

不再想当菜鸟了

不再想当菜鸟了

个人资料



菜鸟出行

+ 加关注

发私信

访问：331665次

积分：3953

等级：

博主

>5

排名：第8285名

原创：44篇

转载：70篇

译文：55篇

评论：13条

文章搜索

Q

文章分类

个人感悟 (13)

java编程 (17)

PHP (17)

ASP.NET (11)

IT动态 (2)

Android开发 (24)

服务器 (6)

SEO优化 (7)

数据库 (23)

健康之道 (0)

ERP (0)

HTML5 (25)

CSS (0)

JS (11)

软件工程 (1)

软件测试 (1)

创业 (2)

C# (15)

代码 (1)

PhotoShop (0)

Android开源 (0)

网络相关 (1)

文章存档

2015年09月 (3)

2015年08月 (1)

2015年07月 (1)

2015年06月 (2)

异步赠书：9月重磅新书升级，本本经典

CSDN新版博客内测群495397560期待你加入

程序员9月书讯

每周荐书：ES6、虚拟现实、物联网（评论送书）

译 应用Hash函数（java描述）

2013-07-19 21:27

1308人阅读

评

目录视图

摘要视图

RSS 订阅

分类：java编程（16）

目录(?)

[+]

作者：冲处宇宙

时间：2007.1.25

计算理论中，没有Hash函数的说法，只有单向函数的说法。所谓的单向函数，是一个复杂的定义。去看计算理论或者密码学方面的数据。用“人类”的语言描述单向函数就是：**如果某个函数在给定输入的时候，很容易计算出其结果来；而当给定结果的时候，很难计算出输入来，这就是单项函数。**各种加密函数都可以被认为是单向函数的逼近。Hash函数（或者成为散列函数）也可以看成是单向函数的一个逼近。即它接近于满足单向函数的定义。

Hash函数还有另外的含义。实际中的Hash函数是指把一个大范围映射到一个小范围。把大范围映射到一个小范围的目的往往是为了节省空间，使得数据容易保存。除此以外，Hash函数往往应用于查找上。所以，在考虑使用Hash函数之前，需要明白它的几个限制：

1. Hash的主要原理就是把大范围映射到小范围；所以，你输入的实际值的个数必须和小范围相当或者比它更小。不然冲突就会很多。

2. 由于Hash逼近单向函数；所以，你可以用它来对数据进行加密。

3. 不同的应用对Hash函数有着不同的要求；比如，用于加密的Hash函数主要考虑它和单项函数的差距，而用于查找的Hash函数主要考虑它映射到小范围的冲突率。

应用于加密的Hash函数已经探讨过太多了，在作者的博客里面有更详细的介绍。所以，本文只探讨用于查找的Hash函数。

Hash函数应用的主要对象是数组（比如，字符串），而其目标一般是一个int类型。以下我们都按照这种方式来说明。

一般的说，Hash函数可以简单的划分为如下几类：

1. 加法Hash；

2. 位运算Hash；

3. 乘法Hash；

4. 除法Hash；

5. 查表Hash；

6. 混合Hash；

下面详细的介绍以上各种方式在实际中的运用。

一 加法Hash

所谓的加法Hash就是把输入元素一个一个的加起来构成最后的结果。标准的加法Hash的构造如下：

[java]

```
01. static int additiveHash(String key, int prime)
02. {
03.     int hash, i;
04.     for (hash = key.length(), i = 0; i < key.length(); i++)
05.         hash += key.charAt(i);
06.     return (hash % prime);
07. }
```

快速回复

我要收藏

返回顶部

微信关注CSDN

获得无限技术资源

关闭

http://blog.csdn.net/zhoujn90/article/details/9385975

1/10

2015年04月 (20)

展开

阅读排行

HTML5的数据缓存

(33851)

Android Studio 安装具体

(20079)

HTML布局

(18837)

剖析Android 线性布局中

(13312)

留言板php代码。

(12935)

php函数返回值

(10357)

php程序员应具备的7种能

(10223)

Socket编程

(9822)

编程中常用的英文单词

(9761)

构造方法(java中跟类名一

(9396)

评论排行

留言板php代码。

(2)

Android Studio 安装具体

(2)

HTML Iframe (HTML内

(1)

Java连接mysql数据库攻

(1)

HTML5的数据缓存

(1)

编程中常用的英文单词

(1)

HTML 5 Canvas 和 HTM

(1)

写给新手程序员的一封信

(1)

HTML 5 拖放 如果在网页

(1)

