SAMS 北京科海培训中心

活动目录编程指南

[美] Gi Kirkpatrik 著

谭郁松 张明杰 张志伟 译

清 华 大 学 出 版 社

第 22 章　扩展 LDAP 搜索

LDAP 提供了一种十分灵活的模式扩展 LDAP 函数的功能。这种扩展模式可以让目
供应商在不牺牲与现存的符合标准的 LDAP 客户和应用程序的兼容性的情况下，为自己
目录服务器增加高级的特性。

本章讨论微软公司在活动目录 LDAP 搜索机制方面进行的多方面的增强，同时我将
你显示如何利用这些功能强大的扩展机制。

22.1　使用 LDAP 控制扩展 LDAP 搜索

LDAPv3 增添了一种扩展机制，它允许 LDAP 提供者向 LDAP 增添 LDAP RFC 中
有指定的功能。利用这种机制可以扩展任何 LDAP 操作。

要激活一个 LDAP 扩展，你必须指定一个 LDAP 控制。一个 LDAP 控制是一个数据
构，它包括以下内容：一个惟一的 OID，它标识你希望使用的一个特定的扩展；一个标志
指示是否应用程序认为扩展是重要的；一个基本编码规则（BER，Bas　En　dig Rul）-
用于对扩展所要求的附加参数值进行编码。

22.1.1　客户方和服务器方控制

LDAP 扩展可以是客户方扩展，也可以是服务器方扩展。客户方扩展通过 LDAP 客
库激活某些特殊的行为。是否对引用进行追踪是利用客户方控制对函数进行修改的一个
子。

服务器方控制改变目录服务器的行为。使服务器将已删除的对象作为搜索结果的一
分返回就是激活服务器控制进行功能扩展的一个例子。

22.1.2　LDAPC　tro 结构

LDAP 使用下面的结构将 LDAP 控制信息传递给扩展 LDAP API。表 22.1 显示
LDAPC　tro 结构的元素。

```
Typedef struct ldapc　tro　{
    PCHAR              ldctl_od;
    struct berval      ldctl_value;
    BOOLEAN            ldctl_i ritial;
} LDAPC　tro　,*PLDAPC　tro　;
```

表 22.1　LDAPC　tro 结构中的元素

元素	描述
ldctl_od	一个指向以 NULL 结尾的、标识控制的 OID 字符串的指针。例如
	OID 字符串“1.2.840.113556.1.4.616”代表引用控制

(续表

元素	描述
ldctl_value	一个 LDAP_BERVAL 结构,包含与控制一起传递的数据。数据的式由扩展定义并且利用抽象语法概念 1 (ASN .1) 基本编码规则进编码。如果扩展并不要求附加的数据,将 ldctl_value 的 bv_l 元设置为 0,将 bv_val元素设置为 NULL
ldctl_i riti al	一个布尔标志,指示控制是否是关键的。如果将这个元素设置TRUE,那么扩展后的操作将在下述情况下失败 :扩展不能使用或制存在某些问题。如果指定为 FALSE 并且控制不被支持或者控存在问题,那么操作将缺省为没有扩展行为

注意： 微软公司在平台 SDK 中并没提供它的 BER 编码函数,例如 ber_pri tf(),所以如果你的 LDAP 控制要求附加的数据,那么就需要手工对其编码。你可以查询 ASN .1 中的 X.680 规范,更多了解 BER 编码的信息。

22.1.3 扩展后的 LDAP 函数

如果你要使用一个客户方控制或者服务器方控制,就必须使用特殊的扩展后的 AP数中的某一个。每一个主要的 LDAP 函数都有一个相关的扩展后的函数,它接受一个客方控制数组和一个服务器方控制数组作为参数。表 22.2 列出了这些函数。

表 22.2 LDAP 函数和对应的扩展后的函数

基本函数	扩展后的函数
ldap_s arc ()	ldap_s arc _ext()
ldap_s arc _s()	ldap_s arc _ext_s()
ldap_m dify()	dap_m dify_ext()
ldap_m dify_s()	ldap_m dify_ext_s()
ldap_re am ()	ldap_re am _ext()
ldap_re am _s()	ldap_re am _ext_s()
ldap_add()	ldap_add_ext()
ldap_add_s()	ldap_add_ext_s()
ldap_c pare()	ldap_c pare_ext()
ldap_c pare_s()	ldap_c pare_ext_s()
ldap_de te()	ldap_de te_ext()
ldap_de te_s()	ldap_de te_ext_s()

