# Openssl实用讲解

目录

[Openssl实用讲解 1](#_Toc14853533)

[Openssl 编译动态库 3](#_Toc14853534)

[1、下载openssl 3](#_Toc14853535)

[2、解压到指定目录 3](#_Toc14853536)

[3、安装 3](#_Toc14853537)

[4、位置 3](#_Toc14853538)

[5、测试 3](#_Toc14853539)

[6、交叉编译 7](#_Toc14853540)

[Openssl 使用方法 7](#_Toc14853541)

[API介绍 7](#_Toc14853542)

[代码分析 9](#_Toc14853543)

[https连接过程 9](#_Toc14853544)

[HTTPS、SSL、TLS 10](#_Toc14853545)

[Libcurl 11](#_Toc14853546)

[Libcurl编译 11](#_Toc14853547)

[1、下载 11](#_Toc14853548)

[2、安装 11](#_Toc14853549)

[使用 11](#_Toc14853550)

[1、API接口 11](#_Toc14853551)

[2、代码示例 12](#_Toc14853552)

[3、线程安全 14](#_Toc14853553)

[Libevent网络库 15](#_Toc14853554)

[环境搭建 15](#_Toc14853555)

[1、 下载 15](#_Toc14853556)

[2、 安装 15](#_Toc14853557)

[3、 编译 15](#_Toc14853558)

[备注 16](#_Toc14853559)

[储备知识点 16](#_Toc14853560)

[数字签名简单理解： 16](#_Toc14853561)

[兴趣 16](#_Toc14853562)

[计划 16](#_Toc14853563)

[github账号 16](#_Toc14853564)

## Openssl 编译动态库

### 1、下载openssl

<https://www.openssl.org/source/>

<http://distfiles.macports.org/openssl/>

这两个网站都可以下载，选择最新源码

### 2、解压到指定目录

tar zxvf openssl-1.0.2s.tar.gz xxx

### 3、安装

然后是经典三部曲，首先使用 ./config   make   make install

./config shared --prefix=/home/linux/opt/openssl --openssldir=/home/linux/opt/openssl/ssl

prefix 是安装目录，openssldir 是配置文件目录，shared 作用是生成动态连接库

### 4、位置

在指定位置生成bin include lib ssl

### 5、测试

测试内容参考demo

-------------------------------------------------------------

#### cryptotest.h

#ifndef \_CRYPTOTEST\_H\_

#define \_CRYPTOTEST\_H\_

typedef enum {

GENERAL = 0,

ECB,

CBC,

CFB,

OFB,

TRIPLE\_ECB,

TRIPLE\_CBC

}CRYPTO\_MODE;

//string DES\_Encrypt(const string cleartext, const string key, CRYPTO\_MODE mode);

//string DES\_Decrypt(const string ciphertext, const string key, CRYPTO\_MODE mode);

char \* RC4\_Encrypt(const char \*cleartext, const char \* key, int cleartextlen, int keylen);

char \* RC4\_Decrypt(const char \* ciphertext, const char \* key, int cleartextlen, int keylen);

#endif //\_CRYPTOTEST\_H\_

-------------------------------------------------------------

#### openssltest.c

#include "cryptotest.h"

#include <string.h>

#include <stdio.h>

int main()

{

char cleartext[] = "中国北京12345$abcde%ABCDE@！！！";

char \*ciphertext;

char key[] = "beijingchina1234567890ABCDEFGH!!!";

ciphertext = RC4\_Encrypt(cleartext, key, strlen(cleartext), strlen(key));

char \* decrypt = RC4\_Decrypt(ciphertext, key, strlen(cleartext), strlen(key));

printf("cleartext:%s\n", cleartext);

printf("genarate ciphertext:%s\n", ciphertext);

printf("src ciphertext:%s\n", ciphertext);

printf("genarate ciphertext:%s\n", decrypt);

if (strcmp(cleartext, decrypt) == 0)

printf("RC4 crypto ok!!!\n");

else

printf("RC4 crypto error!!!\n");

return 0;

}

-------------------------------------------------------------

#### rc4test.c

#include <stdlib.h>

#include <openssl/rc4.h>

#include <string.h>

#include "cryptotest.h"

char \* RC4\_Encrypt(const char \*cleartext, const char \* key, int cleartextlen, int keylen)

{

RC4\_KEY rc4key;

char\* tmp = malloc(cleartextlen + 1);

memset(tmp, 0, cleartextlen + 1);