Collection集合

(1)

推荐文章

* CSDN新版博客feed流内测用户征集令

* Android检查更新下载安装

* 动手打造史上最简单的Recycleview 侧滑菜单

* TCP网络通讯如何解决分包包问题

* SDCC 2017之大数据技术实战线上峰会

* 快速集成一个视频直播功能

最新评论

留言板php代码。
qq_34105935: 566

留言板php代码。
qq_34105935: 1233333

HTML 5 拖放 如果在网页上拖放
fly18702787810: dd

编程中常用的英文单词
eatimon: 好东西，收藏学习了。

HTML Iframe (HTML内联框架)
Pan_cras: 这个粘贴还不错

Android Studio 安装具体步骤
ALDNOAH_ZERO: 图挂了

Android简单的获取网络上的json
marzkh: 抄也要抄完啊

写给新手程序员的一封信
丁国华: 感谢分享 学习了`(*__*)`

Java连接mysql数据库攻略
v7595v: mark

HTML5的数据缓存
Sr_Cjx: 晕，怎么重复这么多啊

这里的prime是任意的质数，看得出，结果的值为[0,prime-1]。

二 位运算Hash

这类型Hash函数通过利用各种位运算（常见的是移位和异或）来充分的混合输入元素。比如，标准的旋转Hash的构造如下：

```
[java]
01. static int rotatingHash(String key, int prime)
02. {
03.     int hash, i;
04.     for (hash=key.length(), i=0; i<key.length(); ++i)
05.         hash = (hash<<4)^(hash>>28)^key.charAt(i);
06.     return (hash % prime);
07. }
```

先移位，然后再进行各种位运算是这种类型Hash函数的主要特点。比如，以上的那段计算hash的代码还可以有如下几种变形：

```
[java]
01. 1. hash = (hash<<5)^(hash>>27)^key.charAt(i);
02. 2. hash += key.charAt(i);
03.     hash += (hash << 10);
04.     hash ^= (hash >> 6);
05. 3. if((i&1) == 0)
06.     {
07.         hash ^= (hash<<7) ^ key.charAt(i) ^ (hash>>3);
08.     }
09.     else
10.     {
11.         hash ^= ~(hash<<11) ^ key.charAt(i) ^ (hash >>5));
12.     }
13. 4. hash += (hash<<5) + key.charAt(i);
14. 5. hash = key.charAt(i) + (hash<<6) + (hash>>16) - hash;
15. 6. hash ^= ((hash<<5) + key.charAt(i) + (hash>>2));
```

三 乘法Hash

这种类型的Hash函数利用了乘法的不相关性（乘法的这种性质，最有名的莫过于平方取头尾的随机数生成算法，虽然这种算法效果并不好）。比如，

```
[java]
01. static int bernstein(String key)
02. {
03.     int hash = 0;
04.     int i;
05.     for (i=0; i<key.length(); ++i) hash = 33*hash + key.charAt(i);
06.     return hash;
07. }
```

jdk5.0里面的String类的hashCode()方法也使用乘法Hash。不过，它使用的乘数是31。推荐的乘数还有：131，1313，13131，131313等等。

使用这种方式的著名Hash函数还有：

```
[java]
01. // 32位FNV算法
02. int M_SHIFT = 0;
03. public int FNVHash(byte[] data)
04. {
05.     int hash = (int)2166136261L;
06.     for(byte b : data)
07.         hash = (hash * 16777619) ^ b;
08.     if (M_SHIFT == 0)
09.         return hash;
10.     return (hash ^ (hash >> M_SHIFT)) & M_MASK;
11. }
```

以及改进的FNV算法：

```
[java]
```

```
01. public static int FNVHash1(String data)
02. {
03.     final int p = 16777619;
04.     int hash = (int)2166136261L;
05.     for(int i=0;i<data.length();i++)
06.         hash = (hash ^ data.charAt(i)) * p;
07.     hash += hash << 13;
08.     hash ^= hash >> 7;
09.     hash += hash << 3;
10.     hash ^= hash >> 17;
11.     hash += hash << 5;
12.     return hash;
13. }
```

除了乘以一个固定的数，常见的还有乘以一个不断改变的数，比如：

```
[java]
01. static int RSHash(String str)
02. {
03.     int b = 378551;
04.     int a = 63689;
05.     int hash = 0;
06.
07.     for(int i = 0; i < str.length(); i++)
08.     {
09.         hash = hash * a + str.charAt(i);
10.         a = a * b;
11.     }
12.     return (hash & 0x7FFFFFFF);
```

