.



**花生壳动态域名SDK**

**Peanuthull dynamic DNS**

上海贝锐信息科技有限公司

**目录**

[1. 概述 2](#_Toc11269)

[2. 官方认证 2](#_Toc29268)

[3. 关于官方认证Key的说明 3](#_Toc25767)

[4. 示例源码 3](#_Toc30225)

[5. 源码说明 6](#_Toc5209)

[6. 关于客户端界面 9](#_Toc6934)

[7. DDNS协议说明 11](#_Toc29058)

[7.1. TCP认证部分 11](#_Toc26631)

[7.1.1. 账号认证 11](#_Toc10766)

[7.1.2. 服务器跳转 15](#_Toc31328)

[7.1.3. 注册域名 16](#_Toc28835)

[7.1.4. 获取域名信息 17](#_Toc6369)

[7.1.5. 获取用户信息 18](#_Toc32005)

[7.2. UDP保持部分 19](#_Toc52)

[7.2.1. 心跳包协议 19](#_Toc21826)

[7.2.2. 维持客户端在线 24](#_Toc12793)

[7.2.3. 获得客户端IP端变更 24](#_Toc24864)

[7.2.4. 注销登陆 24](#_Toc14585)

# **概述**

花生壳是一套完全免费的桌面式域名管理和动态域名解析（ DDNS ）等功能为一体的客户端软件。花生壳客户端向用户提供全方位的桌面式域名管理以及动态域名解析服务。用户无需通过 IE 浏览器，直接通过客户端使用 [http://www.oray.com](http://www.oray.com/" \o "http://www.oray.com) 所提供的动态域名服务，包括用户注册、域名查询、域名管理、 IP 工具以及自诊断等各种服务；且通过树状结构方式可使用户对多达上百个域名进行方便管理，亦可自主添加二级域名，自由设置 A 记录（ IP 指向）、 MX记录、 CName（别名）、 URL重定向等，用户操作界面清晰简单。

花生壳动态 DDNS 服务支持包括Modem、ISDN 、ADSL、有线电视网络、双绞线到户的宽带网和其他任何能够提供互联网真实 IP 的接入服务线路，无论连接获得的 IP 属于动态还是静态，都可根据自己的需求选择合适的系统平台、数据库平台以及站点运营模式，并且可避免在转换服务商时，因受制域名解析服务商而忍受效率低下的修改过程，全面利用花生壳来建立拥有自主域名和最大自主权的互联网主机。

以下为花生壳嵌入式开发所需要的相关资源，通过开发，您可以完成如同官方花生壳客户端完全一样的功能，甚至将花生壳动态域名嵌入您的软件、路由器乃至各种网络设备。

# **官方认证**

获得官方认证的软硬件将被列入官方支持清单：[支持清单](http://www.oray.com/peanuthull/embed_company.php)

具体步骤参见：[花生壳嵌入申请](http://developer.oray.com/)

获得官方认证前，您将需要签订相关合作协议。

# **关于官方认证Key的说明**

您在对产品进行正式的发布前，需要执行以下步骤：

## 从[oray开发者平台](http://developer.oray.com/) 注册开发者，获得App ID、App Secret、App Version：

* App ID - 2个字节
* App Version -2个字节
* App Secret -4个字节

注：网站上申请得到的App ID、App Secret、App Version是十进制

## 服务器地址

Phddns60.oray.net

## App ID与App Version组合到4个字节的整数,得到clientinfo

如0xFFFF为App ID，0xEEEE为App Version，则组合后为：0xFFFFEEEE

## 用3.2步骤得到的clientinfo和App Secret 提交验证。（示例源码中的PH\_EMBED\_CLIENT\_INFO和PH\_EMBED\_CLIENT\_KEY）

# **示例源码**

我们提供一套已实现所有协议的规范代码，您可以进行任意传播与使用，没有任何协议限制。如果你发现本套代码中存在需要修复的问题，请发送Email到:open@oray.com

