# C#、Java和JS实现SHA256+BASE64加密总结

--莫非(www.muphy.me)

## 原理

首先，通过编码格式（UTF-8、ASCII等，如果含有汉字等字符，编码格式不同加密结果也不同）获得加密字符串的字节数组，再计算字节数组的哈希值数组，再把hash数组转换为base64编码的字符串。

## C#实例

**class** Program

{//转换为base64编码为44位

**public** **static** string NewSHA256(string str)

{

//如果str有中文，不同Encoding的sha是不同的！！

using (SHA256Managed sha256 = **new** SHA256Managed())

{

byte[] SHA256Data = Encoding.UTF8.GetBytes(str);

// SHA256Data "104,101,108,108,111,32,119,111,114,108,100"

byte[] by = sha256.ComputeHash(SHA256Data);

//44位 "uU0nuZNNPgilLlLX2n2r+sSE7+N6U4DukIj3rOLvzek="

**return** Convert.ToBase64String(by);

}

}

//转换为十六进制编码为64位 去除了‘-’

**public** **static** string SHA256(string str)

{ //如果str有中文，不同Encoding的sha是不同的！！

using (SHA256Managed sha256 = **new** SHA256Managed())

{

byte[] SHA256Data = Encoding.UTF8.GetBytes(str);

// SHA256Data "104,101,108,108,111,32,119,111,114,108,100"

byte[] by = sha256.ComputeHash(SHA256Data);

//64位 "b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9"

**return** BitConverter.ToString(by).Replace("-", "").ToLower();

}

}

**static** void Main(string[] args)

{

string s = "hello world";

//sha265：b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9

string sha = SHA256(s);

Console.WriteLine("{0}\n{1} \nLength: {2}", s, sha, sha.Length);

//结果 ”uU0nuZNNPgilLlLX2n2r+sSE7+N6U4DukIj3rOLvzek=”

}

}

## Java实例

/\*\*

\* 利用java原生的摘要实现SHA256加密

\* **@param** str 加密后的报文

\* **@return**

\*/

**public** **static** String getSHA256StrJava(String str){

MessageDigest messageDigest;

String encodeStr = "";

**try** {

messageDigest = MessageDigest.getInstance("SHA-256");

messageDigest.update(str.getBytes("UTF-8"));

//转换的方式选择

//encodeStr = byte2Hex(messageDigest.digest());//转换成二进制

encodeStr = base64Entrypt(messageDigest.digest());//转换成Base64

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (UnsupportedEncodingException e) {

e.printStackTrace();

}

**return** encodeStr;

}

/\*\*

\* 将byte转为16进制

\* **@param** bytes

\* **@return**

\*/

**private** **static** String byte2Hex(byte[] bytes){

StringBuffer stringBuffer = **new** StringBuffer();

String temp = **null**;

**for** (int i=0;i<bytes.length;i++){

temp = Integer.toHexString(bytes[i] & 0xFF);

**if** (temp.length()==1){

//1得到一位的进行补0操作

stringBuffer.append("0");

}

stringBuffer.append(temp);

}

**return** stringBuffer.toString();

}

/\*\*

\* 转换为BASE64编码

\* **@param** bytes

\* **@return**

\*/

**private** **static** String base64Entrypt(byte[] bytes){

//import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;

**return** Base64.encode(bytes);

}

**private** **static** String newBase64Entrypt(byte[] bytes){

//import sun.misc.BASE64Encoder;

**return** **new** BASE64Encoder().encode(bytes);

}

/\*\*

\* 测试

\* **@param** Args

\*/

**public** **static** void main(String[] Args){

String str = "hello world";

String sha = getSHA256StrJava(str);

System.out.println(str + ":" + sha);