虽然Adler32算法的应用没有CRC32广泛，不过，它可能是乘法Hash里面最有名的一个了。关于它的介绍，大家可以去看RFC 1950规范。

四 除法Hash

除法和乘法一样，同样具有表面上看起来的不相关性。不过，因为除法太慢，这种方式几乎找不到真正的应用。需要注意的是，我们在前面看到的hash的结果除以一个prime的目的只是为了保证结果的范围。如果你不需要它限制一个范围的话，可以使用如下的代码替代“hash%prime”： $hash = hash \wedge (hash \gg 10) \wedge (hash \gg 20)$ 。

五 查表Hash

查表Hash最有名的例子莫过于CRC系列算法。虽然CRC系列算法本身并不是查表，但是，查表是它的一种最快的实现方式。下面是CRC32的实现：

```
[java]
01. static int crctab[256] = {
02.     0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f, 0xe963a535, 0x9e6495
03.     0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0, 0x10da7a5a, 0x67dd4acc, 0xf9b9df6f, 0x8ebe
04. };
05. int crc32(String key, int hash)
06. {
07.     int i;
08.     for (hash=key.length(), i=0; i<key.length(); ++i)
09.         hash = (hash >> 8) ^ crctab[(hash & 0xff) ^ k.charAt(i)];
10.     return hash;
11. }
```

查表Hash中有名的例子有：Universal Hashing和Zobrist Hashing。他们的表格都是随机生成的。

六 混合Hash

混合Hash算法利用了以上各种方式。各种常见的Hash算法，比如MD5、Tiger都属于这个范围。它们一般很少在面向查找的Hash函数里面使用。

七 对Hash算法的评价

<http://www.burtleburtle.net/bob/hash/doobs.html> 这个页面提供了对几种流行Hash算法的评价。我们对Hash函数的建议如下：



微信关注CSDN
获得无限技术资源

快速回复

我要收藏

返回顶部

1. 字符串的Hash。最简单可以使用基本的乘法Hash，当乘数为33时，对于英文单词有很好的散列效果（小于6个的小写形式可以保证没有冲突）。复杂一点可以使用FNV算法（及其改进形式），它对于比较长的字符串，在速度和效果上都不错。
2. 长数组的Hash。可以使用<http://burtleburtle.net/bob/c/lookup3.c>这种算法，它一次运算多个字节，速度还算不错。

八后记

本文简略的介绍了一番实际应用中的用于查找的Hash算法。Hash算法除了应用于这个方面以外，^{另外一个著名}的应用是巨型字符串匹配（这时的Hash算法叫做：rolling hash，因为它必须可以滚动的计算）。好的Hash算法并不是一件容易的事情。做为应用来说，选择一个适合的算法是最重要的。

常用hash算法类：

```
[java]
01. package lotusroots.algorithms.math;
02.
03. import java.security.MessageDigest;
04.
05. /**
06.  * Hash算法大全<br>
07.  * 推荐使用FNV1算法
08.  *
09.  * @algorithm None
10.  * @author Goodzzp 2006-11-20
11.  * @lastEdit Goodzzp 2006-11-20
12.  * @editDetail Create
13.  */
14. public class HashAlgorithms {
15.     /**
16.      * 加法hash
17.      *
18.      * @param key
19.      *      字符串
20.      * @param prime
21.      *      一个质数
22.      * @return hash结果
23.      */
24.     public static int additiveHash(String key, int prime) {
25.         int hash, i;
26.         for (hash = key.length(), i = 0; i < key.length(); i++)
27.             hash += key.charAt(i);
28.         return (hash % prime);
29.     }
30.
31.     /**
32.      * 旋转hash
33.      *
34.      * @param key
35.      *      输入字符串
36.      * @param prime
37.      *      质数
38.      * @return hash值
39.      */
40.     public static int rotatingHash(String key, int prime) {
41.         int hash, i;
42.         for (hash = key.length(), i = 0; i < key.length(); ++i)
43.             hash = (hash << 4) ^ (hash >> 28) ^ key.charAt(i);
44.         return (hash % prime);
45.         // return (hash ^ (hash>>10) ^ (hash>>20));
46.     }
47.
48.     // 替代:
49.     // 使用: hash = (hash ^ (hash>>10) ^ (hash>>20)) & mask;
50.     // 替代: hash %= prime;
51.
52.     /**
53.      * MASK值，随便找一个值，最好是质数
54.      */
55.     static int M_MASK = 0x8765fed1;
56.
57.     /**
58.      * 一次一个hash
59.      */