## **源码清单**

|  |  |
| --- | --- |
| **文件** | **说明** |
| main.c | 主入口函数，一般修改该文件和phkey.h里的值就行了 |
| Makefile.am | Linux编译Makefile |
| PHGlobal.c | 全局变量及公用函数实现 |
| phkey.h | Key值定义 |
| PHGlobal.h | 程序用到的数据结构的定义 |
| phupdate.c | PHDDNS Embed网络通信主过程 |
| phupdate.h |
| base64.c | 通用加密算法相关代码 |
| bitstream.c |
| bitstream.h |
| blowfish.c |
| blowfish.h |
| global.h |
| lutil.h |
| md5.c |
| md5.h |
| generate.c | DDNS嵌入式私有专用加密算法实现，用于TCP主认证过程，以及每个心跳包的加解密 |
| generate.h |
| log.c | 日志实现 |
| log.h |
| PHSocket.c | SOCKET封装，支持TCP/UDP |
| PHSocket.h |

## **编译**

### **windows编译**

源码压缩包内包含VC6工程文件，可在Windows上编译测试，无其他依赖项

### **Linux(X86, ARM, MIPS)**

### 编译过程：

1、解压缩

#tar zxvf phddns-2.0.6.32828.tar.gz

# cd phddns-2.0.6.32828

2、编译前configure为当前平台编译

#./configure

3、编译最终可执行文件

#make

ARM与MIPS交叉编译：您需要首先安装相关的交叉编译器，相关工具链的安装请参考开发板文档，并执行类似以下的编译前configure

#./configure --host=i386-linux --target=arm-linux--build=arm-linux CXX=arm-linux-c++ CC=arm-linux-gcc LD=arm-linux-ld

### **运行**

* 直接运行，根据提示进行操作
* featured.exe -h 有参数说明

# **源码说明**

我们提供的源码封装了DDNS协议的解析，对外暴露结构PHGlobal，只要往结构里填入相关信息，即可实现一个简单的客户端。PHGlobal结构说明参考源码里的phglobal.h文件

例子：简单客户端

#include <stdio.h>

#include <signal.h>

#include "phruncall.h"

#include "phupdate.h"

#include "log.h"

#include "phkey.h"

PHGlobal global;

PH\_parameter parameter;

static void my\_handleSIG (int sig)

{

if (sig == SIGINT)

{

printf ("signal = SIGINT\n");

phddns\_stop(&global);

exit(0);

}

if (sig == SIGTERM)

{

printf ("signal = SIGTERM\n");

phddns\_stop(&global);

}

signal (sig, my\_handleSIG);

}

//状态更新回调

static void myOnStatusChanged(PHGlobal\* global, int status, long data)

{

printf("myOnStatusChanged %s", convert\_status\_code(status));

if (status == okKeepAliveRecved)

{

printf(", IP: %d", data);

}

if (status == okDomainsRegistered)

{

printf(", UserType: %d", data);

}

printf("\n");

}

//域名注册回调

static void myOnDomainRegistered(PHGlobal\* global, char \*domain)

{

printf("myOnDomainRegistered %s\n", domain);

}

//用户信息XML数据回调

static void myOnUserInfo(PHGlobal\* global, char \*userInfo, int len)

{

printf("myOnUserInfo %s\n", userInfo);

}

//域名信息XML数据回调

static void myOnAccountDomainInfo(PHGlobal\* global, char \*domainInfo, int len)

{

printf("myOnAccountDomainInfo %s\n", domainInfo);

}

int main(int argc, char \*argv[])

{

void (\*ohandler) (int);

WORD VersionRequested;

WSADATA WsaData;

int error;

VersionRequested = MAKEWORD(2, 0);

WSAStartup(VersionRequested, &WsaData);

ohandler = signal (SIGINT, my\_handleSIG);

if (ohandler != SIG\_DFL) {

printf ("previous signal handler for SIGINT is not a default handler\n");

signal (SIGINT, ohandler);

