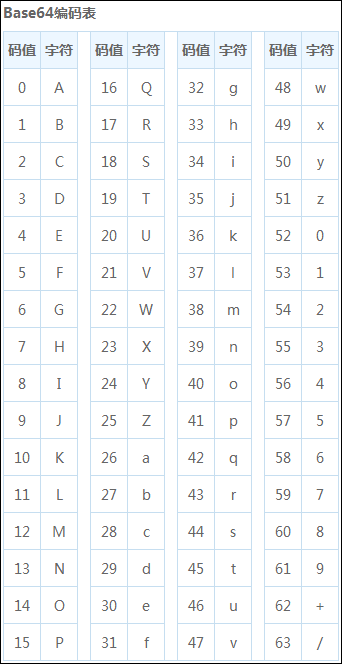
**BASE64 编码**

Base64被定义为：Base64内容传送编码被设计用来把任意序列的8位字节描述为一种不易被人直接识别的形式。

　Base64编码是把每3个8位字节（3\*8=24）转成4个6位字节（4\*6=24），且在每个6位字节前补两个0，重新形成8位一个字节的形式，这样总的长度还是相同的，并且保证最高位不是1（即不是负数）。

　　如果最后剩下的字符不足3个字符，则用0填充高位，输出字符使用“=”。因此编码后的字符串末尾可能会出现一个或两个“=”。

　　根据上述原理，最高两位均为0，那么剩下6位全为1二进制对应的十进制为63，所以最大数不会超过63，根据这64个数进行一个码值与字符的对应，就成了base64编码，对应编码表如下图所示：



　　在Java中我们使用BASE64Encoder对象调用encode方法来将某个字节数组进行编码。

　　我们将上面的MD5码进行base64编码：

1 BASE64Encoder be = new BASE64Encoder();

2 String base64 = be.encode(md5);

3 System.out.println(base64);

常见于邮件、http加密，截取http信息，你就会发现登录操作的用户名、密码字段通过BASE64加密的

1. /\*\*
2. \* BASE64解密
3. \*
4. \* @param key
5. \* @return
6. \* @throws Exception
7. \*/
8. **public** **static** **byte**[] decryptBASE64(String key) **throws** Exception {
9. **return** (**new** BASE64Decoder()).decodeBuffer(key);
10. }
12. /\*\*
13. \* BASE64加密
14. \*
15. \* @param key
16. \* @return
17. \* @throws Exception
18. \*/
19. **public** **static** String encryptBASE64(**byte**[] key) **throws** Exception {
20. **return** (**new** BASE64Encoder()).encodeBuffer(key);
21. }

主要就是BASE64Encoder、BASE64Decoder两个类，我们只需要知道使用对应的方法即可。另，BASE加密后产生的字节位数是8的倍数，如果不够位数以=符号填充。

**MD5**

MD5 -- message-digest algorithm 5 （信息-摘要算法）缩写，广泛用于加密和解密技术，常用于文件校验。校验？不管文件多大，经过MD5后都能生成唯一的MD5值。好比现在的ISO校验，都是MD5校验。怎么用？当然是把ISO经过MD5后产生MD5的值。一般下载linux-ISO的朋友都见过下载链接旁边放着MD5的串。就是用来验证文件是否一致的。

MD5可以将所有的数字组合进行一个算法的运算得出一个新的数字组合，并且这个新的数字组合都是128位，即16字节，也称为MD5码**。使这个字节数组转换成一个没有乱码的字符串作为一种随机数？我们可以采用“Base64”编码。**MD5码是不可逆的，即无法通过MD5码来进行反向运算得到原始的数据。MD5的应用场合非常多，比如保存用户密码（防止数据库中用户密码被管理员看到，所以数据库中的密码不许使用明文密码），或者文件校验（防止下载的文件是被别人修改后的文件而不是官方原文件，比如苹果开发工具的XCodeGhost事件），或者光盘破损校验等等。

　　在Java中通过MessageDigest对象来调用方法获取某些数字组合的MD5码（顺便说一句，MessageDigest也可以用来获取SHA校验码）。通过MessageDigest对象调用digest方法来将数字组合转变为字符数组：

1. /\*\*
2. \* MD5加密
3. \*
4. \* @param data
5. \* @return
6. \* @throws Exception
7. \*/
8. **public** **static** **byte**[] encryptMD5(**byte**[] data) **throws** Exception {
10. MessageDigest md5 = MessageDigest.getInstance(KEY\_MD5);
11. md5.update(data);
13. **return** md5.digest();
15. }

**SHA**  
SHA(Secure Hash Algorithm，安全散列算法），数字签名等密码学应用中重要的工具，被广泛地应用于电子商务等信息安全领域。虽然，SHA与MD5通过碰撞法都被破解了， 但是SHA仍然是公认的安全加密算法，较之MD5更为安全。

1. /\*\*
2. \* SHA加密
3. \*
4. \* @param data
5. \* @return
6. \* @throws Exception
7. \*/
8. **public** **static** **byte**[] encryptSHA(**byte**[] data) **throws** Exception {
10. MessageDigest sha = MessageDigest.getInstance(KEY\_SHA);
11. sha.update(data);
13. **return** sha.digest();
15. }

**HMAC**  
HMAC(Hash Message Authentication Code，散列消息鉴别码，基于密钥的Hash算法的认证协议。消息鉴别码实现鉴别的原理是，用公开函数和密钥产生一个固定长度的值作为认证标识，用这个标识鉴别消息的完整性。使用一个密钥生成一个固定大小的小数据块，即MAC，并将其加入到消息中，然后传输。接收方利用与发送方共享的密钥进行鉴别认证等。

1. /\*\*
2. \* 初始化HMAC密钥
3. \*
4. \* @return
5. \* @throws Exception
6. \*/
7. **public** **static** String initMacKey() **throws** Exception {
8. KeyGenerator keyGenerator = KeyGenerator.getInstance(KEY\_MAC);
10. SecretKey secretKey = keyGenerator.generateKey();
11. **return** encryptBASE64(secretKey.getEncoded());
12. }
14. /\*\*
15. \* HMAC加密
16. \*
17. \* @param data
18. \* @param key
19. \* @return
20. \* @throws Exception
21. \*/
22. **public** **static** **byte**[] encryptHMAC(**byte**[] data, String key) **throws** Exception {
24. SecretKey secretKey = **new** SecretKeySpec(decryptBASE64(key), KEY\_MAC);
25. Mac mac = Mac.getInstance(secretKey.getAlgorithm());
26. mac.init(secretKey);
28. **return** mac.doFinal(data);
30. }