```



```

60.  * @param key
61.  *      输入字符串
62.  * @return 输出hash值
63.  */
64. public static int oneByOneHash(String key) {
65.     int hash, i;
66.     for (hash = 0, i = 0; i < key.length(); ++i) {
67.         hash += key.charAt(i);
68.         hash += (hash << 10);
69.         hash ^= (hash >> 6);
70.     }
71.     hash += (hash << 3);
72.     hash ^= (hash >> 11);
73.     hash += (hash << 15);
74.     // return (hash & M_MASK);
75.     return hash;
76. }
77.
78. /**
79.  * Bernstein's hash
80.  *
81.  * @param key
82.  *      输入字节数组
83.  * @param level
84.  *      初始hash常量
85.  * @return 结果hash
86.  */
87. public static int bernstein(String key) {
88.     int hash = 0;
89.     int i;
90.     for (i = 0; i < key.length(); ++i)
91.         hash = 33 * hash + key.charAt(i);
92.     return hash;
93. }
94.
95. //
96. // // Pearson's Hash
97. // char pearson(char[]key, ub4 len, char tab[256])
98. // {
99. //     char hash;
100. //     ub4 i;
101. //     for (hash=len, i=0; i<len; ++i)
102. //         hash=tab[hash^key[i]];
103. //     return (hash);
104. // }
105.
106. // // CRC Hashing, 计算crc,具体代码见其他
107. // ub4 crc(char *key, ub4 len, ub4 mask, ub4 tab[256])
108. // {
109. //     ub4 hash, i;
110. //     for (hash=len, i=0; i<len; ++i)
111. //         hash = (hash >> 8) ^ tab[(hash & 0xff) ^ key[i]];
112. //     return (hash & mask);
113. // }
114.
115. /**
116.  * Universal Hashing
117.  */
118. public static int universal(char[] key, int mask, int[] tab) {
119.     int hash = key.length, i, len = key.length;
120.     for (i = 0; i < (len << 3); i += 8) {
121.         char k = key[i >> 3];
122.         if ((k & 0x01) == 0)
123.             hash ^= tab[i + 0];
124.         if ((k & 0x02) == 0)
125.             hash ^= tab[i + 1];
126.         if ((k & 0x04) == 0)
127.             hash ^= tab[i + 2];
128.         if ((k & 0x08) == 0)
129.             hash ^= tab[i + 3];
130.         if ((k & 0x10) == 0)
131.             hash ^= tab[i + 4];
132.         if ((k & 0x20) == 0)
133.             hash ^= tab[i + 5];
134.         if ((k & 0x40) == 0)
135.             hash ^= tab[i + 6];
136.         if ((k & 0x80) == 0)
137.             hash ^= tab[i + 7];
138.     }

```



微信关注CSDN
获得无限技术资源

快速回复

我要收藏

返回顶部

```
139.     return (hash & mask);
140. }
141.
142. /**
143.  * Zobrist Hashing
144.  */
145. public static int zobrist(char[] key, int mask, int[][] tab) {
146.     int hash, i;
147.     for (hash = key.length, i = 0; i < key.length; ++i)
148.         hash ^= tab[i][key[i]];
149.     return (hash & mask);
150. }
151.
152. // LOOKUP3
153. // 见Bob Jenkins(3).c文件
154.
155. // 32位FNV算法
156. static int M_SHIFT = 0;
157.
158. /**
159.  * 32位的FNV算法
160.  *
161.  * @param data
162.  *      数组
163.  * @return int值
164.  */
165. public static int FNVHash(byte[] data) {
166.     int hash = (int) 2166136261L;
167.     for (byte b : data)
168.         hash = (hash * 16777619) ^ b;
169.     if (M_SHIFT == 0)
170.         return hash;
171.     return (hash ^ (hash >> M_SHIFT)) & M_MASK;
172. }
173.
174. /**
175.  * 改进的32位FNV算法1
176.  *
177.  * @param data
178.  *      数组
179.  * @return int值
180.  */
181. public static int FNVHash1(byte[] data) {
182.     final int p = 16777619;
183.     int hash = (int) 2166136261L;
184.     for (byte b : data)
185.         hash = (hash ^ b) * p;
186.     hash += hash << 13;
187.     hash ^= hash >> 7;
188.     hash += hash << 3;
189.     hash ^= hash >> 17;
190.     hash += hash << 5;
191.     return hash;
192. }
193.
194. /**
195.  * 改进的32位FNV算法1
196.  *
197.  * @param data
198.  *      字符串
199.  * @return int值
200.  */
201. public static int FNVHash1(String data) {
202.     final int p = 16777619;
203.     int hash = (int) 2166136261L;
204.     for (int i = 0; i < data.length(); i++)
205.         hash = (hash ^ data.charAt(i)) * p;
206.     hash += hash << 13;
207.     hash ^= hash >> 7;
208.     hash += hash << 3;
209.     hash ^= hash >> 17;
210.     hash += hash << 5;
211.     return hash;
212. }
213.
214. /**
215.  * Thomas Wang的算法, 整数hash
216.  */
217. public static int intHash(int key) {
```