}

init\_global(&global);

global.cbOnStatusChanged = myOnStatusChanged;

global.cbOnDomainRegistered = myOnDomainRegistered;

global.cbOnUserInfo = myOnUserInfo;

global.cbOnAccountDomainInfo = myOnAccountDomainInfo;

set\_default\_callback(&global);

strcpy(global.szHost,"phddns60.oray.net");

strcpy(global.szUserID,"youname");

strcpy(global.szUserPWD,"youpassword");

global.clientinfo = PH\_EMBED\_CLIENT\_INFO;

global.challengekey = PH\_EMBED\_CLIENT\_KEY;

for (;;)

{

int next = phddns\_step(&global);

sleep(next);

}

phddns\_stop(&global);

return 0;

}

# **关于客户端界面**

客户端界面一般包含登陆、登录后的状态显示、介绍三个页面。我们提供了一套界面模板供参考。模板代码在源码包UI目录下。

登录界面：



登录后的状态显示界面：



介绍界面：



# **DDNS协议说明**

动态主机注册协议提供一种将域名和动态IP地址绑定的方法，允许客户机将自己的IP地址提交给服务器，并选择希望注册的域名记录，服务器确认客户机拥有该域名的权限，并修改DNS相应记录，在客户机在线的情况下维持这些动态注册的域名，并且在客户机离线自动删除动态注册的域名。

DDNS协议包含TCP认证和UDP保持两部分

## **TCP认证部分**

### **账号认证**

客户机必须通过账号验证才能进行域名申请、修改、注册操作。账号验证必须在服务器的就绪状态下进行。在其他状态下客户机发出账号验证请求服务器都返回 503命令顺序错误的回应。

验证过程如下：

客户机发出AUTH authtype指令，authtype为一种验证类型，我们这里使用

AUTH ROUTER6(CRAM-MD5扩展)验证。 服务器回应一个挑战串，该串使用客户机的IP地址、服务器的当前时间等不可重复不可预测的参数生成，服务器使用BASE64方式将该串返回客户机，回应码为334

“**客户机将挑战串作为key使用CRAM-MD5的单向加密方式加密自己的密码形成加密串，将账号和加密串座位回应，其格式为:“账号名+一个空格+加密嵌入认证码+客户端信息+加密串”。其中加密嵌入认证码工4个字节，为嵌入认证码与服务器当前时间运算得到，服务器时间是挑战串的第六个字节后的4个字节，算法是嵌入认证码与该时间的去翻进或运算后循环右移一定位数：该右移位数是用服务器时间整除30，客户端信息也是4个字节，前两节为嵌入式的客户号，后两位为客户端版本号，客户机使用base64方式对回应进行编码返回服务器。**”

以上说明对应的代码：

int GenerateCrypt(char \*szUser, char \*szPassword, char \*szChallenge64,

long clientinfo, long embkey, char \*szResult)

{

unsigned char szDecoded[256];

unsigned char szKey[256];

unsigned char szAscii[256];

unsigned int nDecodedLen;

long challengetime = 0;

int nMoveBits;

long challengetime\_new = 0;

long a, b, c, d;

unsigned int nKey;

int nUser;

unsigned int nEncoded;

//Base64 解码

nDecodedLen = lutil\_b64\_pton(szChallenge64, szDecoded, 256);

memcpy(&challengetime, szDecoded + 6, 4);

//取反进行或运算

challengetime |= ~embkey;

//得到循环移位位数

nMoveBits = challengetime % 30;

//完成32位的循环位移

a = challengetime << ((32 - nMoveBits) % 32);

b = challengetime >> (((unsigned int )nMoveBits) % 32);

c = ~(0xffffffff << ((32 - nMoveBits) % 32));

d = b & c;

challengetime\_new = a | d;