RC4\_set\_key(&rc4key, keylen, (const unsigned char\*)key);

RC4(&rc4key, cleartextlen, (const unsigned char\*)cleartext, tmp);

return tmp;

}

char \* RC4\_Decrypt(const char \* ciphertext, const char \* key, int cleartextlen, int keylen)

{

RC4\_KEY rc4key;

unsigned char\* tmp = malloc(cleartextlen + 1);

memset(tmp, 0, cleartextlen + 1);

RC4\_set\_key(&rc4key, keylen, (const unsigned char\*)key);

RC4(&rc4key, cleartextlen, (const unsigned char\*)ciphertext, tmp);

return tmp;

}

-------------------------------------------------------------

#### makefile

CC = gcc

CPP = g++

RM = rm -rf

## debug flag

DBG\_ENABLE = 0

## source file path

SRC\_PATH := .

## target exec file name

TARGET := openssltest

## get all source files

SRCS += $(wildcard $(SRC\_PATH)/\*.c)

## all .o based on all .c

OBJS := $(SRCS:.c=.o)

## need libs, add at here

LIBS := ssl crypto

## used headers file path

INCLUDE\_PATH := ../openssl\_install\_dir/include/

## used include librarys file path

LIBRARY\_PATH := ../openssl\_install\_dir/lib/

## debug for debug info, when use gdb to debug

ifeq (1, ${DBG\_ENABLE})

CFLAGS += -D\_DEBUG -O0 -g -DDEBUG=1

endif

## get all include path

CFLAGS += $(foreach dir, $(INCLUDE\_PATH),-I $(dir))

#SourceFiles = $(foreach dir,$(SourcePath),$(shell find $(dir) -maxdepth 5 -name "\*.c"))

## get all library path

LDFLAGS += $(foreach lib, $(LIBRARY\_PATH), -L $(lib))

## get all librarys

LDFLAGS += $(foreach lib, $(LIBS), -l $(lib))

all: clean build

build:

$(CC) -c $(SRCS) $(CFLAGS)

$(CC) $(CFLAGS) -o $(TARGET) $(OBJS) $(LDFLAGS)

#$(RM) $(OBJS)

clean:

$(RM) $(OBJS) $(TARGET)

-------------------------------------------------------------

### 6、交叉编译

进入源码目录

./config CC=arm-linux-gcc no-asm shared --prefix=/home/linux/arm/openssl --openssldir=/home/linux/arm/openssl/ssl

make make install

参考博客<https://blog.csdn.net/qq_28643619/article/details/86639321>

## Openssl 使用方法

### API介绍

初始化

int SSL\_library\_init(void);

设置会话环境

SSL\_CTX \*SSL\_CTX\_new(SSL\_METHOD \* method);

加载CA证书

为了验证证书，首先需要加载CA证书（因为对端证书需要用CA证书来验证）。SSL\_CTX\_load\_verify\_locations加载CA证书到SSL\_CTX结构中。

函数原型如下：

int SSL\_CTX\_load\_verify\_locations(SSL\_CTX \*ctx, const char \*CAfile, const char \*CApath);

加载证书(客户端/服务端证书)

SSL\_CTX\_use\_certificate\_file函数加载一个证书到一个SSL\_CTX结构中

SSL\_use\_certificate\_file函数加载一个证书到SSL结构中

当创建SSl结构，SSL结构自动加载同样的证书，并且包含了SSL\_CTX结构；因此，当创建SSL结构只需要调用SSL\_use\_certificate\_file()，除非需要加载一个不同的证书，而不是默认证书包含在SSL\_CTX结构中。

设置CTX的属性，通常的设置是指定SSL握手阶段证书的验证方式和加载自己的证书。

int SSL\_CTX\_set\_verify(SSL\_CTX \*ctx,intmode,int(\*verify\_callback),int(X509\_STORE\_CTX \*));

为SSL会话环境加载CA证书的函数是：  
SSL\_CTX\_load\_verify\_location(SSL\_CTX \*ctx,const char \*Cafile,constchar \*Capath);

为SSL会话加载用户私钥的函数是：

SSL\_CTX\_use\_PrivateKey\_file(SSL\_CTX \*ctx,const char\* file,inttype);

在将证书和私钥加载到SSL会话环境之后，就可以调用下面的函数来验证私钥和证书是否相符：

int SSL\_CTX\_check\_private\_key(SSL\_CTX \*ctx);

