# 1：equals和hashcode

http://www.cnblogs.com/vincent4code/p/5321834.html

 1：equals()相等的两个对象，hashcode()一定相等，equals()不相等的两个对象，却并不能证明他们的hashcode()不相等。换句话说，equals()方法不相等的两个对象，hashCode()有可能相等。（我的理解是由于哈希码在生成的时候产生冲突造成的）

2：总结：两者满足的要求或规范：反过来：hashcode()不等，一定能推出equals()也不等；hashcode()相等，equals()可能相等，也可能不等。

3：在object类中，hashcode()方法是本地方法，返回的是对象的地址值，而object类中的equals()方法比较的也是两个对象的地址值，如果equals()相等，说明两个对象地址值也相等，当然hashcode() 也就相等了；

toString（）

1. **public** String toString() {
2. **return** getClass().getName() + "@" + Integer.toHexString(hashCode());
3. }

4.当String 、Math、Integer、Double等这些封装类在使用equals()方法时，已经覆盖了Object类的equals()方法。

类中覆盖hashcode()方法，比如String、Integer、Double

5.hashcode主=主要就是，当你重写了equals后，若是相同的对象，要能存储到同一个位置上。所以要重写.hashcode（） .hashcode（）主要就是快速定位到要存储的地址。

5：扩展：

关于equals方法：

1）首先equals方法必须满足

自反性（x.equals(x)必须返回true）、

对称性（x.equals(y)返回true时，y.equals(x)也必须返回true）、

传递性（x.equals(y)和y.equals(z)都返回true时，x.equals(z)也必须返回true）

和一致性（当x和y引用的对象信息没有被修改时，多次调用x.equals(y)应该得到同样的返回值），

而且对于任何非null值的引用x，x.equals(null)必须返回false。

2）实现高质量的equals方法的诀窍包括：

1. 使用==操作符检查"参数是否为这个对象的引用"；

2. 使用instanceof操作符检查"参数是否为正确的类型"；

3. 对于类中的关键属性，检查参数传入对象的属性是否与之相匹配；

4. 编写完equals方法后，问自己它是否满足对称性、传递性、一致性；

5. 重写equals时总是要重写hashCode；

6. 不要将equals方法参数中的Object对象替换为其他的类型，在重写时不要忘掉@Override注解

6.用在哪：

集合中用

# 2：compare与compareTo方法

## 一：compareTo

1,compareTo(Object o)方法是**[Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "_blank)**.lang.Comparable<T>接口中的方法，当需要对某个类的对象进行排序时，该类需要实现Comparable<T>接口的，必须重写public int compareTo (T o)方法。

2：**java中的compareto方法，返回参与比较的前后两个字符串的asc码的差值**

**3：也就是说，如果两个字符串首字母不同，则该方法返回首字母的asc码的差值:**

**4：即参与比较的两个字符串如果首字符相同，则比较下一个字符，直到有不同的为止，返回该不同的字符的asc码差值，如果两个字符串不一样长，可以参与比较的字符又完全一样，则返回两个字符串的长度差值**

2,compare(Object o1,Object o2)方法是java.util.Comparator<T>接口的方法，它实际上用的是待比较对象的compareTo(Object o)方法。

# 3：排序方法

1：collections.sort()

2：Arrays.sort()

这两个的默认的排序器都是comparable，但可以自己添加comparator的排序器

# 4：string，stringbuffer，StringBuilder

1：其实这三个类都是用finnal修饰的但是，integer也是final的

所以这类final的对象想要值改变的话就是新建一同一个对象在将值改变了。

string的

/\*\* The value is used for character storage. \*/

**private** **final** **char** value[];

/\*\* Cache the hash code for the string \*/

**private** **int** hash; // Default to 0

/\*\* use serialVersionUID from JDK 1.0.2 for interoperability \*/

**private** **static** **final** **long** ***serialVersionUID*** = -6849794470754667710L;

并且该类中的成员方法都默认为final方法

2：stringbuff

\* A cache of the last value returned by toString. Cleared

\* whenever the StringBuffer is modified.

\*/

**private** **transient** **char**[] toStringCache;

/\*\* use serialVersionUID from JDK 1.0.2 for interoperability \*/

**static** **final** **long** ***serialVersionUID*** = 3388685877147921107L;

3：StringBuilder

**abstract** **class** AbstractStringBuilder **implements** Appendable, CharSequence {

/\*\*

\* The value is used for character storage.

\*/

**char**[] value;

/\*\*

\* The count is the number of characters used.

\*/

**int** count;

# 5：object类的方法

http://blog.csdn.net/cangchen/article/details/44977881

首先大家要明白回调(callback)方法，所谓回调方法就是程序在运行特定功能时JVM会自动调动这些方法，假设你使用System.out.print(obj)打印出对象obj信息，则运行时JVM会自动调用obj对象的toString()方法，toString()方法就是回调方法。理解回调方法后，下面我们来看这些方法：

## Wait

1：执行了该方法的线程释放对象的锁，JVM会把该线程放到对象的等待池中。该线程等待其它线程唤醒

2：wait还有带有时间参数的版本，在等待了超过所设时间之后，T1线程一样会被唤醒，进入到争夺obj对象锁的行列；另外中断可以直接跳出wait方法） 普通的就是notify notifyall

## Notify

1：执行了该方法的线程释放对象的锁，JVM会把该线程放到对象的等待池中。该线程等待其它线程唤醒NotifAll

2：该方法的调用，会从所有正在等待obj对象锁的线程中，唤醒其中的一个（选择算法依赖于不同实现），被唤醒的线程此时加入到了obj对象锁的争夺之中，然而该notify方法的执行线程此时并未释放obj的对象锁，而是离开synchronized代码块时释放。因此在notify方法之后，synchronized代码块结束之前，所有其他被唤醒的，等待obj对象锁的线程依旧被阻塞

## Equals

默认的equals的是==只有当两个对象地址相同时才返回true

## Clone

## Finalize

这是GC清理对象之前所调用的清理方法，是回调方法，我们可以覆盖这个方法写一些清理的代码，GC会自动扫描没有引用的对象，即对象赋值为null；可以通过调用System.runFinalization()或System.runFinalizersOnExit()强制GC清理该对象前调用finalize()方法，GC有时不会调用对象的finalize()方法(由JVM决定)；

## Getclass

## Hashcode

默认的hashCode()返回的值就是对象在内存中的地址。

## Tostring

而默认的toString()方法就是打印出对象的地址。