long field at the specified offset in the
 * supplied object to the given value, with volatile store semantics.
 *
 * @param obj the object containing the field to modify.
 * @param offset the offset of the long field within <code>obj</code>.
 * @param value the new value of the field.
 * @see #putLong(Object,long,long)
 */
public native void putLongVolatile(Object obj,long offset, long value);

/**
 * Sets the value of the long field at the specified offset in the
 * supplied object to the given value.
 *
 * @param obj the object containing the field to modify.
 * @param offset the offset of the long field within <code>obj</code>.
 * @param value the new value of the field.
 * @see #putLongVolatile(Object,long,long)
 */
public native void putLong(Object obj, long offset, long value);

/**
 * Retrieves the value of the long field at the specified offset in the
 * supplied object with volatile load semantics.
 *
 * @param obj the object containing the field to read.
 * @param offset the offset of the long field within <code>obj</code>.
 * @see #getLong(Object,long)
 */
public native long getLongVolatile(Object obj,long offset);

/**
 * Retrieves the value of the long field at the specified offset in the
 * supplied object.
 *
 * @param obj the object containing the field to read.
 * @param offset the offset of the long field within <code>obj</code>.
 * @see #getLongVolatile(Object,long)
 */
public native long getLong(Object obj, long offset);

/**
 * Sets the value of the object field at the specified offset in the
 * supplied object to the given value, with volatile store semantics.
 *
 * @param obj the object containing the field to modify.
 * @param offset the offset of the object field within <code>obj</code>.
 * @param value the new value of the field.
 * @see #putObject(Object,long,Object)
 */
public native void putObjectVolatile(Object obj,long offset, Object value);

/**
 * Sets the value of the object field at the specified offset in the
 * supplied object to the given value.
 *
 * @param obj the object containing the field to modify.
 * @param offset the offset of the object field within <code>obj</code>.
 * @param value the new value of the field.
 * @see #putObjectVolatile(Object,long,Object)
 */
public native void putObject(Object obj, long offset, Object value);

/**
 * Retrieves the value of the object field at the specified offset in the
 * supplied object with volatile load semantics.
 *
 * @param obj the object containing the field to read.
 * @param offset the offset of the object field within <code>obj</code>.
 */
public native Object getObjectVolatile(Object obj,long offset);

/**
 * Returns the offset of the first element for a given array class.
 * To access elements of the array class, this value may be used along
 * with that returned by
 * <a href="#arrayIndexScale"><code>arrayIndexScale</code></a>,
 * if non-zero.
```

```
     *
     * @param arrayClass the class for which the first element's address should
     *                   be obtained.
     * @return the offset of the first element of the array class.
     * @see arrayIndexScale(Class)
     */
    public native int arrayBaseOffset(Class arrayClass);

    /**
     * Returns the scale factor used for addressing elements of the supplied
     * array class.  Where a suitable scale factor can not be returned (e.g.
     * for primitive types), zero should be returned.  The returned value
     * can be used with
     * <a href="#arrayBaseOffset"><code>arrayBaseOffset</code></a>
     * to access elements of the class.
     *
     * @param arrayClass the class whose scale factor should be returned.
     * @return the scale factor, or zero if not supported for this array class.
     */
    public native int arrayIndexScale(Class arrayClass);

    /**
     * Releases the block on a thread created by
     * <a href="#park"><code>park</code></a>.  This method can also be used
     * to terminate a blockage caused by a prior call to <code>park</code>.
     * This operation is unsafe, as the thread must be guaranteed to be
     * live.  This is true of Java, but not native code.
     *
     * @param thread the thread to unblock.
     */
    public native void unpark(Thread thread);

    /**
     * Blocks the thread until a matching
     * <a href="#unpark"><code>unpark</code></a> occurs, the thread is
     * interrupted or the optional timeout expires.  If an <code>unpark</code>
     * call has already occurred, this also counts.  A timeout value of zero
     * is defined as no timeout.  When <code>isAbsolute</code> is
     * <code>true</code>, the timeout is in milliseconds relative to the
     * epoch.  Otherwise, the value is the number of nanoseconds which must
     * occur before timeout.  This call may also return spuriously (i.e.
     * for no apparent reason).
     *
     * @param isAbsolute true if the timeout is specified in milliseconds from
     *                   the epoch.
     * @param time either the number of nanoseconds to wait, or a time in
     *             milliseconds from the epoch to wait for.
     */
    public native void park(boolean isAbsolute, long time);