//结果：hello world:uU0nuZNNPgilLlLX2n2r+sSE7+N6U4DukIj3rOLvzek=

}

## JS实例

/\*\*

\* SHA256加密

\*/

**function** SHA256(s) {

**var** chrsz = 8;

**var** hexcase = 0;

**function** safe\_add(x, y) {

**var** lsw = (x & 0xFFFF) + (y & 0xFFFF);

**var** msw = (x >> 16) + (y >> 16) + (lsw >> 16);

**return** (msw << 16) | (lsw & 0xFFFF);

}

**function** S(X, n) {

**return** (X >>> n) | (X << (32 - n));

}

**function** R(X, n) {

**return** (X >>> n);

}

**function** Ch(x, y, z) {

**return** ((x & y) ^ ((~x) & z));

}

**function** Maj(x, y, z) {

**return** ((x & y) ^ (x & z) ^ (y & z));

}

**function** Sigma0256(x) {

**return** (S(x, 2) ^ S(x, 13) ^ S(x, 22));

}

**function** Sigma1256(x) {

**return** (S(x, 6) ^ S(x, 11) ^ S(x, 25));

}

**function** Gamma0256(x) {

**return** (S(x, 7) ^ S(x, 18) ^ R(x, 3));

}

**function** Gamma1256(x) {

**return** (S(x, 17) ^ S(x, 19) ^ R(x, 10));

}

**function** core\_sha256(m, l) {

**var** K = **new** Array(0x428A2F98, 0x71374491, 0xB5C0FBCF, 0xE9B5DBA5,

0x3956C25B, 0x59F111F1, 0x923F82A4, 0xAB1C5ED5, 0xD807AA98,

0x12835B01, 0x243185BE, 0x550C7DC3, 0x72BE5D74, 0x80DEB1FE,

0x9BDC06A7, 0xC19BF174, 0xE49B69C1, 0xEFBE4786, 0xFC19DC6,

0x240CA1CC, 0x2DE92C6F, 0x4A7484AA, 0x5CB0A9DC, 0x76F988DA,

0x983E5152, 0xA831C66D, 0xB00327C8, 0xBF597FC7, 0xC6E00BF3,

0xD5A79147, 0x6CA6351, 0x14292967, 0x27B70A85, 0x2E1B2138,

0x4D2C6DFC, 0x53380D13, 0x650A7354, 0x766A0ABB, 0x81C2C92E,

0x92722C85, 0xA2BFE8A1, 0xA81A664B, 0xC24B8B70, 0xC76C51A3,

0xD192E819, 0xD6990624, 0xF40E3585, 0x106AA070, 0x19A4C116,

0x1E376C08, 0x2748774C, 0x34B0BCB5, 0x391C0CB3, 0x4ED8AA4A,

0x5B9CCA4F, 0x682E6FF3, 0x748F82EE, 0x78A5636F, 0x84C87814,

0x8CC70208, 0x90BEFFFA, 0xA4506CEB, 0xBEF9A3F7, 0xC67178F2);

**var** HASH = **new** Array(0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A,

0x510E527F, 0x9B05688C, 0x1F83D9AB, 0x5BE0CD19);

**var** W = **new** Array(64);

**var** a, b, c, d, e, f, g, h, i, j;

**var** T1, T2;

m[l >> 5] |= 0x80 << (24 - l % 32);

m[((l + 64 >> 9) << 4) + 15] = l;

**for** (**var** i = 0; i < m.length; i += 16) {

a = HASH[0];

b = HASH[1];

c = HASH[2];

d = HASH[3];

e = HASH[4];

f = HASH[5];

g = HASH[6];

h = HASH[7];

**for** (**var** j = 0; j < 64; j++) {

**if** (j < 16)

W[j] = m[j + i];

**else**

W[j] = safe\_add(safe\_add(safe\_add(Gamma1256(W[j - 2]),

W[j - 7]), Gamma0256(W[j - 15])), W[j - 16]);

T1 = safe\_add(safe\_add(safe\_add(safe\_add(h, Sigma1256(e)), Ch(

e, f, g)), K[j]), W[j]);

T2 = safe\_add(Sigma0256(a), Maj(a, b, c));

h = g;

g = f;

f = e;

e = safe\_add(d, T1);

d = c;

c = b;

b = a;

a = safe\_add(T1, T2);

}

HASH[0] = safe\_add(a, HASH[0]);

HASH[1] = safe\_add(b, HASH[1]);

HASH[2] = safe\_add(c, HASH[2]);

HASH[3] = safe\_add(d, HASH[3]);

HASH[4] = safe\_add(e, HASH[4]);

HASH[5] = safe\_add(f, HASH[5]);

HASH[6] = safe\_add(g, HASH[6]);

HASH[7] = safe\_add(h, HASH[7]);

}

**return** HASH;

}

**function** str2binb(str) {

**var** bin = Array();

**var** mask = (1 << chrsz) - 1;

**for** (**var** i = 0; i < str.length \* chrsz; i += chrsz) {

bin[i >> 5] |= (str.charCodeAt(i / chrsz) & mask) << (24 - i % 32);

}

**return** bin;

}

**function** Utf8Encode(string) {

string = string.replace(/\r\n/g, "\n");

**var** utftext = "";

**for** (**var** n = 0; n < string.length; n++) {

**var** c = string.charCodeAt(n);

**if** (c < 128) {

utftext += String.fromCharCode(c);

} **else** **if** ((c > 127) && (c < 2048)) {

utftext += String.fromCharCode((c >> 6) | 192);

utftext += String.fromCharCode((c & 63) | 128);

} **else** {

utftext += String.fromCharCode((c >> 12) | 224);

utftext += String.fromCharCode(((c >> 6) & 63) | 128);

utftext += String.fromCharCode((c & 63) | 128);

}

}

**return** utftext;

}

**function** binb2hex(binarray) {

**var** hex\_tab = hexcase ? "0123456789ABCDEF" : "0123456789abcdef";

**var** str = "";

**for** (**var** i = 0; i < binarray.length \* 4; i++) {

str += hex\_tab

.charAt((binarray[i >> 2] >> ((3 - i % 4) \* 8 + 4)) & 0xF)

+ hex\_tab.charAt((binarray[i >> 2] >> ((3 - i % 4) \* 8)) & 0xF);

}

**return** str;

}

s = Utf8Encode(s);

**return** binb2hex(core\_sha256(str2binb(s), s.length \* chrsz));

}