建立SSL套接字

SSL套接字是建立在普通的TCP套接字基础之上，在建立SSL套接字时可以使用下面的一些函数：

SSL \*SSl\_new(SSL\_CTX \*ctx);    //申请一个SSL套接字

int SSL\_set\_fd(SSL \*ssl,int fd);    //绑定读写套接字

int SSL\_set\_rfd(SSL \*ssl,int fd);    //绑定只读套接字

int SSL\_set\_wfd(SSL \*ssl,int fd);    //绑定只写套接字

完成SSL握手  
在成功创建SSL套接字后，客户端应使用函数SSL\_connect( )替代传统的函数connect( )来完成握手过程:  
int SSL\_connect(SSL \*ssl);

而对服务器来讲，则应使用函数SSL\_ accept ( )替代传统的函数accept ( )来完成握手过程:

int SSL\_accept(SSL \*ssl);

握手过程完成之后，通常需要询问通信双方的证书信息，以便进行相应的验证，这可以借助于下面的函数来实现:

X509 \*SSL\_get\_peer\_certificate(SSL \*ssl);

该函数可以从SSL套接字中提取对方的证书信息，这些信息已经被SSL验证过了。

X509\_NAME \*X509\_get\_subject\_name(X509 \*a);

该函数得到证书所用者的名字。

进行数据传输

当SSL握手完成之后，就可以进行安全的数据传输了，在数据传输阶段，需要使用SSL\_read( )和SSL\_write()来替代传统的read( )和write( )函数，来完成对套接字的读写操作：

int SSL\_read(SSL \*ssl,void \*buf,int num);

int SSL\_write(SSL \*ssl,const void \*buf,int num);

结束SSL通信

当客户端和服务器之间的数据通信完成之后，调用下面的函数来释放已经申请的SSL资源：

int SSL\_shutdown(SSL \*ssl);    //关闭SSL套接字

void SSl\_free(SSL \*ssl);    //释放SSL套接字

void SSL\_CTX\_free(SSL\_CTX \*ctx);     //释放SSL会话环境

对程序来说,openssl将整个握手过程用一对函数体现,即客户端的SSL\_connect和服务端的SSL\_accept.而后的应用层数据交换则用SSL\_read和 SSL\_write来完成.

### 代码分析

编译程序用下列命令：

gcc -Wall ssl-client.c -o client -L /usr/local/openssl/lib/ -Wl,-R /usr/local/openssl/lib/ -lssl -lcrypto

gcc -Wall ssl-server.c -o server -L /usr/local/openssl/lib/ -Wl,-R /usr/local/openssl/lib/ -lssl -lcrypto

运行程序用如下命令：

./server 7838 1 127.0.0.1 cacert.pem privkey.pem

./client 127.0.0.1 7838

用下面这两个命令产生上述cacert.pem和privkey.pem文件：

openssl genrsa -out privkey.pem 2048

openssl req -new -x509 -key privkey.pem -out cacert.pem -days 1095

参考博客<https://blog.csdn.net/bytxl/article/details/17200571>

## https连接过程

客户端发送请求到服务器端

服务器端返回证书和公开密钥，公开密钥作为证书的一部分而存在

客户端验证证书和公开密钥的有效性，如果有效，则生成共享密钥并使用公开密钥加密发送到服务器端

服务器端使用私有密钥解密数据，并使用收到的共享密钥加密数据，发送到客户端

客户端使用共享密钥解密数据

SSL加密建立………

参考博客<https://blog.csdn.net/bytxl/article/details/17200571>

## HTTPS、SSL、TLS

三者之间的联系和区别

<https://blog.csdn.net/enweitech/article/details/81781405>

长连接 短连接

<https://www.cnblogs.com/gotodsp/p/6366163.html>

<https://www.jianshu.com/p/3fc3646fad80>

CA证书扫盲贴

<https://www.cnblogs.com/handsomeBoys/p/6556336.html>

签名、加密、证书的基本原理和理解

<https://blog.csdn.net/zjf535214685/article/details/79899693>

SSL/TLS协议运行机制的概述

<http://www.ruanyifeng.com/blog/2014/02/ssl_tls.html>

权博客

<https://www.cnblogs.com/ningci/p/10606408.html>

CNNIC事件

<https://www.williamlong.info/archives/4183.html>

# Libcurl

## Libcurl编译

### 1、下载

WEB SITE

Visit the curl web site for the latest news and downloads:

https://curl.haxx.se/

GIT

To download the very latest source off the GIT server do this:

git clone https://github.com/curl/curl.git

(you'll get a directory named curl created, filled with the source code)

