一，哈希算法的概念  
  
    在计算机领域，哈希算法具有非常广泛的应用，比如快速查找和加密。今天我们来讨论一下哈希算法。我们先从理论知识开始。  
  
1，什么是哈希算法  
    百科中，从哈希算法的功能上，对哈希算法进行了定义。百科是这样定义哈希算法的：哈希算法可以将任意长度的二进制值映射  
为较短的，固定长度的二进制值。我们把这个二进制值成为哈希值。  
  
  
2，哈希值的特点  
  \* 哈希值是二进制值；  
  \* 哈希值具有一定的唯一性；  
  \* 哈希值极其紧凑；  
  \* 要找到生成同一个哈希值的2个不同输入，在一定时间范围内，是不可能的。  
  
    正因为哈希值的这些特点，使得哈希算法应用在加密领域成为可能。哈希算法在加密领域的应用，源于哈希算法的不可逆性，对于  
用户输入的密码，通过哈希算法可以得到一个哈希值。并且，同一个密码，生成的哈希值总是相等的。这样，服务器就可以在不知道  
用户输入的密码的情况下，判断用户输入的密码是否正确。  
  
3，哈希表  
    哈希表是一种数据机构。哈希表根据关键字（key），生成关键字的哈希值，然后通过哈希值映射关键字对应的值。哈希表存储了多  
余的key（我们本可以只存储值的），是一种用空间换时间的做法。在内存足够的情况下，这种“空间换时间”的做法是值得的。哈希表的  
产生，灵感来源于数组。我们知道，数组号称查询效率最高的数据结构，因为不管数组的容量多大，查询的时间复杂度都是O(1)。如果  
所有的key都是不重复的整数，那么这就完美了，不需要新增一张哈希表，来做关键字（key）到值（value）的映射。但是，如果key是  
字符串，情况就不一样了。我们必须要来建一张哈希表，进行映射。  
    数据库索引的原理，其实和哈希表是相同的。数据库索引也是用空间换时间的做法。  
  
二，哈希算法的具体实现  
  
    哈希算法在不同的语言，具有不同的实现。这里我们以java语言为例来进行说明。  
  
1，哈希算法在HashMap类中的应用  
    java中的集合，比如HashMap/Hashtable/HashSet等，在实现时，都用到了哈希算法。当我们向容器中添加元素时，我们有必要知道  
这个元素是否已经存在。  
    从实现上来说，java是借助hashcode()方法和equals()方法来实现判断元素是否已经存在的。当我们向HashMap中插入元素A时，首先，  
调用hashcode()方法，判断元素A在容器中是否已经存在。如果A元素的hashcode值在HashMap中不存在，则直接插入。否则，接着调用  
equals()方法，判断A元素在容器中是否已经存在。hashcode()的时间复杂度为O(1)，equals()方法的时间复杂度为O(m)，整体的时间复杂度  
就是：O(1) + O(m)。其中，m是桶的深度。桶的深度是一个什么概念呢，桶的深度是指具有相同hashcode值得元素的个数，也就是发生哈希  
碰撞的元素的个数。  
    一个好的哈希算法应该尽量减少哈希碰撞的次数。  
  
2，哈希算法在String类中的应用

    Sring类重写了Object类的equals()方法和hashcode()方法。hashcode()方法的源代码如下：

**[java]** [view plain](http://blog.csdn.net/reggergdsg/article/details/53819293" \o "view plain" \t "_blank) [copy](http://blog.csdn.net/reggergdsg/article/details/53819293" \o "copy" \t "_blank)

1. **public** **int** hashCode() {
2. **int** h = hash;
3. **if** (h == 0) {
4. **int** off = offset;
5. **char** val[] = value;
6. **int** len = count;
8. **for** (**int** i = 0; i < len; i++) {
9. h = 31\*h + val[off++];
10. }
11. hash = h;
12. }
13. **return** h;
14. }

    源代码写的比较简洁，阅读起来也不是太方便，下面我详细解读一下：  
// String类的hashcode值（哈希值）是如何计算得到的？具体实现？为了方便阅读，我们来进行分步说明