}
```

Unsafe.h

```
// DO NOT EDIT THIS FILE - it is machine generated -*- c++ -*-

#ifndef __sun_misc_Unsafe__
#define __sun_misc_Unsafe__

#pragma interface

#include <java/lang/Object.h>
extern "Java"
{
  namespace sun
  {
    namespace misc
    {
        class Unsafe;
    }
  }
}

class sun::misc::Unsafe : public ::java::lang::Object
{

  Unsafe();
```

```
public:
  static ::sun::misc::Unsafe * getUnsafe();
  virtual jlong objectFieldOffset(::java::lang::reflect::Field *);
  virtual jboolean compareAndSwapInt(::java::lang::Object *, jlong, jint, jint);
  virtual jboolean compareAndSwapLong(::java::lang::Object *, jlong, jlong, jlong);
  virtual jboolean compareAndSwapObject(::java::lang::Object *, jlong,
::java::lang::Object *, ::java::lang::Object *);
  virtual void putOrderedInt(::java::lang::Object *, jlong, jint);
  virtual void putOrderedLong(::java::lang::Object *, jlong, jlong);
  virtual void putOrderedObject(::java::lang::Object *, jlong, ::java::lang::Object *);
  virtual void putIntVolatile(::java::lang::Object *, jlong, jint);
  virtual jint getIntVolatile(::java::lang::Object *, jlong);
  virtual void putLongVolatile(::java::lang::Object *, jlong, jlong);
  virtual void putLong(::java::lang::Object *, jlong, jlong);
  virtual jlong getLongVolatile(::java::lang::Object *, jlong);
  virtual jlong getLong(::java::lang::Object *, jlong);
  virtual void putObjectVolatile(::java::lang::Object *, jlong, ::java::lang::Object *);
  virtual void putObject(::java::lang::Object *, jlong, ::java::lang::Object *);
  virtual ::java::lang::Object * getObjectVolatile(::java::lang::Object *,jlong);
  virtual jint arrayBaseOffset(::java::lang::Class *);
  virtual jint arrayIndexScale(::java::lang::Class *);
  virtual void unpark(::java::lang::Thread *);
  virtual void park(jboolean, jlong);
private:
  static ::sun::misc::Unsafe * unsafe;
public:
  static ::java::lang::Class class$;
};

#endif // __sun_misc_Unsafe__
```

ServiceConfigurationError.h

```
// DO NOT EDIT THIS FILE - it is machine generated -*- c++ -*-

#ifndef __sun_misc_ServiceConfigurationError__
#define __sun_misc_ServiceConfigurationError__

#pragma interface

#include <java/lang/Error.h>
extern "Java"
{
  namespace sun
  {
    namespace misc
    {
        class ServiceConfigurationError;
    }
  }
}

class sun::misc::ServiceConfigurationError :public ::java::lang::Error
{

public:
  ServiceConfigurationError();
  ServiceConfigurationError(::java::lang::String*);
  static ::java::lang::Class class$;
};

#endif // __sun_misc_ServiceConfigurationError__
```

Service.h

```
// DO NOT EDIT THIS FILE - it is machine generated -*- c++ -*-

#ifndef __sun_misc_Service__
#define __sun_misc_Service__

#pragma interface

#include <java/lang/Object.h>
```

```
extern "Java"
{
  namespace sun
  {
    namespace misc
    {
        class Service;
    }
  }
}

class sun::misc::Service : public ::java::lang::Object
{

public:
  Service();
  static ::java::util::Iterator * providers(::java::lang::Class *,
::java::lang::ClassLoader *);
  static ::java::lang::Class class$;
};

#endif // __sun_misc_Service__
```

natUnsafe.cc

```
// natUnsafe.cc - Implementation of sun.misc.Unsafe native methods.

/** Copyright (C) 2006, 2007
   Free Software Foundation

   This file is part of libgcj.

This software is copyrighted work licensed under the terms of the
Libgcj License.  Please consult the file "LIBGCJ_LICENSE" for
details.  */

#include <gcj/cni.h>
#include <gcj/field.h>
#include <gcj/javaprims.h>
#include <jvm.h>
#include <sun/misc/Unsafe.h>
#include <java/lang/System.h>
#include <java/lang/InterruptedException.h>

#include <java/lang/Thread.h>
#include <java/lang/Long.h>

#include "sysdep/locks.h"

// Use a spinlock for multi-word accesses
class spinlock
{
  static volatile obj_addr_t lock;

public:

spinlock ()
  {
    while (! compare_and_swap (&lock, 0, 1))
      _Jv_ThreadYield ();
  }
  ~spinlock ()
  {
    release_set (&lock, 0);
  }
};

// This is a single lock that is used for all synchronized accesses if
// the compiler can't generate inline compare-and-swap operations.  In
// most cases it'll never be used, but the i386 needs it for 64-bit
// locked accesses and so does PPC32.  It's worth building libgcj with
// target=i486 (or above) to get the inlines.
volatile obj_addr_t spinlock::lock;


static inline bool
compareAndSwap (volatile jint *addr, jint old, jint new_val)
```

```
{
  jboolean result = false;
  spinlock lock;
  if ((result = (*addr == old)))
    *addr = new_val;
  return result;
}

static inline bool
compareAndSwap (volatile jlong *addr, jlong old, jlong new_val)
{
  jboolean result = false;
  spinlock lock;
  if ((result = (*addr == old)))
    *addr = new_val;
  return result;
}

static inline bool
compareAndSwap (volatile jobject *addr, jobject old, jobject new_val)
{
  jboolean result = false;
  spinlock lock;
  if ((result = (*addr == old)))
    *addr = new_val;
  return result;
}


jlong
sun::misc::Unsafe::objectFieldOffset (::java::lang::reflect::Field *field)
{
  _Jv_Field *fld = _Jv_FromReflectedField (field);
  // FIXME: what if it is not an instance field?
  return fld->getOffset();
}

jint
sun::misc::Unsafe::arrayBaseOffset (jclass arrayClass)
{
  // FIXME: assert that arrayClass is array.
  jclass eltClass = arrayClass->getComponentType();
  return (jint)(jlong) _Jv_GetArrayElementFromElementType (NULL, eltClass);
}

jint
sun::misc::Unsafe::arrayIndexScale (jclass arrayClass)
{
  // FIXME: assert that arrayClass is array.
  jclass eltClass = arrayClass->getComponentType();
  if (eltClass->isPrimitive())
    return eltClass->size();
  return sizeof (void *);
}