### 2、安装

编译libcurl需要用到openssl

./configure --prefix=/mnt/e/MyTest/libcurl/libcurl-install-dir --with-ssl=/mnt/e/MyTest/openssl/openssl\_install\_dir

屏幕会输出配置项，根据配置项可安需求再次修改

make

make install

参考博客<https://www.jianshu.com/p/0429ff858454>

## 使用easy

### 1、API接口

1.       调用curl\_global\_init()初始化libcurl  
2.       调用curl\_easy\_init()函数得到 easy interface型指针  
3.       调用curl\_easy\_setopt()设置传输选项  
4.       根据curl\_easy\_setopt()设置的传输选项，实现回调函数以完成用户特定任务  
5.       调用curl\_easy\_perform()函数完成传输任务  
6.       调用curl\_easy\_cleanup()释放内存

2 void curl\_global\_cleanup(void);  
描述：在结束libcurl使用的时候，用来对curl\_global\_init做的工作清理。类似于close的函数。

注意：虽然libcurl是线程安全的，但curl\_global\_init是不能保证线程安全的，所以不要在每个线程中都调用curl\_global\_init，应该将该函数的调用放在主线程中。

API说明：

<https://blog.csdn.net/linux1500176279/article/details/78702470>

demo：

<https://blog.csdn.net/ilovec1/article/details/52039672>

阅读一下，是什么东西

<https://www.cnblogs.com/lidabo/p/4159589.html>

### 2、代码示例

main.c

#include <stdio.h>

#include <curl.h>

#include <fcntl.h>

#include <stdlib.h>

int OnDebug( CURL\* crl, curl\_infotype itype, char \* pData, size\_t size, void \* p)

{

printf("OnDebug\n");

}

static size\_t OnWriteData( void \*ptr, size\_t size, size\_t nmemb, void \*stream)

{

printf("OnWriteData\n");

fwrite(ptr, size, nmemb, (FILE\*)stream);

return nmemb \* size;

}

int url\_get(const char\* url, void \*p)

{

printf("in=================%s,===%s,===%d行\n",\_\_FILE\_\_,\_\_FUNCTION\_\_,\_\_LINE\_\_);

CURLcode res;

CURL\* curl = curl\_easy\_init();

if ( NULL == curl )

{

printf( "Info[%s] : CURLE\_FAILED\_INIT.\n", \_\_FUNCTION\_\_ );

printf("in=================%s,===%s,===%d行\n",\_\_FILE\_\_,\_\_FUNCTION\_\_,\_\_LINE\_\_);

return CURLE\_FAILED\_INIT;

}

//curl\_easy\_setopt( curl, CURLOPT\_VERBOSE, 1 );

//curl\_easy\_setopt( curl, CURLOPT\_DEBUGFUNCTION, OnDebug );

curl\_easy\_setopt( curl, CURLOPT\_URL, url);

curl\_easy\_setopt( curl, CURLOPT\_READFUNCTION, NULL );

curl\_easy\_setopt( curl, CURLOPT\_WRITEFUNCTION, OnWriteData );

curl\_easy\_setopt( curl, CURLOPT\_WRITEDATA, p);

/\*\*

\* 当多个线程都使用超时处理的时候，同时主线程中有sleep或是wait等操作。

\* 如果不设置这个选项，libcurl将会发信号打断这个wait从而导致程序退出。

\*/

curl\_easy\_setopt( curl, CURLOPT\_NOSIGNAL, 1 );

curl\_easy\_setopt( curl, CURLOPT\_CONNECTTIMEOUT, 10);

curl\_easy\_setopt( curl, CURLOPT\_TIMEOUT, 10 );

res = curl\_easy\_perform( curl );

curl\_easy\_cleanup( curl );

printf("in=================%s,===%s,===%d行\n",\_\_FILE\_\_,\_\_FUNCTION\_\_,\_\_LINE\_\_);

return res;

}

int main()

{

curl\_global\_init(CURL\_GLOBAL\_ALL);

char \*url = "http://isure.stream.qqmusic.qq.com/C2000004zcfr0neC9O.m4a?guid=2000001576&vkey=A8342FCB3D1C208FFC09690FB116AF312DB90A5911806BB5E311CCE520ACDAA26E239B6767FE6CEB39672FBEED28EA7BE90CEA11ADC62245&uin=&fromtag=50";