//KEY-MD5

nKey = KeyMD5Encode(szKey,

(unsigned char\*)szPassword,

strlen((char\*)szPassword),

(unsigned char\*)szDecoded,

nDecodedLen);

szKey[nKey] = 0;

nUser = strlen((char \*)szUser);

memcpy(szAscii, szUser, nUser);

szAscii[nUser] = ' ';

memcpy(szAscii+nUser+1, &challengetime\_new,4);

memcpy(szAscii+nUser+1+4,&clientinfo,4);

memcpy(szAscii+nUser+1+4+4, szKey, nKey);

//base64 编码

nEncoded = lutil\_b64\_ntop((unsigned char \*)szAscii,

nUser + 1 + 4 + 4 + nKey,

szResult,

256);

return nEncoded;

}

服务器验证客户的回应，如果账号和密码错误则返回535，如果正确则返回一个多行回应，第一行为250，以后每行为该账号已经注册的主机记录。如果该账号没有注册任何记录则返回250<CRLF>.<CRLF>。 验证成功后服务器进入验证状态（Auth）。

例子1：成功登陆

C：AUTH ROUTER6

S：334 UghjtYsdyu==

C：awerasdfhjkloiuyqwertyuio

S：250 Auth passed at level <1>

www.my.oray.net

ftp.my.oray.net

.

其中，<1>中的数字表示用户级别：从0开始，分别为：

0 标准级

1 专业级

2 商业级

3 旗舰级

例子2：不成功登陆

C：AUTH ROUTER

S：334 UghjtYsdyu==

C：awerasdfhjkloiujqwertyuio

S：535 Authentication failure.

例子3：验证格式不正确

C：AUTH foobar

S：504 Authentication mechanism unsupported

### **服务器跳转**

在以上的帐户验证过程中，如果服务器发现该帐号不应该在本台服务器登陆，将返回需要跳转字串536 Should Redirection

例子：

C：建立6060连接

S：220 oray.cn DDNS ServerX4 Ready.

C：AUTH ROUTER6

S：334 UghjtYsdyu==

C：awerasdfhjkloiuyqwertyuio

S: 536 Should Redirection To <phddns60std.oray.net>

C：QUIT

S：221 Good bye

此时，客户端应重新连接到：phddns60std.oray.net

### **注册域名**

在验证状态下客户机可以选择注册域名，客户机使用REGI命令注册，最后使用CNFM确认注册操作获取注册码，完成注册工作。一般要求要注册账号下的全部域名，否则不注册的域名无法解析。

例子：

C：REGI A www.my.oray.net

S：250 register successfully

C：CNFM

S：250 00123456 012381090

在一次会话中客户机只能执行一次注册动作，一次注册动作可以注册或撤销注册多个名称。如果客户机在CNFM指令成功后再次使用REGI或CNFM服务器将返回命令顺序出错。

发出CNFM指令后，得到的回应码后的字符串分别为更新协议使用的会话ID和初始序号。如果客户机在一次在线过程中多次连接并注册主机则使用最新的注册码。

* 会话编号为非0的有符号长整数，使用十进制表示。
* 初始序号为非0的无符号长整数，使用十进制表示。

如果没有成功动态注册域名则服务器返回507 no name registered. REGI指令和CNFM指令支持批命令方式发送，也就是客户端可以连续发送若干条REGI指令并以一条CNFM指令结尾。中间使用CRLF隔开，服务器会按顺序批量回应所有的REGI指令和CNFM指令，每个回应同样使用CRLF隔开。

### **获取域名信息**

在验证状态下客户机可以选择获取该帐号的域名清单，命令格式为：STAT DOMAIN\r\n，服务器第一行返回250 DomainInfo OK，后接XML格式的域名清单，以CRLF.CRLF结尾。

