# **[JMeter乱码问题的解决](https://www.cnblogs.com/houzhizhe/p/7122811.html)**

一、JMeter返回数据是乱码

　　解决办法是：

　　在JMeter安装路径的bin目录下，以记事本打开文件jmeter.properties，

       找到Sampleresult.default.encoding，将它的值改为 utf-8 ，并把这行开头的 # 去掉，

　　保存文件，重启JMeter。

二、JMeter在body中汉字符显示成乱码

　　这种情况在JMeter3.0的版本中才会出现，注意，这并不是乱码，而是由于3.0中优化body data后，使用默认的字体（Consolas）不支持汉字的显示。

　　解决办法是：

打开jmeter.properties，找到jsyntaxtextarea.font.family这个选项，取消前面的#号，使其使用Hack字体即可。当然，也可以换成 宋体以及其他支持的字体集。

# **[接口测试用例设计](https://www.cnblogs.com/houzhizhe/p/7449246.html)**

对于web接口测试来说有哪些测试要点：

　　1、请求是否正确，默认请求成功是200，如果请求错误也能返回404、500等。

　　2、检查返回数据的正确性与格式；json是一种非常创建的格式。

　　3、接口的安全性，一般web都不会暴露在网上任意被调用，需要做一些限制，比如鉴权或认证。

　　4、接口的性能，web接口同样注重性能，这直接影响用户的使用体验。如果我搜索一个关键字半天结果都没返回，果断弃用。

接口测试的用例设计出了要验证正常功能之外，还需要考虑其他的一些情况，总结起来，接口测试的用例设计可以从以下几个方面入手：

**1.功能用例设计**

服务端的接口与客户端的功能是对应的，那么这个接口是否能提供给客户端某个特定功能所需要的数据自然是我们需要验证的地方，功能性用例的主要目的是帮助我们验证该接口最初设计的功能是否被实现以及该功能是否按照规定的接口文档进行实现等等。

**2.业务逻辑用例设计**

业务逻辑方面的测试用例主要是针对服务端接口的处理逻辑进行的用例设计，这种用例设计不是针对某个功能点是否实现，而是对接口的处理逻辑以及一些相互依赖的业务进行验证，通常依照接口的逻辑流程图来进行。举一个例子，购物系统中的两个动作：登录和下单操作，这两个业务是相互依赖的，下单操作必须在登录完成后（登录状态下），否则无法完成下单，这个时候我们就可以设计这样一条case：没有登录的状态下进行下单操作，看服务端如何处理。

**3.异常处理的情况**

服务端接口和客户端之间通常是通过HTTP请求来传递数据，在发送请求的时候，客户端会携带各种不同的参数，此时服务端会根据不同的参数进行不同的处理，所以异常处理主要是针对请求中的参数情况：比如参数增加和缺省、参数的数据类型错误，参数携带错误的值、参数为空等等，这需要我们根据接口文档中各种不同的参数去构造不同的参数异常，检查服务端的响应情况。

**4.性能和安全性方面**

服务器的性能往往是个非常重要的关注点，在实际的测试中我们主要关注的是接口的QPS数值，以及CPU和内存占用等性能指标，通常借助于其他工具比如LoadRunner进行性能测试。安全性方面，主要考虑一些常见的安全策略比如请求加密、sql注入等等。

# **[接口加密测试](https://www.cnblogs.com/houzhizhe/p/7461163.html)**

**拿什么守护你，我的接口？**    引自 <http://www.jianshu.com/p/add7518a3fbe>

接口测试那什么来做，这是刚接触接口测试的同学最常问的问题。  
个人觉得这个和你在拍照的时候是直接用美图秀秀里面的滤镜模式和自己去学PS后期进行处理图片的逻辑是一样一样的。  
在一般的情况下，滤镜处理出来的和PS出来的效果一样么？一样  
但是在妹子需要在背景里面加个城堡，鲜花满地的时候，或者妹子需要做成女神一样的特效的时候  
滤镜能处理么？不能。  
PS能处理么？能！！！就是，就是，就是，学习成本稍微高一丢丢。  
高的这一丢丢是什么呢？那就是我们经常在接口测试中遇到的各种各样用来守护接口的安全机制问题。  
以下图片自行体会一下 接口测试工具和自行编写接口测试代码的区别。  
美图秀秀的效果图：