FILE \*pfile = fopen("a.mp3", "wb");

if(pfile == NULL) {

printf("open error\n");

return 0;

}

url\_get(url, pfile);

fclose(pfile);

printf("%s\n", curl\_version());

curl\_global\_cleanup();

return 0;

}

### 3、线程安全

线程安全问题

<https://www.cnblogs.com/edgeyang/articles/3722035.html>

### 4、FAQ

curl\_easy\_perform 和 writecallback是同步还是异步

猜测是同步

## 使用multi

<https://www.cnblogs.com/zhaoyl/p/4001151.html>

multi接口的使用会比easy 接口稍微复杂点，毕竟multi接口是依赖easy接口的，首先粗略的讲下其使用流程：curl\_multi \_init初始化一个multi curl对象，为了同时进行多个curl的并发访问，我们需要初始化多个easy curl对象，使用curl\_easy\_setopt进行相关设置，然后调用curl\_multi \_add\_handle把easy curl对象添加到multi curl对象中，添加完毕后执行curl\_multi\_perform方法进行并发的访问，访问结束后curl\_multi\_remove\_handle移除相关easy curl对象，curl\_easy\_cleanup清除easy curl对象，最后curl\_multi\_cleanup清除multi curl对象。

HTTP2接口

<https://curl.haxx.se/libcurl/c/CURLMOPT_PIPELINING.html>

# Libevent网络库

## 环境搭建

### 下载

在github上下载

<https://github.com/libevent/libevent>

如何提高性能

<https://www.cnblogs.com/yutingliuyl/p/7244203.html>

### 安装

### 编译

# Gstream

## 下载

可以在github中下载

# mbedTLS

## 下载

# Makefile

## 讲解

<https://blog.csdn.net/baidu_38172402/article/details/88864517>

makefile中cc会软连接到gcc 查找命令使用 which cc

指定文件可以使用make –f xxx

指定include路径，make –I 或 make –include-dir

$@ 表示目标的集合，就像一个数组 $@依次取出目标并执行

$< 表示所有依赖的目标集

Makefile中自带函数subst、filter

cc –MM main.c 可以输出main.c所依赖的头文件 –M会把标准库包含进来

GNU建议把编译器为每个源文件的自动生成依赖关系放到文件中，为每一个.c生成一个.d的Makefile文件，[.d]文件中就存放对应[.c]文件的依赖关系

执行make时，带“-n” 或 “—just-print”，那么只显示命令，但不会执行命令，可用于调试

命令前添加“-”，可以忽略执行错误，例如mkdir，如果存在，就会报错

make 加上“-i” 或 “—ignore-errors”，makefile会忽略所有错误。”.IGNORE”作为目标，这个规则中的所有命令都会忽略错误。

“-k” 或 “—keep-going”，意思是如果规则命令出错，终止该规则执行，但继续执行其他规则。

### 嵌套makefile

在嵌套makefile中，可以使用export <variable …>将变量传递到下级makefile

unexport <variable …> 不传递到下级makefile

“-C” 加下级makefile会指定下级makefile编译

嵌套makefile中比较有用的参数是”-w”，会自动打印运行的哪个makefile，如下

进入：

make: Entering directory `/home/hchen/gnu/make'.

离开：

make: Leaving directory `/home/hchen/gnu/make'

“-s”(“--slient”)或是“—no-print-directory”，“-w”总是失效的。

定义命令包

定义命令序列以”define”开始，以”endef”结束

define run-yacc  
yacc $(firstword $^)  
mv y.tab.c $@  
endef

### 变量

变量中的变量

foo = $(bar)  
bar = $(ugh)  
ugh = Huh?

可以这样先使用，后定义，但这样使用可能会引起递归定义

A = $(B)

B = $(A)

使用:=操作符，前面的变量不能使用后面的变量，只能使用前面定义好的

y := $(x) bar

x := foo

y值是”bar”，而不是”foo bar”

MAKELEVEL会记录当前makefile的调用层数

定义一个变量为空格的方法：

nullstring :=

space := $(nullstring) # end of the line

$(nullstring)用于标明变量值开始，”#”标明变量值结束

?=操作符是如果被定义过，将什么也不做，如果没有定义则定义

FOO ?= bar

等价于

ifeq ($(origin FOO), undefined)