**[java]** [view plain](http://blog.csdn.net/reggergdsg/article/details/53819293" \o "view plain" \t "_blank) [copy](http://blog.csdn.net/reggergdsg/article/details/53819293" \o "copy" \t "_blank)

1. **static** **void** hashcodeTest(){
3. String str = "yangcq";
5. // 第一步 = (int)'y'
6. // 第二步 = (31 \* (int)'y') + (int)'a'
7. // 第三步 = 31 \* ((31 \* (int)'y') + (int)'a') + (int)'n'
8. // 第四步 = 31 \* (31 \* ((31 \* (int)'y') + (int)'a') + (int)'n') + (int)'g'
9. // 第五步 = 31 \* (31 \* (31 \* ((31 \* (int)'y') + (int)'a') + (int)'n') + (int)'g') + (int)'c'
10. // 第六步 = 31 \* (31 \* (31 \* (31 \* ((31 \* (int)'y') + (int)'a') + (int)'n') + (int)'g') + (int)'c') + (int)'q'
12. // 上面的过程，也可以用下面的方式表示
14. // 第一步 = (int)'y'
15. // 第二步 = 31 \* (第一步的计算结果) + (int)'a'
16. // 第三步 = 31 \* (第二步的计算结果) + (int)'n'
17. // 第四步 = 31 \* (第三步的计算结果) + (int)'g'
18. // 第五步 = 31 \* (第四步的计算结果) + (int)'c'
19. // 第六步 = 31 \* (第五步的计算结果) + (int)'q'
21. **int** hashcode = 31 \* (31 \* (31 \* (31 \* ((31 \* (**int**)'y') + (**int**)'a') + (**int**)'n') + (**int**)'g') + (**int**)'c') + (**int**)'q';
22. System.out.println("yangcq的hashcode = " + hashcode);        // yangcq的hashcode = -737879313
23. System.out.println("yangcq的hashcode = " + str.hashCode());  // yangcq的hashcode = -737879313
25. }

    通过上面的测试方法，我们可以很清晰的看到hashcode()方法具体的计算过程。下面再贴上2个类，一个是自己写的测试类  
MyHashcode.java，另一个是HashcodeOfString.java

**[java]** [view plain](http://blog.csdn.net/reggergdsg/article/details/53819293" \o "view plain" \t "_blank) [copy](http://blog.csdn.net/reggergdsg/article/details/53819293" \o "copy" \t "_blank)