1.jpg

PS处理的效果图：



2.jpg

为啥非要费劲的在接口上加上各种复杂的安全机制呢？  
好问题。  
如果你不想自己的信息轻易的被泄露出去，自家网站被攻击的瘫痪，公司倒闭，CEO跑路，男女朋友分手。。。还是乖乖的跟着CC先生看一下接口相关的安全机制吧。  
一般接口开发中有以下常用的几种安全机制：

* 用户认证
* 数字签名
* 接口加密

用户认证  
一般的接口测试工具都会提供一个User Auth/Authorization的选项，以Postman为栗子，你可以看到以下的选项：

* 基本认证（Basic Auth）
* 摘要认证（Digest Auth）
* OAuth 1.0a
* OAuth 2.0（最常见，现在的网站接口多数提供此用户认证方式）  
  在对应的工具上，你可以选取对应的用户认证选项，这里CC先生主要介绍如果用Python如何实现用户认证。

1 首先安装Requests库，Requests库的get()和post()方法均提供有auth参数，用于设置用户签名。  
2 假定我们有一个接口为添加一个新的公告，接口需要认证：auth=("username","password")  
nid 或 name两个参数二选一  
3 伪代码：

def test\_get\_notice\_list\_nid\_sucess(self):

auth\_user = ('admin' , 'admin123456')

r = requests.get(self.base\_url, auth = auth\_user, params = {'nid' : 1})

result = r.json()

self.assertEqual(result['status'], 200)

....

# **数字签名**

在使用 HTTP/SOAP 协议传输数据的时候，签名作为其中一个参数，可以起到关键作用：  
先来一个简单的，通过客户的密钥，服务端的密钥匹配；  
这个很有好理解，例如一个接口传参为：  
[http://127.0.0.1:8000/api/?a=1&b=2](http://127.0.0.1:8000/api/?a=1&b=2" \t "https://www.cnblogs.com/houzhizhe/p/_blank)  
假设签名的密钥为：@signpassword  
加上签名之后的接口参数为：  
[http://127.0.0.1:8000/sign/?a=1&b=2&sign=@signpassword](http://127.0.0.1:8000/sign/?a=1&b=2&sign=@signpassword" \t "https://www.cnblogs.com/houzhizhe/p/_blank)  
但是，这样的sign 参数明文传输是不安全的,一般会选择一些加密算法，比如MD5 算法（MD5算法是不可逆向的），比如MD5代码如下：

import hashlib

md5 = hashlib.md5()

sign\_str = "@signpassword"

sign\_bytes\_utf8 = sign\_str.encode()

md5.update(sign\_bytes\_utf8)

sign\_md5 = md5.hexdigest()

print(sign\_md5)

执行后得到：6648e929329e53e7a91c50ae685a88b5  
此时带签名的接口为：  
[http://127.0.0.1:8000/sign/?a=1&b=2&sign=6648e929329e53e7a91c50ae685a88b5](http://127.0.0.1:8000/sign/?a=1&b=2&sign=6648e929329e53e7a91c50ae685a88b5" \t "https://www.cnblogs.com/houzhizhe/p/_blank)  
所以，当服务器接收到请求后，同样需要对“signpassword”进行 MD5 加密，然后，比对与调用者传来的 sign 加密串是否一致，从而来鉴别调用者是否有权限使用该接口。  
接着，我们来理解一个复杂一点的：把sign参数传递为api key（申请获取）+timestramp（时间戳）同样需要 用代码来实现，原理和上面这个一致的。（伪代码）

def setUp(self):

self.base\_url = "http://127.0.0.1:8000/api/sec\_add\_notice/"

# app\_key

self.api\_key = "&APIkey"

# 当前时间

now\_time = time()

self.client\_time = str(now\_time).split('.')[0]