// These methods are used when the compiler fails to generate inline
// versions of the compare-and-swap primitives.

jboolean
sun::misc::Unsafe::compareAndSwapInt (jobject obj, jlong offset,
                      jint expect, jint update)
{
  jint *addr = (jint *)((char *)obj + offset);
  return compareAndSwap (addr, expect, update);
}

jboolean
sun::misc::Unsafe::compareAndSwapLong (jobject obj, jlong offset,
                       jlong expect, jlong update)
{
  volatile jlong *addr = (jlong*)((char *) obj + offset);
  return compareAndSwap (addr, expect, update);
}

jboolean
sun::misc::Unsafe::compareAndSwapObject (jobject obj, jlong offset,
                     jobject expect, jobject update)
{
  jobject *addr = (jobject*)((char *) obj + offset);
  return compareAndSwap (addr, expect, update);
```

```
}

void
sun::misc::Unsafe::putOrderedInt (jobject obj, jlong offset, jint value)
{
  volatile jint *addr = (jint *) ((char *) obj + offset);
  *addr = value;
}

void
sun::misc::Unsafe::putOrderedLong (jobject obj, jlong offset, jlong value)
{
  volatile jlong *addr = (jlong *) ((char *) obj + offset);
  spinlock lock;
  *addr = value;
}

void
sun::misc::Unsafe::putOrderedObject (jobject obj, jlong offset, jobject value)
{
  volatile jobject *addr = (jobject *) ((char *) obj + offset);
  *addr = value;
}

void
sun::misc::Unsafe::putIntVolatile (jobject obj, jlong offset, jint value)
{
  write_barrier ();
  volatile jint *addr = (jint *) ((char *) obj + offset);
  *addr = value;
}

void
sun::misc::Unsafe::putLongVolatile (jobject obj, jlong offset, jlong value)
{
  volatile jlong *addr = (jlong *) ((char *) obj + offset);
  spinlock lock;
  *addr = value;
}

void
sun::misc::Unsafe::putObjectVolatile (jobject obj, jlong offset, jobject value)
{
  write_barrier ();
  volatile jobject *addr = (jobject *) ((char *) obj + offset);
  *addr = value;
}

#if 0  // FIXME
void
sun::misc::Unsafe::putInt (jobject obj, jlong offset, jint value)
{
  jint *addr = (jint *) ((char *) obj + offset);
  *addr = value;
}
#endif

void
sun::misc::Unsafe::putLong (jobject obj, jlong offset, jlong value)
{
  jlong *addr = (jlong *) ((char *) obj + offset);
  spinlock lock;
  *addr = value;
}

void
sun::misc::Unsafe::putObject (jobject obj, jlong offset, jobject value)
{
  jobject *addr = (jobject *) ((char *) obj + offset);
  *addr = value;
}

jint
sun::misc::Unsafe::getIntVolatile (jobject obj, jlong offset)
{
  volatile jint *addr = (jint *) ((char *) obj + offset);
  jint result = *addr;
  read_barrier ();
  return result;
}
```

```
jobject
sun::misc::Unsafe::getObjectVolatile (jobject obj, jlong offset)
{
  volatile jobject *addr = (jobject *) ((char *) obj + offset);
  jobject result = *addr;
  read_barrier ();
  return result;
}

jlong
sun::misc::Unsafe::getLong (jobject obj, jlong offset)
{
  jlong *addr = (jlong *) ((char *) obj + offset);
  spinlock lock;
  return *addr;
}

jlong
sun::misc::Unsafe::getLongVolatile (jobject obj, jlong offset)
{
  volatile jlong *addr = (jlong *) ((char *) obj + offset);
  spinlock lock;
  return *addr;
}

void
sun::misc::Unsafe::unpark (::java::lang::Thread*thread)
{
  natThread *nt = (natThread *) thread->data;
  nt->park_helper.unpark ();
}

void
sun::misc::Unsafe::park (jboolean isAbsolute, jlong time)
{
  using namespace ::java::lang;
  Thread *thread = Thread::currentThread();
  natThread *nt = (natThread *) thread->data;
  nt->park_helper.park (isAbsolute, time);
}
```

----------我和我追猪的梦----------

作者： mickole
出处： http://www.cnblogs.com/mickole/

« 上一篇： LeetCode124:Binary Tree Maximum Path Sum

posted @ 2014-05-28 16:09 mickole 阅读(4527) 评论(0) 编辑 收藏