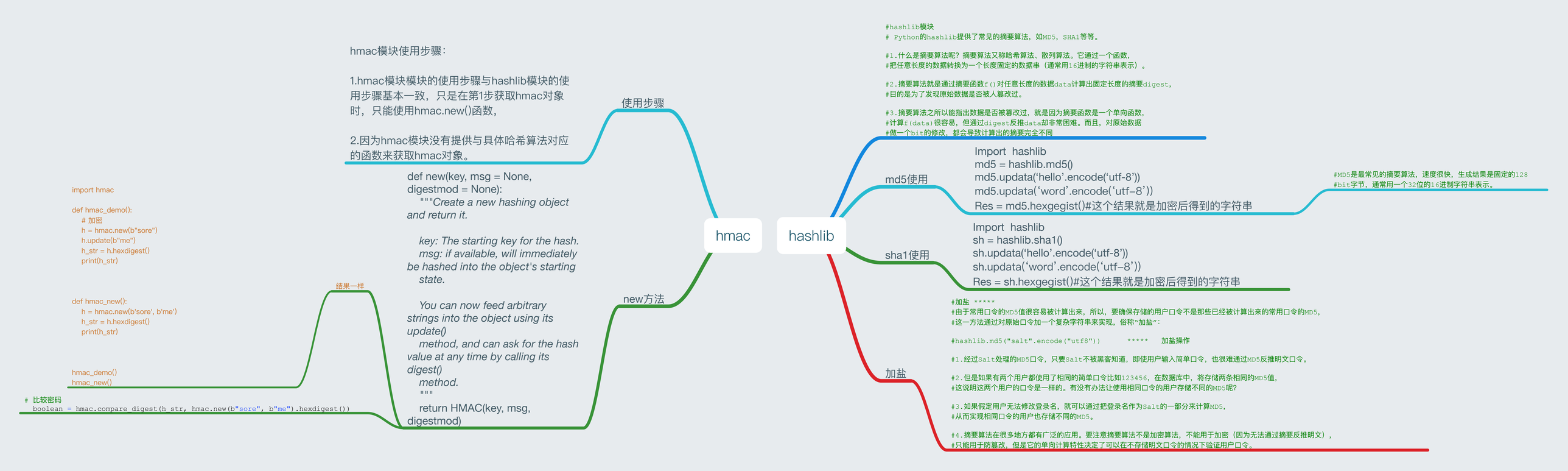
hmac模块使用步骤：
1.hmac模块模块的使用步骤与hashlib模块的使用步骤基本一致，只是在第1步获取hmac对象时，只能使用hmac.new()函数，
2.因为hmac模块没有提供与具体哈希算法对应的函数来获取hmac对象。
使用步骤
hmac
new方法
def new(key, msg = None, digestmod = None):
"""Create a new hashing object and return it.
key: The starting key for the hash.
msg: if available, will immediately be hashed into the object's starting state.
You can now feed arbitrary strings into the object using its update()
method, and can ask for the hash value at any time by calling its digest()
method.
"""
return HMAC(key, msg, digestmod)
import hmac
def hmac_demo():
# 加密
h = hmac.new(b"sore")
h.update(b"me")
h_str = h.hexdigest()
print(h_str)
def hmac_new():
h = hmac.new(b'sore', b'me')
h_str = h.hexdigest()
print(h_str)
结果一样
hmac_demo()
hmac_new()
# 比较密码
boolean = hmac.compare_digest(h_str, hmac.new(b"sore", b"me").hexdigest())
hashlib
#hashlib模块
Python的hashlib提供了常见的摘要算法，如MD5，SHA1等等。
#1.什么是摘要算法呢？摘要算法又称哈希算法、散列算法。它通过一个函数，
#把任意长度的数据转换为一个长度固定的数据串（通常用16进制的字符串表示）。
#2.摘要算法就是通过摘要函数f()对任意长度的数据data计算出固定长度的摘要digest，
#目的是为了发现原始数据是否被人篡改过。
#3.摘要算法之所以能指出数据是否被篡改过，就是因为摘要函数是一个单向函数，
#计算f(data)很容易，但通过digest反推data却非常困难。而且，对原始数据
#做一个bit的修改，都会导致计算出的摘要完全不同
md5使用
Import hashlib
md5 = hashlib.md5()
md5.updata('hello'.encode('utf-8'))
md5.updata('word'.encode('utf-8'))
Res = md5.hexgegist()#这个结果就是加密后得到的字符串
#MD5是最常见的摘要算法，速度很快，生成结果是固定的128
#bit字节，通常用一个32位的16进制字符串表示。
sha1使用
Import hashlib
sh = hashlib.sha1()
sh.updata('hello'.encode('utf-8'))
sh.updata('word'.encode('utf-8'))
Res = sh.hexgegist()#这个结果就是加密后得到的字符串
加盐
#加盐 *****
#由于常用口令的MD5值很容易被计算出来，所以，要确保存储的用户口令不是那些已经被计算出来的常用口令的MD5，
#这一方法通过对原始口令加一个复杂字符串来实现，俗称"加盐":
#hashlib.md5("salt".encode("utf8")) ***** 加盐操作
#1.经过Salt处理的MD5口令，只要Salt不被黑客知道，即使用户输入简单口令，也很难通过MD5反推明文口令。
#2.但是如果有两个用户都使用了相同的简单口令比如123456，在数据库中，将存储两条相同的MD5值，
#这说明这两个用户的口令是一样的。有没有办法让使用相同口令的用户存储不同的MD5呢？
#3.如果假定用户无法修改登录名，就可以通过把登录名作为Salt的一部分来计算MD5，
#从而实现相同口令的用户也存储不同的MD5。
#4.摘要算法在很多地方都有广泛的应用。要注意摘要算法不是加密算法，不能用于加密（因为无法通过摘要反推明文），
#只能用于防篡改，但是它的单向计算特性决定了可以在不存储明文口令的情况下验证用户口令。