# sign

md5 = hashlib.md5()

sign\_str = self.client\_time + self.api\_key

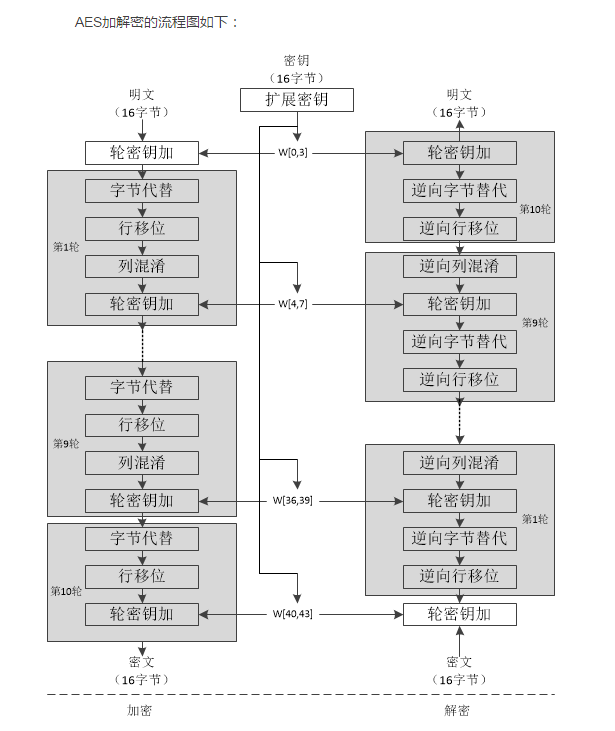
sign\_bytes\_utf8 = sign\_str.encode(encoding="utf-8")

md5.update(sign\_bytes\_utf8)

self.sign\_md5 = md5.hexdigest()

# **接口加密**

通常接口会使用更复杂一点的方式来进行加密的操作，常见的是AES的使用，放一张图让大家感受一下AES加解密的过程：



1.png

详细的过程可以参见[http://www.mamicode.com/info-detail-514466.html](http://www.mamicode.com/info-detail-514466.html" \t "https://www.cnblogs.com/houzhizhe/p/_blank) 或者自行百度。

Python里面有一个很好的黑魔法，叫PyCrypto库，支持常见的 DES、AES 加密以及 MD5、SHA 各种 HASH 运算。（官方网站下载最新版本：[https://www.dlitz.net/software/pycrypto/](https://www.dlitz.net/software/pycrypto/" \t "https://www.cnblogs.com/houzhizhe/p/_blank)  
另外，也可以在 PyPi 仓库中下载安装：[https://pypi.python.org/pypi/pycrypto）](https://pypi.python.org/pypi/pycrypto%EF%BC%89" \t "https://www.cnblogs.com/houzhizhe/p/_blank)  
对于AES的加密来说，看一下用了PyCrypto库的结果  
加密：

from Crypto.Cipher import AES# 加密

obj = AES.new('This is a key123', AES.MODE\_CBC, 'This is an IV456')

message = "The answer is no"

ciphertext = obj.encrypt(message)print(ciphertext)

程序运行后的结果为：b'\xd6\x83\x8dd!VT\x92\xaa`A\x05\xe0\x9b\x8b\xf1'  
AES加密里面有两个关键，一个是key（必须为16,24,32位），一个是VI（必须为16位）  
解密：解谜者必须要同时知道key和VI才可以解密

obj2 = AES.new('This is a key123', AES.MODE\_CBC, 'This is an IV456')

s = obj2.decrypt(ciphertext)

print(s)

===  
So,接口加密测试的关键就在于开发小哥哥们用了什么加密算法来加密接口。你用相同的加密算法来加密你要发送的值 即可。

CC先生说，现在有很多的公司也把加密的算法直接封装成一个类或者一个接口，也就是测试的时候，你直接调用加密的类或者加密的接口即可完成你的加密工作。剩下的接口测试和以往没加密的接口测试一样没两样了。  
附送一个Pypi库里一堆加密算法库的链接，自取：  
[https://pypi.python.org/pypi?%3Aaction=search&term=crypt&submit=search](https://pypi.python.org/pypi?:action=search&term=crypt&submit=search" \t "https://www.cnblogs.com/houzhizhe/p/_blank)