1. /\*\*
2. \* java中对象的hashcode值，是如何计算得到的。
3. \*/
4. **public** **class** MyHashcode {
6. /\*\* The value is used for character storage. \*/
7. **static** **char** value[];
9. /\*\* The offset is the first index of the storage that is used. \*/
10. **static** **int** offset;
12. /\*\* The count is the number of characters in the String. \*/
13. **static** **int** count;
15. /\*\* Cache the hash code for the string \*/
16. **static** **int** hash; // Default to 0
18. /\*\*
19. \* @param args
20. \*/
21. **public** **static** **void** main(String[] args) {
22. // TODO Auto-generated method stub
23. String str1 = **new** String("yangcq");
24. String str2 = **new** String("yangcq");
25. // 如果2个字符串的内容相同，那么这2个字符串的hashcode必然相同
26. System.out.println(**new** String("yangcq").hashCode() == **new** String("yangcq").hashCode());
27. System.out.println(str1.hashCode() == str2.hashCode());
29. System.out.println(str1.hashCode());
30. System.out.println(hashCode1(str1));
32. // 测试自定义的hashcode方法
33. HashcodeOfString hashcodeOfString = **new** HashcodeOfString();
34. hashcodeOfString.hashCode(str1);
35. System.out.println("str1的hashcode = " + hashcodeOfString.hashCode(str1));
36. System.out.println("str1的hashcode = " + str1.hashCode());
37. }
39. // HashMap中实现的hash算法(再hash算法)
40. **static** **int** hash(**int** h) {
41. // This function ensures that hashCodes that differ only by
42. // constant multiples at each bit position have a bounded
43. // number of collisions (approximately 8 at default load factor).
44. h ^= (h >>> 20) ^ (h >>> 12);
45. **return** h ^ (h >>> 7) ^ (h >>> 4);
46. }
48. // String类实现的hashcode方法源代码
49. **static** **int** hashCode1(String str) {
50. **int** h = hash;
51. **if** (h == 0) {
52. **int** off = 0;
53. **char** val[] = str.toCharArray();
54. **int** len = str.length();
56. **for** (**int** i = 0; i < len; i++) {
57. h = 31 \* h + val[off++];
58. }
59. hash = h;
60. }
61. **return** h;
62. }
64. // String类的hashcode值（哈希值）是如何计算得到的？具体实现？为了方便阅读，我们来进行分步说明
65. **static** **void** hashcodeTest(){
67. String str = "yangcq";
69. // 第一步 = (int)'y'
70. // 第二步 = (31 \* (int)'y') + (int)'a'
71. // 第三步 = 31 \* ((31 \* (int)'y') + (int)'a') + (int)'n'
72. // 第四步 = 31 \* (31 \* ((31 \* (int)'y') + (int)'a') + (int)'n') + (int)'g'
73. // 第五步 = 31 \* (31 \* (31 \* ((31 \* (int)'y') + (int)'a') + (int)'n') + (int)'g') + (int)'c'
74. // 第六步 = 31 \* (31 \* (31 \* (31 \* ((31 \* (int)'y') + (int)'a') + (int)'n') + (int)'g') + (int)'c') + (int)'q'
76. // 上面的过程，也可以用下面的方式表示
78. // 第一步 = (int)'y'
79. // 第二步 = 31 \* (第一步的计算结果) + (int)'a'
80. // 第三步 = 31 \* (第二步的计算结果) + (int)'n'
81. // 第四步 = 31 \* (第三步的计算结果) + (int)'g'
82. // 第五步 = 31 \* (第四步的计算结果) + (int)'c'
83. // 第六步 = 31 \* (第五步的计算结果) + (int)'q'
85. **int** hashcode = 31 \* (31 \* (31 \* (31 \* ((31 \* (**int**)'y') + (**int**)'a') + (**int**)'n') + (**int**)'g') + (**int**)'c') + (**int**)'q';
86. System.out.println("yangcq的hashcode = " + hashcode);        // yangcq的hashcode = -737879313
87. System.out.println("yangcq的hashcode = " + str.hashCode());  // yangcq的hashcode = -737879313
89. }
90. }

**[java]** [view plain](http://blog.csdn.net/reggergdsg/article/details/53819293" \o "view plain" \t "_blank) [copy](http://blog.csdn.net/reggergdsg/article/details/53819293" \o "copy" \t "_blank)

1. /\*\*
2. \*
3. \* @yangcq
4. \* @描述：String类的hashcode值是如何计算得到的。
5. \* 参考String类的hashcode()方法，写一个方法，计算任意字符串的hashcode
6. \*
7. \* 哈希值的特点：
8. \* 1，一定程度的唯一性；
9. \* 2，长度固定；
10. \* 3，哈希值是二进制值；
11. \* 4，要找到生成相同哈希值的2个不同输入，在有限时间内是不可能的；哈希算法应用于加密领域的原因。
12. \*
13. \*/
14. **public** **class** HashcodeOfString **implements** java.io.Serializable{
16. **private** **static** **final** **long** serialVersionUID = -4208161728160233397L;
18. **public** **int** hashCode(String str) {
19. // 最终计算得出的哈希值，转化为int以后的哈希值
20. **int** hashcode = 0;
21. // 临时哈希值变量
22. **int** hash = 0;
23. **if** (hash == 0) {
24. // 当前char的索引
25. **int** off = 0;
26. // 字符串str的字符数组表示
27. **char** val[] = str.toCharArray();
28. // 字符串str的长度
29. **int** len = str.length();
30. **for** (**int** i = 0; i < len; i++) {
31. hash = 31 \* hash + val[off++];
32. }
33. hashcode = hash;
34. }
35. **return** hashcode;
36. }
37. }