例子：

C：STAT DOMAIN

S：250 DomainInfo OK

<domainInfo account='test' login='test'>

<domains><domain>

<DomainName>test.gicp.net</DomainName>

<RegDate>1246419992</RegDate>

<Account>test</Account>

<StatusCode>25</StatusCode>

<RootName>gicp.net</RootName>

<IsFree>1</IsFree>

</domain></domains>

<domainInfo>

C: QUIT

S：221 Good bye

### **获取用户信息**

在验证状态下客户机可以选择获取用户信息，命令格式为：STAT USER\r\n，服务器第一行返回250 Userinfo OK，后接XML格式的用户数据，以CRLF.CRLF结尾。

例子：

C：STAT USER

S：250 Userinfo OK

<userInfo account='test' login='test'>

<ID>88888</ID>

<Account>test</Account>

<Password></Password>

<Email>test@gmail.com</Email>

<RegDate>1246419991</RegDate>

<Credit>0.0</Credit>

<Largess>0.0</Largess>

<IsEnable></IsEnable>

<PHServer></PHServer>

<PWDQuestion></PWDQuestion>

<IsEnterprise>0</IsEnterprise>

<Contactor></Contactor>

<IsMale>0</IsMale>

<Address></Address>

<PostCode>200000</PostCode>

<IDType></IDType>

<IDNumber>310110198807218455</IDNumber>

<Country>cn</Country>

<Province></Province>

<City></City>

<Tel>34333333</Tel>

<ServiceType>0</ServiceType>

<ClientIP>-1301289100</ClientIP>

</userInfo>

C: QUIT

S：221 Good bye

## **UDP保持部分**

在完成TCP登陆和注册过程后，客户端就需要断开TCP连接，进入长期的“心跳保持”状态。 心跳保持协议使用UDP 6060端口，客户机必须按规定的时间隔定时向服务器发送更新数据包，如果服务器在5分钟内没有收到客户机的更新请求将自动删除该客户的所有注册的域名。

### **心跳包协议**

客户端使用TCP过程中后获得的挑战串加密操作码、序号和校验码连同TCP过程中完成域名注册后的会话编号发送给服务器。如果没有错误服务器回应正常UDP\_OPCODE\_UPDATE\_OK，否则服务器回应更新失败信息UDP\_OPCODE\_UPDATE\_ERROR，此时客户端必须重新登录并注册域名。如果客户机3分钟内没有收到服务器得回应则判断网络连接出现问题。但在没有程序错误的时候继续发送更新信息。

所有操作，客户端都发送同样结构与大小的数据包：

struct DATA\_KEEPALIVE

{

//! 会话ID ,注册域名成功后返回的会话ID

int lChatID;

//! 操作码 参考后面的define值

Int lOpCode;

//! 包ID ,注册域名成功后返回的起始序号

//! 客户机每成功发送一次更新信息后，序号加1

Int lID;

//! 校验和 计算方法为：0-（包ID+操作码）

Int lSum;

//! 保留字，目前不使用

Int lReserved;

};

//! 更新包扩展结构，用于服务器返回时IP地址

typedef struct

{

DATA\_KEEPALIVE keepalive;

Int ip;

}DATA\_KEEPALIVE\_EXT;

//! 心跳包更新指令

#define UDP\_OPCODE\_UPDATE\_VER2 0x2010

//! 心跳包服务器返回正常

#define UDP\_OPCODE\_UPDATE\_OK 0x2050

//! 心跳包服务器返回错误

#define UDP\_OPCODE\_UPDATE\_ERROR 1000

//! 心跳包注销登录

#define UDP\_OPCODE\_LOGOUT 11

//! 心跳包加密部分大小

#define KEEPALIVE\_PACKET\_LEN 20

每次发送都需要将该数据报加密，客户端使用TCP过程中后获得的挑战串加密操作码、序号和校验码连同TCP过程中完成域名注册后的会话ID发送给服务器。其中，会话ID和序号为注册域名成功后返回的会话ID和起始序号，客户机每成功发送一次更新信息，序号加1。

校验码计算方法为：0-（包ID+操作码）

加密与发送过程代码段：

BOOL SendKeepAlive(PHGlobal \*phglobal, int opCode)

