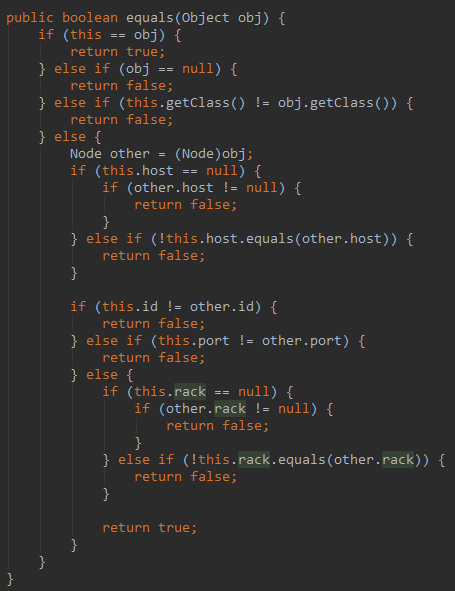
## 如何重写equals方法

在JavaSE 7中，对于equals方法有几点要求：

1. 自反性：a.equals(a)->true
2. 对称性：a.equals(b)->true => b.equals(a)->true
3. 传递性：a.equals(b)->true，b.equals(c)->true => a.equals(c)->true
4. 一致性：a，b状态不改变，a.equals(b)返回值必须相同
5. a.equals(null)->false

所以写法：

1. this==obj：两个引用是否指向同一个对象，自反性；
2. 是否为null，最后一条
3. 判断class对象是否是同一个，不是同一个也不用比了；
4. 开始比较双方所有的字段值，这里要注意在调用equals方法时候要避免null的出现；



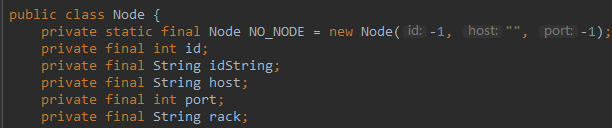
## 重写hashCode的目的

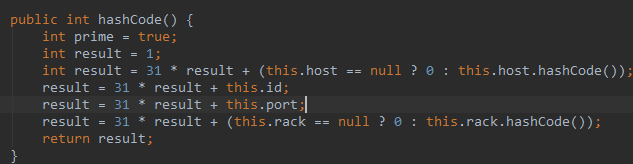
在A.equals(B)返回true的情况下，A, B 的hashCode()要返回相同的值。

举个例子，hash table使用了数组就链表的结构实现，通过hashCode实现定位，若hashCode不同，那么即便相同的对象(即equals为true)也会找不到。

## hashCode方法里的数字31

hashCode方法里为什么选择数字31作为生成hashCode值的乘数





这个是kafka源码中的hashCode方法，偶然发现这个特点。可以看出它的hashCode计算方法：result=31\*result+字段(若字段为int则直接计算，反之则要调用hashCode())。

第一原因：更少的乘积结果冲突

31是质子数中一个“不大不小”的存在，如果你使用的是一个如2的较小质数，那么得出的乘积会在一个很小的范围，很容易造成哈希值的冲突。而如果选择一个100以上的质数，得出的哈希值会超出int的最大范围，这两种都不合适。而如果对超过 50,000 个英文单词（由两个不同版本的 Unix 字典合并而成）进行 hash code 运算，并使用常数 31, 33, 37, 39 和 41 作为乘子，每个常数算出的哈希值冲突数都小于7个（国外大神做的测试），那么这几个数就被作为生成hashCode值的备选乘数了。(总结一句，就是测试出来的)

第二个原因：31可以被JVM优化

JVM里最有效的计算方式就是进行位运算了：

\* 左移 << : 左边的最高位丢弃，右边补全0（把 << 左边的数据\*2的移动次幂）。

\* 右移 >> : 把>>左边的数据/2的移动次幂。

\* 无符号右移 >>> : 无论最高位是0还是1，左边补齐0。

所以 ： 31 \* i = (i << 5) - i（左边 31\*2=62,右边 2\*2^5-2=62） - 两边相等，JVM就可以高效的进行计算啦。。。