本章只讨论扩展后的搜索函数,我们将在后面的章节中讨论其他扩展后的函数。

22.1.4 活动目录搜索控制简介

本章将讨论六个活动目录支持的 LDAP 搜索控制。可能也存在其他的控制,但是我能找到这些函数相关的一些信息。

对象被改变后,活动目录将通知你的程序。

已删除项目搜索控制 (de ted ite arc tro) 检索那些已被删除,但是其占位
(plac de) 仍然存在的项目。

安全描述符搜索控制 (s urity de riptor s arc tro) 检索与目录中对象关联
W i do NT 安全描述符。

扩展名字控制 (exte ded nam tro) 检索对象的 GU ID、SID 和对象的可区别名字

分页搜索控制 (paged s arc tro) 提供了一种一次可以检索多个搜索结果的方法
是惟一一种可以超越域控制器在一次搜索操作中返回的项数目的"管理限制"的方法。

最后,排序搜索控制 (s rted s arc tro) 按照排序后的顺序返回搜索结果。

这些控制中的一些控制——例如,排序搜索控制和分页搜索控制——是由 IE
Intern t 草案所描述的。其他一些控制,例如安全描述符搜索控制,是活动目录所特有的

22.1.5　获取目录改变通知信息

通知控制可能是微软公司对 LDAP 提供的一个最有用的扩展。通知控制可以让你发
一个搜索操作,然后它不像一般的搜索函数那样返回满足搜索过滤器中指定条件的对象
返回目录中已经改变的对象。这种特性可以让你自己编写应用程序向目录服务器注册改
通知行为,这样当一个特定的对象或者容器改变时就可以执行有意义的操作。

注意: 活动目录通知搜索控制提供了描述在 Intern t 草案 draft-i tf-ldapext-
ps arc -00.txt 中的持久搜索控制功能的一个子集。

注意: 通知搜索控制只能使用 ldap_s arc _ext() 与异步搜索一起工作。

下面我们解释带通知控制的扩展搜索调用时的控制流程 (参考图 22.1 中的可视化
释) 。

1. LDAP 程序向服务器发送一个搜索请求。搜索请求包括一个基本的搜索起始 DN
2. 活动目录服务将搜索请求添加到与搜索请求中指定的 DN 关联的注册列表中。每
   个 LDAP 会话缺省情况下可以最多有八个通知搜索。
3. LDAP 程序周期性地使用 LDAP_MSG_ONE 调用 ldap_re ult() 查看是否有结
   一般情况下这个调用将超时。
4. 当活动目录对搜索中指定的对象进行改变后,它检查 LDAP 会话的注册列表,同
   它向被改变对象注册的列表中的每一个注册项发送搜索结果。
5. LDAP 客户库接受搜索结果。
6. LDAP 程序调用 ldap_re ult() ,获得搜索结果并且处理它,就好像是对搜索请求的
   个正常响应。

通知注册仍然有效直到释放 LDAP 会话为此。每一次一旦注册的对象发生了改变
录服务器向客户发送另外一次搜索结果,这样程序就被通知到改变已经发生。

这正如一个微软首席目录工程师所说的那样——通知控制使搜索的功能更为完美。
即使客户方应用程序必须进行调用 ldap_re ult() "轮询"控制信息,这也是一个低成

