**大量Hash算法的实现**

[小算法](http://xdeduzb.blog.163.com/blog/#m=0&t=1&c=fks_087068081080083071093087094095092087088074087080085068) 2010-08-02 15:28:19 阅读10 评论0 字号：大中小

|  |
| --- |
| Hash算法有很多很多种类。具体的可以参考之前我写的Hash算法的一些分析。本处给大家提供一个集合了很多使用的Hash算法的类，应该可以满足不少人的需要的：  /\*\* \* Hash算法大全<br> \* 推荐使用FNV1算法 \* @algorithm None \* @author Goodzzp 2006-11-20 \* @lastEdit Goodzzp 2006-11-20  \* @editDetail Create \*/ public class HashAlgorithms { /\*\* \* 加法hash \* @param key 字符串 \* @param prime 一个质数 \* @return hash结果 \*/ public static int additiveHash(String key, int prime) {    int hash, i;    for (hash = key.length(), i = 0; i < key.length(); i++)     hash += key.charAt(i);    return (hash % prime); } /\*\* \* 旋转hash \* @param key 输入字符串 \* @param prime 质数 \* @return hash值 \*/ public static int rotatingHash(String key, int prime) {    int hash, i;    for (hash=key.length(), i=0; i<key.length(); ++i)      hash = (hash<<4)^(hash>>28)^key.charAt(i);    return (hash % prime); //   return (hash ^ (hash>>10) ^ (hash>>20)); }  // 替代： // 使用：hash = (hash ^ (hash>>10) ^ (hash>>20)) & mask; // 替代：hash %= prime; /\*\* \* MASK值，随便找一个值，最好是质数 \*/ static int M\_MASK = 0x8765fed1; /\*\* \* 一次一个hash \* @param key 输入字符串 \* @return 输出hash值 \*/ public static int oneByOneHash(String key) {    int   hash, i;    for (hash=0, i=0; i<key.length(); ++i)    {      hash += key.charAt(i);      hash += (hash << 10);      hash ^= (hash >> 6);    }    hash += (hash << 3);    hash ^= (hash >> 11);    hash += (hash << 15); //   return (hash & M\_MASK);    return hash; }  /\*\* \* Bernstein's hash \* @param key 输入字节数组 \* @param level 初始hash常量 \* @return 结果hash \*/ public static int bernstein(String key) {    int hash = 0;    int i;    for (i=0; i<key.length(); ++i) hash = 33\*hash + key.charAt(i);    return hash; }  // //// Pearson's Hash // char pearson(char[]key, ub4 len, char tab[256]) // { //   char hash; //   ub4 i; //   for (hash=len, i=0; i<len; ++i)  //     hash=tab[hash^key[i]]; //   return (hash); // }  //// CRC Hashing，计算crc,具体代码见其他 // ub4 crc(char \*key, ub4 len, ub4 mask, ub4 tab[256]) // { //   ub4 hash, i; //   for (hash=len, i=0; i<len; ++i) //     hash = (hash >> 8) ^ tab[(hash & 0xff) ^ key[i]]; //   return (hash & mask); // }  /\*\* \* Universal Hashing \*/ public static int universal(char[]key, int mask, int[] tab) {    int hash = key.length, i, len = key.length;    for (i=0; i<(len<<3); i+=8)    {      char k = key[i>>3];      if ((k&0x01) == 0) hash ^= tab[i+0];      if ((k&0x02) == 0) hash ^= tab[i+1];      if ((k&0x04) == 0) hash ^= tab[i+2];      if ((k&0x08) == 0) hash ^= tab[i+3];      if ((k&0x10) == 0) hash ^= tab[i+4];      if ((k&0x20) == 0) hash ^= tab[i+5];      if ((k&0x40) == 0) hash ^= tab[i+6];      if ((k&0x80) == 0) hash ^= tab[i+7];    }    return (hash & mask); }  /\*\* \* Zobrist Hashing \*/  public static int zobrist( char[] key,int mask, int[][] tab) {    int hash, i;    for (hash=key.length, i=0; i<key.length; ++i)      hash ^= tab[i][key[i]];    return (hash & mask); }  // LOOKUP3  // 见Bob Jenkins(3).c文件  // 32位FNV算法 static int M\_SHIFT = 0; /\*\* \* 32位的FNV算法 \* @param data 数组 \* @return int值 \*/     public static int FNVHash(byte[] data)     {         int hash = (int)2166136261L;         for(byte b : data)             hash = (hash \* 16777619) ^ b;         if (M\_SHIFT == 0)             return hash;         return (hash ^ (hash >> M\_SHIFT)) & M\_MASK;     }     /\*\*      \* 改进的32位FNV算法1      \* @param data 数组      \* @return int值      \*/     public static int FNVHash1(byte[] data)     {         final int p = 16777619;         int hash = (int)2166136261L;         for(byte b:data)             hash = (hash ^ b) \* p;         hash += hash << 13;         hash ^= hash >> 7;         hash += hash << 3;         hash ^= hash >> 17;         hash += hash << 5;         return hash;     }     /\*\*      \* 改进的32位FNV算法1      \* @param data 字符串      \* @return int值      \*/     public static int FNVHash1(String data)     {         final int p = 16777619;         int hash = (int)2166136261L;         for(int i=0;i<data.length();i++)             hash = (hash ^ data.charAt(i)) \* p;         hash += hash << 13;         hash ^= hash >> 7;         hash += hash << 3;         hash ^= hash >> 17;         hash += hash << 5;         return hash;     }      /\*\*      \* Thomas Wang的算法，整数hash      \*/      public static int intHash(int key)     {       key += ~(key << 15);       key ^= (key >>> 10);       key += (key << 3);       key ^= (key >>> 6);       key += ~(key << 11);       key ^= (key >>> 16);       return key;     }     /\*\*      \* RS算法hash      \* @param str 字符串      \*/     public static int RSHash(String str)     {         int b    = 378551;         int a    = 63689;         int hash = 0;         for(int i = 0; i < str.length(); i++)        {           hash = hash \* a + str.charAt(i);           a    = a \* b;        }         return (hash & 0x7FFFFFFF);     }     /\* End Of RS Hash Function \*/      /\*\*      \* JS算法      \*/     public static int JSHash(String str)     {        int hash = 1315423911;         for(int i = 0; i < str.length(); i++)        {           hash ^= ((hash << 5) + str.charAt(i) + (hash >> 2));        }         return (hash & 0x7FFFFFFF);     }     /\* End Of JS Hash Function \*/      /\*\*      \* PJW算法      \*/     public static int PJWHash(String str)     {         int BitsInUnsignedInt = 32;         int ThreeQuarters     = (BitsInUnsignedInt \* 3) / 4;         int OneEighth         = BitsInUnsignedInt / 8;         int HighBits          = 0xFFFFFFFF << (BitsInUnsignedInt - OneEighth);         int hash              = 0;         int test              = 0;         for(int i = 0; i < str.length();i++)        {           hash = (hash << OneEighth) + str.charAt(i);            if((test = hash & HighBits) != 0)           {              hash = (( hash ^ (test >> ThreeQuarters)) & (~HighBits));           }        }         return (hash & 0x7FFFFFFF);     }     /\* End Of P. J. Weinberger Hash Function \*/      /\*\*      \* ELF算法      \*/     public static int ELFHash(String str)     {         int hash = 0;         int x    = 0;         for(int i = 0; i < str.length(); i++)        {           hash = (hash << 4) + str.charAt(i);           if((x = (int)(hash & 0xF0000000L)) != 0)           {              hash ^= (x >> 24);              hash &= ~x;           }        }         return (hash & 0x7FFFFFFF);     }     /\* End Of ELF Hash Function \*/      /\*\*      \* BKDR算法      \*/     public static int BKDRHash(String str)     {         int seed = 131; // 31 131 1313 13131 131313 etc..         int hash = 0;         for(int i = 0; i < str.length(); i++)        {           hash = (hash \* seed) + str.charAt(i);        }         return (hash & 0x7FFFFFFF);     }     /\* End Of BKDR Hash Function \*/      /\*\*      \* SDBM算法      \*/     public static int SDBMHash(String str)     {         int hash = 0;         for(int i = 0; i < str.length(); i++)        {           hash = str.charAt(i) + (hash << 6) + (hash << 16) - hash;        }         return (hash & 0x7FFFFFFF);     }     /\* End Of SDBM Hash Function \*/      /\*\*      \* DJB算法      \*/     public static int DJBHash(String str)     {        int hash = 5381;         for(int i = 0; i < str.length(); i++)        {           hash = ((hash << 5) + hash) + str.charAt(i);        }         return (hash & 0x7FFFFFFF);     }     /\* End Of DJB Hash Function \*/      /\*\*      \* DEK算法      \*/     public static int DEKHash(String str)     {         int hash = str.length();         for(int i = 0; i < str.length(); i++)        {           hash = ((hash << 5) ^ (hash >> 27)) ^ str.charAt(i);        }         return (hash & 0x7FFFFFFF);     }     /\* End Of DEK Hash Function \*/      /\*\*      \* AP算法      \*/     public static int APHash(String str)     {         int hash = 0;         for(int i = 0; i < str.length(); i++)        {           hash ^= ((i & 1) == 0) ? ( (hash << 7) ^ str.charAt(i) ^ (hash >> 3)) :                                    (~((hash << 11) ^ str.charAt(i) ^ (hash >> 5)));        }  //       return (hash & 0x7FFFFFFF);        return hash;     }     /\* End Of AP Hash Function \*/          /\*\*      \* JAVA自己带的算法      \*/     public static int java(String str) {    int h = 0;    int off = 0;    int len = str.length();    for (int i = 0; i < len; i++)    {     h = 31 \* h + str.charAt(off++);    }    return h; }          /\*\*      \* 混合hash算法，输出64位的值      \*/     public static long mixHash(String str)     {     long hash = str.hashCode();     hash <<= 32;     hash |= FNVHash1(str);     return hash;     } } |