微信关注CSDN
获得无限技术资源

快速回复

我要收藏

返回顶部

```

218.     key += ~(key << 15);
219.     key ^= (key >>> 10);
220.     key += (key << 3);
221.     key ^= (key >>> 6);
222.     key += ~(key << 11);
223.     key ^= (key >>> 16);
224.     return key;
225. }
226.
227. /**
228.  * RS算法hash
229.  */
230. public static int RSHash(String str) {
231.     int b = 378551;
232.     int a = 63689;
233.     int hash = 0;
234.
235.     for (int i = 0; i < str.length(); i++) {
236.         hash = hash * a + str.charAt(i);
237.         a = a * b;
238.     }
239.
240.     return (hash & 0x7FFFFFFF);
241. }
242.
243. /* End Of RS Hash Function */
244.
245. /**
246.  * JS算法
247.  */
248. public static int JSHash(String str) {
249.     int hash = 1315423911;
250.
251.     for (int i = 0; i < str.length(); i++) {
252.         hash ^= ((hash << 5) + str.charAt(i) + (hash >> 2));
253.     }
254.
255.     return (hash & 0x7FFFFFFF);
256. }
257.
258. /* End Of JS Hash Function */
259.
260. /**
261.  * PJW算法
262.  */
263. public static int PJWHash(String str) {
264.     int BitsInUnsignedInt = 32;
265.     int ThreeQuarters = (BitsInUnsignedInt * 3) / 4;
266.     int OneEighth = BitsInUnsignedInt / 8;
267.     int HighBits = 0xFFFFFFFF << (BitsInUnsignedInt - OneEighth);
268.     int hash = 0;
269.     int test = 0;
270.
271.     for (int i = 0; i < str.length(); i++) {
272.         hash = (hash << OneEighth) + str.charAt(i);
273.
274.         if ((test = hash & HighBits) != 0) {
275.             hash = ((hash ^ (test >> ThreeQuarters)) & (~HighBits));
276.         }
277.     }
278.
279.     return (hash & 0x7FFFFFFF);
280. }
281.
282. /* End Of P. J. Weinberger Hash Function */
283.
284. /**
285.  * ELF算法
286.  */
287. public static int ELFHash(String str) {
288.     int hash = 0;
289.     int x = 0;
290.
291.     for (int i = 0; i < str.length(); i++) {
292.         hash = (hash << 4) + str.charAt(i);
293.         if ((x = (int) (hash & 0xF0000000L)) != 0) {
294.             hash ^= (x >> 24);
295.             hash &= ~x;
296.         }

```



微信关注CSDN
获得无限技术资源

快速回复

我要收藏

返回顶部

```
297.     }
298.
299.     return (hash & 0x7FFFFFFF);
300. }
301.
302. /* End Of ELF Hash Function */
303.
304. /**
305.  * BKDR算法
306.  */
307. public static int BKDRHash(String str) {
308.     int seed = 131; // 31 131 1313 13131 131313 etc..
309.     int hash = 0;
310.
311.     for (int i = 0; i < str.length(); i++) {
312.         hash = (hash * seed) + str.charAt(i);
313.     }
314.
315.     return (hash & 0x7FFFFFFF);
316. }
317.
318. /* End Of BKDR Hash Function */
319.
320. /**
321.  * SDBM算法
322.  */
323. public static int SDBMHash(String str) {
324.     int hash = 0;
325.
326.     for (int i = 0; i < str.length(); i++) {
327.         hash = str.charAt(i) + (hash << 6) + (hash << 16) - hash;
328.     }
329.
330.     return (hash & 0x7FFFFFFF);
331. }
332.
333. /* End Of SDBM Hash Function */
334.
335. /**
336.  * DJB算法
337.  */
338. public static int DJBHash(String str) {
339.     int hash = 5381;
340.
341.     for (int i = 0; i < str.length(); i++) {
342.         hash = ((hash << 5) + hash) + str.charAt(i);
343.     }
344.
345.     return (hash & 0x7FFFFFFF);
346. }
347.
348. /* End Of DJB Hash Function */
349.
350. /**
351.  * DEK算法
352.  */
353. public static int DEKHash(String str) {
354.     int hash = str.length();
355.
356.     for (int i = 0; i < str.length(); i++) {
357.         hash = ((hash << 5) ^ (hash >> 27)) ^ str.charAt(i);
358.     }
359.
360.     return (hash & 0x7FFFFFFF);
361. }
362.
363. /* End Of DEK Hash Function */
364.
365. /**
366.  * AP算法
367.  */
368. public static int APHash(String str) {
369.     int hash = 0;
370.
371.     for (int i = 0; i < str.length(); i++) {
372.         hash ^= ((i & 1) == 0) ? ((hash << 7) ^ str.charAt(i) ^ (hash >> 3)) : (~
            ((hash << 11) ^ str.charAt(i) ^ (hash >> 5)));
373.     }
374. }
```