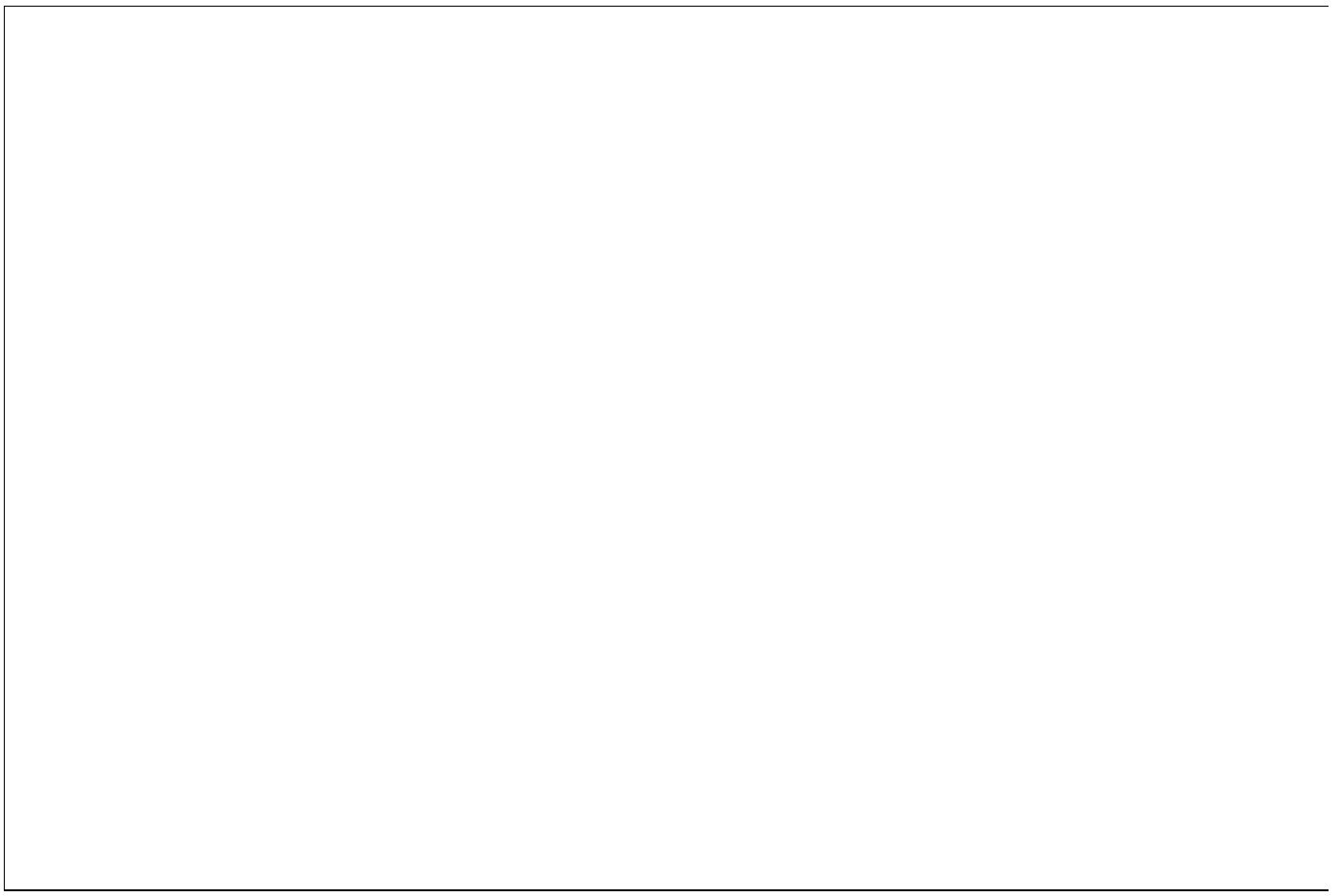

图 22.1 通知控制的搜索流

程序清单 22.1 中的程序向你显示了如何对目录中的一个用户对象的改变进行注册

注意： 只有当发出一个通知搜索请求时,你才可以使用 LDAP_SCOPE_BASE 或者 LDAP_SCOPE_ONELEVEL。当指定的对象发生改变后,使用 LDAP_SCOPE_BASE 产生一个通知。当指定的对象所包含的任何对象发生改变后,使用 LDAP_SCOPE_ONELEVEL 会产生一个通知。目录服务器将返回被修改对象的 DN,这样就可以分辨容器中哪一个特定的对象发生了改变。