FOO = bar

endif

变量高级用法

“$(var:a=b)” 或是 “${var:a=b}”，把变量”var”中所有以”a”字串结尾的”a”替换成”b”字串

foo := a.o b.o c.o

bar := $(foo:.o=.c)

把foo中以.o为结尾的都替换成.c

bar := $(foo:%.o=%.c) 用静态模式也能达到此效果

x = y

y = z

a := $($(x))

于是，$(a)的值就是z

追加变量

追加变量使用”+=”操作符

override指示符

如果有变量是通过make命令参数设置的，makefile对这些变量的赋值会忽略。如果想在makefile中设置这类参数，需要使用”override”指示符。

override <variable> = <value>  
override <variable> := <value>

也可以写在define指示符前

override define foo

bar

endef

多行变量

define two-lines

echo foo

echo $(bar)

endef

环境变量

make时系统环境变量会载入到makefile中

如果环境变量中设置了”CFLAGS”环境变量，我们的所有makefile都可以使用这个变量

如果makefile中定义了CFLAGS，makefile会使用这个环境变量。

如果make时加”-e”参数，系统环境变量将覆盖makefile中定义的变量

目标变量

prog:CFLAGS=-g

不管全局的$(CFLAGS)的值是什么，在prog目标，以及其所引发的规则中，$(CFLAGS)都是-g

模式变量

make的模式至少包含一个”%”，可以用如下方式给以.o结尾的目标定义目标变量

%.o:CFLAGS=-o

判断示例

libs\_for\_gcc = -lgnu  
normal\_libs =  
  
foo: $(objects)  
ifeq ($(CC),gcc)  
$(CC) -o foo $(objects) $(libs\_for\_gcc)  
else  
$(CC) -o foo $(objects) $(normal\_libs)  
endif

根据$(CC)是否是gcc来进行编译

### 函数

$(<function> <arguments> )  
或是  
${<function> <arguments>}

示例，将空格替换成逗号：

comma:= ,  
empty:=  
space:= $(empty) $(empty)  
foo:= a b c  
bar:= $(subst $(space),$(comma),$(foo))

$(bar)的值是a,b,c

字符串处理函数

$(subst <from>,<to>,<text> )

名称：字符串替换函数——subst。  
功能：把字串<text>中的<from>字符串替换成<to>。  
返回：函数返回被替换过后的字符串。

$(patsubst <pattern>,<replacement>,<text> )   
名称：模式字符串替换函数——patsubst。  
功能：查找<text>中的单词（单词以“空格”、“Tab”或“回车”“换行”分隔）是否符合模式<pattern>，如果匹配的话，则以<replacement>替换。这里，<pattern>可以包括通配符“%”，表示任意长度的字串。如果<replacement>中也包含“%”，那么，<replacement>中的这个“%”将是<pattern>中的那个“%”所代表的字串。（可以用“/”来转义，以“/%”来表示真实含义的“%”字符）  
返回：函数返回被替换过后的字符串。

$(strip <string> )  
名称：去空格函数——strip。  
功能：去掉<string>字串中开头和结尾的空字符。  
返回：返回被去掉空格的字符串值。

$(filter <pattern...>,<text> )  
名称：过滤函数——filter。  
功能：以<pattern>模式过滤<text>字符串中的单词，保留符合模式<pattern>的单词。可以有多个模式。  
返回：返回符合模式<pattern>的字串。

$(filter-out <pattern...>,<text> )  
名称：反过滤函数——filter-out。  
功能：以<pattern>模式过滤<text>字符串中的单词，去除符合模式<pattern>的单词。可以有多个模式。  
返回：返回不符合模式<pattern>的字串。

$(firstword <text> )  
名称：首单词函数——firstword。  
功能：取字符串<text>中的第一个单词。  
返回：返回字符串<text>的第一个单词。  
示例：$(firstword foo bar)返回值是“foo”。  
备注：这个函数可以用word函数来实现：$(word 1,<text> )。

# 备注

## 储备知识点

### 数字签名简单理解：

发送方使用对原文进行哈希函数形成摘要，使用私钥进行加密。将公钥及数据发送给接收方。接收方使用公钥进行解密，并对原文使用相同的哈希函数，比较前后哈希值，相同则视为验证成功。

## 兴趣

charles监听软件，有兴趣可以学习下

## 计划

Libcurl学习

libev

github 上传

## github账号

[1\*\*\*\*\*\*\*\*4@qq.com](mailto:1********4@qq.com)

S\*\*\*\*\*\*\*\*6

<https://github.com>