{

DATA\_KEEPALIVE data;

blf\_ctx blf;

char p1[KEEPALIVE\_PACKET\_LEN],p2[KEEPALIVE\_PACKET\_LEN];

memset(&data,0,sizeof(data));

data.lChatID = phglobal->nChatID;

data.lID = phglobal->nStartID;

data.lOpCode = opCode;

data.lSum = 0 - (data.lID + data.lOpCode);

data.lReserved = 0;

if (!phglobal->bTcpUpdateSuccessed) return FALSE;

LOG(1) ("SendKeepAlive() %d\n",opCode);

InitBlowfish(&blf, (unsigned char\*)phglobal->szChallenge,phglobal->nChallengeLen);

memcpy(p1,&data,KEEPALIVE\_PACKET\_LEN);

memcpy(p2,&data,KEEPALIVE\_PACKET\_LEN);

Blowfish\_EnCode(&blf, p1+4,p2+4,KEEPALIVE\_PACKET\_LEN-4); phSend(phglobal->m\_udpsocket, p2, KEEPALIVE\_PACKET\_LEN,0);

//RecvKeepaliveResponse();

return TRUE;

}

数据包接受与解码过程代码段：

int RecvKeepaliveResponse(PHGlobal \*phglobal)

{

char temp[100];

DATA\_KEEPALIVE\_EXT rdata;

DATA\_KEEPALIVE data; blf\_ctx blf;

char p1[KEEPALIVE\_PACKET\_LEN],p2[KEEPALIVE\_PACKET\_LEN];

if (!phglobal->bTcpUpdateSuccessed) return errorOccupyReconnect;

//prevent the thread to be suspended while waiting for data

if (phDataReadable(phglobal->m\_udpsocket, 0)<=0)

{

return okNoData;

}

if (phReceive(phglobal->m\_udpsocket, temp,sizeof(temp),0)<=0)

return okNoData;

memcpy(&rdata, temp, sizeof(DATA\_KEEPALIVE\_EXT));

data = rdata.keepalive;

InitBlowfish(&blf, (unsignedchar\*)phglobal->szChallenge,phglobal->nChallengeLen);

memcpy(p1,&data,KEEPALIVE\_PACKET\_LEN);

memcpy(p2,&data,KEEPALIVE\_PACKET\_LEN);

Blowfish\_DeCode(&blf, p1+4,p2+4,KEEPALIVE\_PACKET\_LEN-4);

memcpy(&data,p2,KEEPALIVE\_PACKET\_LEN);

phglobal->nStartID = data.lID + 1;

LOG(1) (("RecvKeepaliveResponse() Data comes, OPCODE:%d\n"),data.lOpCode);

if (data.lID - phglobal->nLastResponseID > 3 && phglobal->nLastResponseID != -1)

{

return errorOccupyReconnect;

}

phglobal->nLastResponseID = data.lID;

phglobal->tmLastResponse = time(0);

phglobal->ip = rdata.ip;

if (data.lOpCode == UDP\_OPCODE\_UPDATE\_ERROR) return okServerER;

//if (data.lOpCode == UDP\_OPCODE\_LOGOUT) return okNormal;

return okKeepAliveRecved;

}

### **维持客户端在线**

使用操作码UDP\_OPCODE\_UPDATE\_VER2发送

SendKeepAlive(phglobal, UDP\_OPCODE\_UPDATE\_VER2);

客户端必须按以下时间隔定时向服务器发送心跳包

|  |  |
| --- | --- |
| 用户级别 | 时间间隔（秒） |
| 标准级 | 60 |
| 专业级 | 30 |
| 商业级 | 30 |
| 旗舰级 | 15 |

### **获得客户端IP端变更**

服务器将通过客户端发送的心跳包自动侦测

### **注销登陆**

使用操作码UDP\_OPCODE\_LOGOUT发送

SendKeepAlive(phglobal, UDP\_OPCODE\_LOGOUT);