程序清单 22.1 中的程序将在当前登录用户所在域的 Users 容器中发出一个通知搜索,每当一个用户对象发生改变后都会打印一条消息。

请你注意,即使改变了另一个域控制器中的一个用户对象,当活动目录将这一改变复制到该程序所连接的域控制器时,该程序将指示改变已经发生。

程序清单 22.1 使用通知搜索控制

```
1:#include <windows.h>
2:#include <stream.h>
3:#include <winldap.h>
4:#include <conio.h>
5:
6:int main(int argc, char* argv[])
7:{
8:    LDAP* pLdap = ldap_init(NULL, LDAP_PORT);
9:    if(pLdap != NULL)
```

```
11:     ULONG uErr=ldap_bind_s(pLdap,NULL,NULL,LDAP_AUTH_NEGOTIATE)
12:     if(uErr == LDAP_SUCCESS)
13:     {
14:       //DN of object we register for change to
15:       char szUsrDN[1024] = "CN=Users,";
16:
17:       LDAPMessage* pMsg = NULL;
18:
19:       //Get root DSE attr defaultNamingContext
20:       uErr = ldap_search_s(pLdap, NULL, LDAP_SCOPE_BASE,
            "objectClass=*", NULL, false, &pMsg);
21:       if(uErr == LDAP_SUCCESS)
22:       {
23:         char** ppszDomain = ldap_get_value(pLdap, pMsg,
              "defaultNamingContext");
24:         if(ppszDomain != NULL)
25:         {
26:           strcat(szUsrDN, ppszDomain[0]);
27:         }
28:         ldap_msgfree(pMsg);
29:       }
30:
31:       //LDAPControl for notification search
32:       LDAPControl control =
33:       {
34:         "1.2.840.113556.1.4.528",  //OID for notification search
35:         {0, NULL},                 //BERval structure, no data
36:         true                       //control critical
37:       };
38:
39:       //array of pointers to server controls
40:       LDAPControl* apsServerControl[] = {&control, NULL};
41:
42:       ULONG uMsgID;
43:
44:       //Issue notification search
45:       uErr = ldap_search_ext(pLdap, szUsrDN, LDAP_SCOPE_ONELEVEL,
            "objectClass=*", NULL, false, apsServerControl,
            NULL, 0, 0, &uMsgID);
46:       if(uErr == LDAP_SUCCESS)
47:       {
48:         //wait for notification
49:         cout << "Waiting for change to " << szUsrDN << ",
              press a key to quit..." << endl;
50:
51:         while(!_kbhit() == 0)
52:         {
53:           LDAP_TIMEVAL sTimeout = {1, 0};
              //old timeout for ldap_result()
54:
55:           //Poll for result
56:           ULONG uResult = ldap_result(pLdap, uMsgID,
                LDAP_MSG_ONE, &sTimeout, &pMsg);
```

```
58:                     {
59:                     cas 0://ti   ut, just c  ti ue
60:                         break;
61:
62:                     cas 0xffff://error, just ign re for n
63:                         c ut << "Error "<< LdapGetLastError() << e dl;
64:                         break;
65:
66:                     cas LDAP_RES_SEARCH_ENTRY:
67:                         c ar* pszDN = ldap_get_dn(pLdap, pM sg);
68:                         c ut << "Objet c anged:"<< pszDN << e dl;
69:                         ldap_m gfre (pM sg);
70:                         break;
71:                     }
72:                 }
73:             }
74:         }
75:
76:          (vod) ldap_unbi d(pLdap);
77:    }
78:
79:     return 0;
80:}
```

程序首先调用 ldap_i t() 初始化一个 LDAP 会话,服务器名字参数值为 NULL,这程序就连接至 DSA。这将选择一个域控制器,它位于登录用户所在域中。程序接着调 ldap_bi d_s() ,其中的用户名参数和口令参数均为 NULL,这表明该函数将使用登录用的证书。通过调用该函数,程序就完成了对目录的认证。

程序接下来在第 20 行读入 rotDSE 项并且获得域控制器所在域的 defaultNam gC text 属性。

程序在第 29 行构造了那个域中 Usrs 容器的可区别名字。

程序在第 37 行初始化代表通知搜索控制的 LDAPC tro 结构,然后在第 45 行将该构的地址放入 apsServerC tro 数组。

程序在第 50 行通过调用 ldap_s arc _ext() 发出异步扩展搜索。

程序在第 56 行启动轮询循环。程序以 1 秒的超时值重复调用 ldap_re ult() ,目的是了查看是否通知搜索已返回结果。

第 62 行的 s tc 语句开始处理任何返回结果。第 71 行的 cas LDAP_RES_SEARCH_ENTRY 打印被修改对象的 DN。

22.1.6 搜索已删除对象

当你从活动目录中删除一个对象时,活动目录实际上并未从目录中将对象移走。活动目录将被删除对象移动到一个已删除对象容器中,该容器位于包含该对象的命名上下文的部。活动目录接着通过将该对象的 iDe ted 属性设置为 TRUE 而标记该对象已删除。在象被删除 60 天后,活动目录就真正将对象删除。这 60 天的时间间隔确保复制已经正常运同时确保所有副本与被删除对象保持一致。

你可以列出已删除对象容器中的内容并且通过使用“显示已删除对象控制”甚至可以复已删除对象并且将它们恢复至目录中（有一些限制）。已删除对象容器中对象的可区别字不是对象最初的可区别名字，活动目录将可区别名字改变为如下格式：

```
<old RDN>烟<new RDN> 'DEL:'<objectGUID>',CN=Deleted objects,<NC Name>
```

