JAVA 1.6源码解析 - Object类

```
/*
 * @(#)Object.java 1.73 06/03/30
 *
 * Copyright 2006 Sun Microsystems, Inc. All rights reserved.
 * SUN PROPRIETARY/CONFIDENTIAL. Use is subject to license terms.
 */

package java.lang;

/**
 * Class <code>Object</code> is the root of the class hierarchy.
 * Every class has <code>Object</code> as a superclass. All objects,
 * including arrays, implement the methods of this class.
 *
 * @author  unascribed
 * @version 1.73, 03/30/06
 * @see     java.lang.Class
 * @since   JDK1.0
 */
public class Object {
 /**
  * 注册本地方法 用来让JVM能够找到本地方法，
  * 一般来说 命名本地方法 需要按照一定的命名规则。如java.lang.Object.registerNatives对应的C函数叫做
Java_java_lang_Object_registerNatives
  * 为了让编程人员可以自由的命名本地函数，进而提供了registerNatives这个函数，这个绑定的工作有registerNatives来完成。
  */
    private static native void registerNatives();
    static {
        registerNatives();
    }
    /**
     *用来获取当前运行时对象的类名称
     *这个方法被声明为final、native表明这个方法是一个 不可变的原生方法 也就是说 只可以继承、不能重写。
     *native方法 表明 是编译成DLL 由系统底层实现的 我们只管调用就OK。
     */
    public final native Class<?> getClass();

    /**
     * 获取对象的hash值，在JAVA中系统会为每一个对象生成一个唯一的hash值，用来区别对象，hash值也是判断对象是否相等的标
尺。
     * 此方法也是native型，由JAVA底层实现。由于不是final型 所以 如果需要可以进行重写
     */
    public native int hashCode();

    /**
     * 判断两个对象是否相等 从方法中可以看出JAVA中比较两个对象是否相等的默认方法是比较两个对象的内存地址，如果它们的内存地
址相等则认为它们是相等的
     * 如果实际情况中碰到其他特别情况，只需要对象某一部分值相等就认为相等，那么就需要重写这个方法。
     */
    public boolean equals(Object obj) {
  return (this == obj);
    }

    /**
     * 复制一个对象，复制对象的属性与当前对象一致，但是并不相等，是一个新的对象。同时要调用这个方法，所在类必须去实现
Cloneable接口
     * 否则报不运行克隆方法的错误。用的比较多的是数组的复制。
     */
    protected native Object clone() throws CloneNotSupportedException;
    /**
     * 当前对象转换为String对象的值
     * 默认为当前对象所在类名+对象的hash值。这样的话是非常不人性化的，所以如果要人性化，个性化输出的话就必须重写这个方法。
     */
```

```
    public String toString() {
 return getClass().getName() + "@" + Integer.toHexString(hashCode());
    }

   /**
    * 唤醒当前对象监控列表中等待的一个线程
    * 这个方法也是final native型 即不可以重写 由JAVA底层实现 作用是用来唤醒一个等待当前对象的线程。
    * 线程被唤醒后并不会马上执行，它将同其他监控当前对象的线程一样再次平等地竞争当前对象的所有权，每次只会有一个线程得到当前对象的所有权。
    */
    public final native void notify();

    /**
     * 唤醒当前对象监控列表中所有等待的线程
     * 这个方法也是final native型 即不可以重写 由JAVA底层实现 作用是用来唤醒一个等待当前对象的线程。
     * 线程被唤醒后并不会马上执行，它将同其他监控当前对象的线程一样再次平等地竞争当前对象的所有权，每次只会有一个线程得到当前对象的所有权。
     */
    public final native void notifyAll();

   /**
    * 将当前线程加入到等待/阻塞的Set集合中，直到有其他线程调用notify()或notifyAll()方来唤醒，或者等待的时间timeout已经到达或者有其他线程
    * 调用了interrupt()方法来打断当前线程，否则将一直在等待的Set集合中。
    */
    public final native void wait(long timeout) throws InterruptedException;

   /**
    * 这个方法同wait()方法相同 ，唯一的区别就是等待时间控制更加精细，可以精确到纳秒(通过nanos参数)
    * 实现等待时间=(1000000*timeout+nanos)
    */
    public final void wait(long timeout, int nanos) throws InterruptedException {
        if (timeout < 0) {
            throw new IllegalArgumentException("timeout value is negative");
        }

        if (nanos < 0 || nanos > 999999) {
            throw new IllegalArgumentException(
    "nanosecond timeout value out of range");
        }

 if (nanos >= 500000 || (nanos != 0 && timeout == 0)) {
     timeout++;
 }

 wait(timeout);
    }

    /**
     * 通过简单的调用wait(0)方法来释放线程中的对当前监控对象的所有权，从而让其他线程可以拿到对象的所有权而进入到同步方法中，
     * 等待直到notify()或notifyAll()方法被调用，这样才能再次获取对象的所有权。
     */
    public final void wait() throws InterruptedException {
 wait(0);
    }

    /**
     * 调用JVM中的垃圾收集来进行内存回收，但这取决于当前对象与GCRoots之间是不是已经不可到达，也就是说没有其他引用链接。如果没有，则会被回收，
     * 否则不会。同时，调用了finalize()方法也不保证当前对象会被垃圾回收器立即回收。垃圾回收线程的优先级是比较低的。
     */
    protected void finalize() throws Throwable { }
}
```