微信关注CSDN
获得无限技术资源

快速回复

我要收藏

返回顶部


```
375. // return (hash & 0x7FFFFFFF);
376. return hash;
377. }
378.
379. /* End Of AP Hash Function */
380.
381. /**
382.  * JAVA自己带的算法
383.  */
384. public static int java(String str) {
385.     int h = 0;
386.     int off = 0;
387.     int len = str.length();
388.     for (int i = 0; i < len; i++) {
389.         h = 31 * h + str.charAt(off++);
390.     }
391.     return h;
392. }
393.
394. /**
395.  * 混合hash算法，输出64位的值
396.  */
397. public static long mixHash(String str) {
398.     long hash = str.hashCode();
399.     hash <<= 32;
400.     hash |= FNVHash1(str);
401.     return hash;
402. }
403.
404. /**
405.  * 计算sha1
406.  *
407.  * @param text
408.  *         文本
409.  * @return 字节数组
410.  * @throws Exception
411.  */
412. public static byte[] sha1(String text) throws Exception {
413.     MessageDigest md;
414.     md = MessageDigest.getInstance("SHA-1");
415.     byte[] sha1hash = new byte[40];
416.     byte[] input = text.getBytes("utf-8");
417.     md.update(input, 0, input.length);
418.     sha1hash = md.digest();
419.     return sha1hash;
420. }
421.
422. // 4位值对应16进制字符
423. static char[] m_byteToHexChar = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b',
424.
425. /**
426.  * 计算sha1
427.  *
428.  * @param text
429.  *         文本
430.  * @return 16进制表示的hash值
431.  * @throws Exception
432.  */
433. public static String sha1_text(String text) throws Exception {
434.     byte[] hash = sha1(text);
435.     StringBuilder ret = new StringBuilder(hash.length * 2);
436.     for (byte b : hash) {
437.         int d = (b & 0xff);
438.         ret.append(m_byteToHexChar[(d & 0xf)]);
439.         d >>= 4;
440.         ret.append(m_byteToHexChar[(d & 0xf)]);
441.     }
442.     return ret.toString();
443. }
444. }
```



微信关注CSDN
获得无限技术资源

快速回复

我要收藏

返回顶部

分享到：

[Linux 常用命令使用方法大搜刮上一篇：](#)
[用java实现生产者和消费者问题下一篇：](#)

查看评论



微信关注CSDN
获取更多技术干货





快速回复



我要收藏



返回顶部

顶

0

踩

0

上一篇

Android学习

下一篇

静态 与动态（小探究）

相关文章推荐

- 用java实现一个哈希表类
- 用户画像系统应用与技术解析--汪剑
- Java密码学原型算法实现——第一部分：标准Has...
- 实时流计算平台Blink在阿里集团的应用实践--陈守元
- Java中String的hash函数分析
- Java 9新特性解读
- 哈希表、Java中HashMap
- Cocos2d-x 实战演练基础篇

- 应用Hash函数（java描述）
- Unity3D移动端实战经验分享
- 全面解析hash函数的各种应用(持续更新)
- 程序员如何转型AI工程师--蒋涛
- 应用Hash函数
- Hash在Java中的应用
- Java中String的hash函数分析
- js 利用iframe和location.hash跨域解决办法，java...

查看评论

暂无评论

您还没有登录,请[\[登录\]](#)或[\[注册\]](#)

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场