<old RDN>为对象的旧的相对可区别名字。

<objectGUID>为标识对象的 GUID——换句话讲，是 objectGUID 属性的值。

<NC Name>为包含对象的 NC 的名字。

例如，如果我们删除了对象 CN=Bob Key,CN=Users,DC=magcorp,DC=c 删除对象容器中该对象的可区别名字将变为：

```
CN=Bob Key烟<new RDN>DEL:ddbd7037-11d3-a0c6-006097311f72,
  CN=Deleted Objects,DC=magcorp,DC=c
```

请你注意，最初的可区别名字并不出现在已删除对象中，没有办法修复对象的最初可别名字。

“显示已删除对象控制”可以让搜索操作符（可能有其他一些扩展 LDAP 操作，但是没有尝试过）对仍旧位于已删除对象容器中的已删除对象进行操作。

“显示已删除对象控制”的 OID 为 1.2.840.113556.1.4.417，它并不要求你在 LDAP 控制中的 BER 结构中提供任何附加的数据。

程序清单 22.2 列出了域命名上下文的已删除对象容器中的项目。

程序清单 22.2　使用已删除项目搜索控制

```
1：#include <windows.h>
2：#include <stream.h>
3：#include <winldap.h>
4：
5：int main(int argc, char* argv[])
6：{
7：    LDAP* psLdap = ldap_init(NULL, LDAP_PORT);
8：    if(psLdap != NULL)
9：    {
10：        ULONG uErr=ldap_bind_s(psLdap,NULL,NULL,LDAP_AUTH_NEGOTIATE)
11：        if(uErr == LDAP_SUCCESS)
12：        {
13：            //DN of objectwe getdeleteditemsf
14：            charszCattrDN[1024] = "CN=Deleted Objects,";
15：
16：            LDAPMessage* psMsg = NULL;
17：
18：            //Get rootDSE attr defaultNamingContext
19：            uErr = ldap_search_s(psLdap, NULL, LDAP_SCOPE_BASE,
                  "objectClass=*", NULL, false, &psMsg);
20：            if(uErr == LDAP_SUCCESS)
21：            {
22：              LDAPMessage *pEntry=ldap_first_entry(psLdap,psMsg);
```

```
24:             {
25:               char** ppszDoai = ldap_get_value (psLdap, psEntry,
                    "defaultNamingContext");
26:               if(ppszDoai != NULL)
27:               {
28:                   strcat(szCtairDN, ppszDoai[0]);
29:                   ldap_value_free (ppszDoai);
30:               }
31:             }
32:               ldap_msgfree (psMsg);
33:           }
34:
35:           //LDAPCtro for dedite    arc
36:           LDAPCtro    tro =
37:           {
38:               "1.2.840.113556.1.4.417",   //OID for dedite    arc
39:               {0, NULL},                  //BERval tructure, n data
40:               true                        //ctro    ritial
41:           };
42:
43:           //array of pointers to rverctro
44:           LDAPCtro* apsServerCtro[] = {&stro, NULL};
45:
46:           //Isuededite    arc
47:           uErr = ldap_sarch_ext_s(
48:               psLdap,                     //s
49:               szCtairDN,          //baseforsarch
50:               LDAP_SCOPE_ONELEVEL,//spe
51:               "objectClass=*",  //fiter
52:               NULL,                   //attrs to get
53:               fal ,                     //attrs  y?
54:               apsServerCtro ,     //srverctro
55:               NULL,                   //c tctro
56:               0,                        //ti    t
57:               0,                        //sze   t
58:               &psMsg                    //reults
59:           );
60:
61:           if(uErr == LDAP_SUCESS)
62:           {
63:               for(LDAPMeage* pEntry = ldap_firt_entry(psLdap, psMsg);
64:                   pEntry != NULL;
65:                   pEntry = ldap_nxt_entry(psLdap, pEntry))
66:               {
67:                   char* pszDN = ldap_get_dn(psLdap, pEntry);
68:                   cout << "DN:" << pszDN << endl;
69:               }
70:
71:                       ldap_msgfree (psMsg);
72:           }
73:       }
74:
75:       (void)ldap_unbind(psLdap);
```

```
77：
78：    return 0；
79：}
```

