[**关于base64编码的原理及实现**](http://www.cnblogs.com/hongru/archive/2012/01/14/2321397.html)

我们的图片大部分都是可以转换成base64编码的data：image。 这个在将canvas保存为img的时候尤其有用。虽然除ie外，大部分现代浏览器都已经支持原生的基于base64的encode和decode，例如btoa和atob。（将canvas画布保存成img并强制改变mimetype进行下载，会在下一篇记录）

但是处于好奇心，还是驱使我去了解下base64编码的原理。以便也在不支持原生base64编码的ie下可以得以实现。

【Base64】  
-base64的编码都是按字符串长度，以每3个8bit的字符为一组，  
-然后针对每组，首先获取每个字符的ASCII编码，  
-然后将ASCII编码转换成8bit的二进制，得到一组3\*8=24bit的字节  
-然后再将这24bit划分为4个6bit的字节，并在每个6bit的字节前面都填两个高位0，得到4个8bit的字节  
-然后将这4个8bit的字节转换成10进制，对照Base64编码表 （下表），得到对应编码后的字符。

（注：1. 要求被编码字符是8bit的，所以须在ASCII编码范围内，\u0000-\u00ff，中文就不行。  
　　　2. 如果被编码字符长度不是3的倍数的时候，则都用0代替，对应的输出字符为=）

| **Base64 编码表** | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Value** | **Char** |  | **Value** | **Char** |  | **Value** | **Char** |  | **Value** | **Char** |
| 0 | A | 16 | Q | 32 | g | 48 | w |
| 1 | B | 17 | R | 33 | h | 49 | x |
| 2 | C | 18 | S | 34 | i | 50 | y |
| 3 | D | 19 | T | 35 | j | 51 | z |
| 4 | E | 20 | U | 36 | k | 52 | 0 |
| 5 | F | 21 | V | 37 | l | 53 | 1 |
| 6 | G | 22 | W | 38 | m | 54 | 2 |
| 7 | H | 23 | X | 39 | n | 55 | 3 |
| 8 | I | 24 | Y | 40 | o | 56 | 4 |
| 9 | J | 25 | Z | 41 | p | 57 | 5 |
| 10 | K | 26 | a | 42 | q | 58 | 6 |
| 11 | L | 27 | b | 43 | r | 59 | 7 |
| 12 | M | 28 | c | 44 | s | 60 | 8 |
| 13 | N | 29 | d | 45 | t | 61 | 9 |
| 14 | O | 30 | e | 46 | u | 62 | + |
| 15 | P | 31 | f | 47 | v | 63 | / |

比如举下面2个例子：  
a) 字符长度为能被3整除时：比如“Tom” ：

[复制代码](javascript:void(0);)

T o m  
ASCII: 84 111 109  
8bit字节: 01010100 01101111 01101101  
6bit字节: 010101 000110 111101 101101  
十进制: 21 6 61 45  
对应编码: V G 9 t

[复制代码](javascript:void(0);)

所以，btoa('Tom') = VG9t

b) 字符串长度不能被3整除时，比如“Lucy”：

[复制代码](javascript:void(0);)

L u c y  
ASCII: 76 117 99 121  
8bit字节: 01001100 01110101 01100011 01111001 00000000 00000000  
6bit字节: 010011 000111 010101 100011 011110 010000 000000 000000  
十进制: 19 7 21 35 30 16 (异常) (异常)   
对应编码: T H V j e Q = =

[复制代码](javascript:void(0);)

由于Lucy只有4个字母，所以按3个一组的话，第二组还有两个空位，所以需要用0来补齐。这里就需要注意，因为是需要补齐而出现的0，所以转化成十进制的时候就不能按常规用base64编码表来对应，所以不是a， 可以理解成为一种特殊的“异常”，编码应该对应“=”。

有了上面的理论，那我们实现一个base64编码就容易了。

[复制代码](javascript:void(0);)

/\*\*  
 \* base64 encoding & decoding  
 \* for fixing browsers which don't support Base64 | btoa |atob  
 \*/  
  
(function (win, undefined) {  
   
 var Base64 = function () {  
 var base64hash = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/';  
   
 // btoa method  
 function \_btoa (s) {  
 if (/([^\u0000-\u00ff])/.test(s)) {  
 throw new Error('INVALID\_CHARACTER\_ERR');  
 }   
 var i = 0,  
 prev,  
 ascii,  
 mod,  
 result = [];  
  
 while (i < s.length) {  
 ascii = s.charCodeAt(i);  
 mod = i % 3;  
  
 switch(mod) {  
 // 第一个6位只需要让8位二进制右移两位  
 case 0:  
 result.push(base64hash.charAt(ascii >> 2));  
 break;  
 //第二个6位 = 第一个8位的后两位 + 第二个8位的前4位  
 case 1:  
 result.push(base64hash.charAt((prev & 3) << 4 | (ascii >> 4)));  
 break;  
 //第三个6位 = 第二个8位的后4位 + 第三个8位的前2位  
 //第4个6位 = 第三个8位的后6位  
 case 2:  
 result.push(base64hash.charAt((prev & 0x0f) << 2 | (ascii >> 6)));  
 result.push(base64hash.charAt(ascii & 0x3f));  
 break;  
 }  
  
 prev = ascii;  
 i ++;  
 }  
  
 // 循环结束后看mod, 为0 证明需补3个6位，第一个为最后一个8位的最后两位后面补4个0。另外两个6位对应的是异常的“=”；  
 // mod为1，证明还需补两个6位，一个是最后一个8位的后4位补两个0，另一个对应异常的“=”  
 if(mod == 0) {  
 result.push(base64hash.charAt((prev & 3) << 4));  
 result.push('==');  
 } else if (mod == 1) {  
 result.push(base64hash.charAt((prev & 0x0f) << 2));  
 result.push('=');  
 }  
  
 return result.join('');  
 }  
  
 // atob method  
 // 逆转encode的思路即可  
 function \_atob (s) {  
 s = s.replace(/\s|=/g, '');  
 var cur,  
 prev,  
 mod,  
 i = 0,  
 result = [];  
  
 while (i < s.length) {  
 cur = base64hash.indexOf(s.charAt(i));  
 mod = i % 4;  
  
 switch (mod) {  
 case 0:  
 //TODO  
 break;  
 case 1:  
 result.push(String.fromCharCode(prev << 2 | cur >> 4));  
 break;  
 case 2:  
 result.push(String.fromCharCode((prev & 0x0f) << 4 | cur >> 2));  
 break;  
 case 3:  
 result.push(String.fromCharCode((prev & 3) << 6 | cur));  
 break;  
   
 }  
  
 prev = cur;  
 i ++;  
 }  
  
 return result.join('');  
 }  
  
 return {  
 btoa: \_btoa,  
 atob: \_atob,  
 encode: \_btoa,  
 decode: \_atob  
 };  
 }();  
  
 if (!win.Base64) { win.Base64 = Base64 }  
 if (!win.btoa) { win.btoa = Base64.btoa }  
 if (!win.atob) { win.atob = Base64.atob }  
  
 })(window)

[复制代码](javascript:void(0);)