程序在第 9～12 行初始化 LDAP 会话，同时向登录用户所在域中的一个域控制器进
认证。

程序在第 14 行初始化已删除对象容器的可区别名字，第 19～33 行使用来自 ro tt
的 defaultNamingContext 属性构造完整的 DN。

程序在第 21～30 行从 rootDSE 获取域控制器的域名。

程序在第 36～41 行为“显示已删除项目控制”初始化 LDAPControl 结构，第 44 行创
指向 LDAP 控制结构的指针数组。

程序在第 47 行使用 ldap_search_ext_s()在已删除对象容器之上发出一个同步的一
(Moderve) 搜索。

程序在第 63～73 行枚举位于已删除对象容器中的已删除对象并且显示每一个对象
可区别名字。

程序在第 75 行通过调用 ldap_unbind()结束 LDAP 会话。

22.1.7　为目录对象检索安全描述符

目录中的每一个对象都有一个属性，该属性包含它的安全描述符，但是在缺省情况
活动目录并不为任何搜索返回这个属性。如果你希望检查或者操作对象的安全描述符，那
就必须使用 LDAP 搜索操作中的安全描述符控制。

安全描述符控制的 OID 为 1.2.840.113556.1.4.801，它要求一个 BER 编码的位掩
以指示要检索安全描述符的那一部分 (所有者安全 ID、组安全 ID、自选访问控制列表、系
访问控制列表)。参考第 4 章“活动目录安全”，以更多地了解 Windows NT 安全描述符以
如何操纵它们。

对该通知必须指定的附加数据值为一个 32 位的整型位掩码，它包含任意一个或者所
下面四个设置中的一个 (它们定义在 winnt.h 头文件中)。表 22.3 讨论了可以使用的不同
位值。

表 22.3　访问活动目录安全描述符组件对应的四个常量

名字	值	描述
OWNER_SECURITY_INFORMATION	0X00000001	从安全描述符中检索所有者的 SID
GROUP_SECURITY_INFORMATION	0X00000002	从安全描述符中检索组 SID
DACL_SECURITY_INFORMATION	0X00000004	从安全描述符中检索 DACL
SACL_SECURITY_INFORMATION	0X00000008	从安全描述符中检索 SACL

在设置完希望检索安全描述符的哪一部分所对应的位掩码值后必须使用 ASN.1 基
编码规则对位掩码进行编码。微软版本的 LDAP 实施并没有导出执行这种编码任务的函
(在 Windows 2000 测试版本中)，虽然在 Windows 2000 实际发行时可能会提供。下面的

```
#defi  N_SD_BER_BYTES 5
i tBEREn  deSe urityB its(U LO N G uB its, c ar *pBuffer)
{
   *pBuffer++=0x30;     //U n versal, c   tructed s que
   *pBuffer++=0x03;     //U n versal, pri  tive, bitstri g
   *pBuffer++=0x02;     //Tw     te t o tets
   *pBuffer++=0x00;     //Zero unus d bits
   *pBuffer=uB its &0x0f;  //Th  value
   return N_SD_BER_BYTES;  //l gth  f th  BER e  dig
}
```

程序清单 22.3 中的程序从域命名上下文的根对象中读取安全描述符，然后将安全描述符转换为字符串格式最后将其打印出来。

程序清单 22.3 使用安全描述符搜索控制

```
1：#i  ude <w  do  .h>
2：#i  ude <i  tream .h>
3：#i  ude <w   dap.h>
4：
5：//sddl      tly W i 2000 o  y
6：#defi  _W IN32_W INNT 0x0500
7：#i  ude <sddl.h>
8：
9：#defi  N_SD_BER_BYTES 5
10：
11：//fun ti  to BER-e  de a bitstri g
12：i tBEREn  deSe urityBits(U LO N G uB its, c ar* pBuffer)
13：{
14：    *pBuffer++ = 0x30;    //U n versal, c  tructed s que
15：    *pBuffer++ = 0x03;    //U n versal, pri  tive, bitstri g
16：    *pBuffer++ = 0x02;    //Tw     te t o tets
17：    *pBuffer++ = 0x00;    //Zero unus d bits
18：    *pBuffer = uB its & 0x0f;//Th  value
19：
20：    return N_SD_BER_BYTES;//l gth  f th  BER e  dig
21：}
22：
23：i t m ai (i t argc, c ar** argv)
24：{
25：    LDAP* psLdap = ldap_i t(NULL, LDAP_PORT) ;
26：    if(psLdap != NULL)
27：    {
28：        ULONG uErr=ldap_bi d_s(psLdap,NULL,NULL,LDAP_AUTH_NEGOTIATE)
29：        if(uErr == LDAP_SUC  ESS)
30：        {
31：            //DN of obje t w      regi ter for c ange  to
32：            c ar szDo  ai DN [1024] = {0} ;
33：
34：            LDAPM e  age* psM sg = NULL ;
35：
36：            //Get ro tD SE attr defau ltN am  gC  text
```

```
38:             psLdap,                 //s
39:             NULL,               //bas
40:             LDAP_SCOPE_BASE,//s  pe
41:             "obje tC as =*",//fiter
42:             NULL,               //attrs
43:             fal ,                   //attrs  y?
44:             &psM sg                 //re ults
45:         ) ;
46:         if(uErr == LDAP_SUC  ESS)
47:         {
48:             LDAPM e  age *pEntry=ldap_frit_e try(psLdap,psM sg) ;
49:             if(pEntry !=NULL)
50:             {
51:             c ar** ppszDo  ai  = ldap_get_value (psLdap, psEntry,
                  "defaultNam  gC  text") ;
52:             if(ppszDo  ai  != NULL)
53:             {
54:                 strcat(szDo  ai DN, ppszDo  ai [0]) ;
55:                 ldap_value_fre (ppszDo  ai) ;
56:             }
57:         }
58:             ldap_m  gfre (psM sg) ;
59:         }
60:
61:         c ar acBERBuf[N_SD_BER_BYTES] ;
62:         ULONG dwInfo = OWNER_SECURITY_INFORMATION
63:                     燸GROUP_SECURITY_INFORMATION
64:                     燗DA  L_SECURITY_INFORMATION
65:                     燭SA  L_SECURITY_INFORMATION ;
66:
67:         BEREn  deSe urityBits(dwInfo, acBERBuf) ;
68:
69:         //LDAPC  tro  for s urity de  riptor
70:         LDAPC  tro     tro  =
71:         {
72:             "1.2.840.113556.1.4.801", //0 ID for SD ite     arc
73:             {
74:                 N_SD_BER_BYTES,       //num ber of byte    ber val
75:                 acBERBuf              //ber e   ded data
76:             },
77:             true                      //c  tro    riti al
78:         } ;
79:
80:         //array of po  ters to  rver c  tro
81:         LDAPC  tro * apsServerC  tro [] = {&s   tro, NULL} ;
82:
83:         //Is ue  arc
84:         uErr = ldap_s arc _ext_s(
85:             psLdap,                 //s
86:             szDo  ai DN,        //bas
87:             LDAP_SCOPE_BASE,//s  pe
88:             "obje tC as =*",//fiter
89:             NULL,               //attrs
```

```
91:                 apsServerC  tro , //s rver c  tro
92:                 NULL,             //c  t c  tro
93:                 NULL,             //ti     t
94:                 0,                    //sze   t
95:                 &psM sg
96:             ) ;
97:             if(uErr == LDAP_SUC ESS)
98:             {
99:                 for(LDAPMe age* pEntry = ldap_first_e try(psLdap, psM sg) ;
100:                    pEntry != NULL ;
101:                    pEntry = ldap_n xt_e try(psLdap, pEntry) )
102:                {
103:                    c ar* pszDN = ldap_get_dn(psLdap, pEntry) ;
104:                    c ut << "DN :" << pszDN << e dl;
105:
106:                    struct berval** ppsValue  = ldap_get_value_l  (
                         psLdap, pEntry, "hTSe urityDe  riptor") ;
107:
108:                    if(ppsValue  != NULL)
109:                    {
110:                        for(i tnVal = 0;ppsValue [nVal]  != NULL;nVal++)
111:                        {
112:                            SECURITY_DESCRIPTOR* pSD = (
                                 SECURITY_DESCRIPTOR*)ppsValue [nVal] ->
                                   bv_val;
113:                            c ar* pszSDStri g;
114:                            ULONG uSDStri gLe ;
115:
116:                            C  vertSe urityDe  riptorToStri gSe urity
                                 De  riptor(
117:                                        pSD,    //s urity de  riptor
118:                                        SDDL_REV ISION,  //vers
119:                                        dwInfo,          //bits     ant
120:                                        &pszSDStri g,    //ptr to re ult
121:                                        &uSDStri gLe     //l gth  f re ult
122:                                    ) ;
123:
124:                            c ut << pszSDStri g << e dl;
125:
126:                            LoalFre (pszSDStri g) ;//fre  th  tri g
127:                        }
128:
129:                                ldap_value_fre_l  (ppsValue ) ;
130:                    }
131:                }
132:
133:                ldap_m gfre (psM sg) ;
134:            }
135:        }
136:
137:        (vod) ldap_unbi d(psLdap) ;
138:    }
139:
```

```
141 }
```

第 5～7 行包含了 sddl.h 头文件，该文件包含安全描述符字符串转换例程的声明。注
#defi　_W IN32_W INNT 0x500 是必须的，这是因为 sddl.h 中定义的大多数函数声明
能用于 W ido　2000，定义了该常量后可以使相应的函数声明有效。

第 9～21 行包含利用 BER 对位掩码值进行编码的函数。

程序在 25～28 行初始化 LDAP 会话并且利用 ldap_bid_s() 进行认证。

程序在第 37 行检索域控制器的 ro tDSE 的域名。我们就是要从该对象中检索安全描
符。

程序在 61～67 行使用 BER 对我们请求的安全描述符中的部分进行编码。

程序在第 70～81 行建立 LDAPC　tro 结构和指向控制的指针数组。

第 84～112 行实际执行同步搜索、从目录中检索对象并且从对象中获
nTSe urityDe　riptor 属性。请你注意，这里必须使用 ldap_get_value_l　()，原因在
nTSe urityDe　riptor 属性是二进制的，它不是以字符串方式表达的。

程序在第 106 行获得一个指向安全描述符的指针，在第 111～124 行将安全描述符转
为一个字符串或将其打印出来。

程序在第 126 行通过调用 Lo alFre () 释放与字符串关联的内存。

22.1.8　检索扩展名字信息

活动目录有几种方法在目录中识别对象。可以通过可区别名字、GUID 和 SID（如果
象是一个安全资源用户，如一个用户或一个组）识别对象。第 3 章“构成活动目录的组件
论了活动目录识别对象的所有方法。

扩展名字控制提供了一种一次可以获取所有识别信息的方法，不需要实际检索对象
任何属性。

当你使用扩展名字控制时，活动目录并不返回正常的可区别名字，它按照下述格式返
一个字符串：

```
"<GUID="<objetGUID>"> ;<SID="<objetSID>"> ;"<objetDN>
```

举例如下：

```
<GUID=8c4cdbc0002dd311a0c6006097311f72> ;<SID=0105000000000005150000000f53
  6454245e6975b9755449060000> ;CN=Adm　trator,CN=Us rs,DC=m　gac rp,DC=c
```

如果对象不是一个安全资源用户，也就没有 SID，扩展名字将只包含 GUID 和可区别
字。举例如下：

```
<GUID=2bb226fd5f0ad311995200609779521e> ;CN=Syste　,DC=m　gac rp,DC=c
```

扩展名字控制的 OID 为 1.2.840.113556.1.4.529 并且它并不要求在控制的 BER
构中指定任何数据。

程序清单 22.4 中的程序列出了当前登录用户所在域的 Us rs 容器中所有对象的扩
名字。

程序清单 22.4 使用扩展名字搜索控制

```
1：#i  ude <w  do  .h>
2：#i  ude <w  dap.h>
3：#i  ude <i  tream .h>
4：
5：i t m ai (i t argc, c ar** argv)
6：{
7：    LDAP* psLdap = ldap_i t(NULL, LDAP_PORT) ;
8：    if(psLdap != NULL)
9：    {
10：       ULONG uErr=ldap_bi d_s(psLdap,NULL,NULL,LDAP_AUTH_NEGOTIATE)
11；       if(uErr == LDAP_SUC ESS)
12：       {
13：          //DN ofs arc  bas
14：          c ar szDo  ai DN [1024] = {0} ;
15：
16：          LDAPMe  age* psM sg = NULL ;
17：
18：          //Get ro tDSE attr defaultNam  gC  text
19：          uErr = ldap_s arc _s(
20：              psLdap,                  //s
21：              NULL,              //bas
22：              LDAP_SCOPE_BASE,//s  pe
23：              "obje tC as =*",//fiter
24：              NULL,              //attrs
25：              fal ,                  //attrs  y?
26：              &psM sg                  //re ult
27：          ) ;
28：
29：          //Get defaultdo  ai  am
30：          if(uErr == LDAP_SUC ESS)
31：          {
32：              LDAPMe  age* pEntry = ldap_frit_e try(psLdap,psM sg) ;
33：              if(pEntry != NULL)
34：              {
35：                  c ar** ppszDo  ai = ldap_get_value (psLdap, pEntry,
                        "defaultNam  gC  text") ;
36：                  if(ppszDo  ai  != NULL)
37：                  {
38：                      strcat(szDo  ai DN, ppszDo  ai [0]) ;
39：                      ldap_value_fre (ppszDo  ai) ;
40：                  }
41：              ldap_m gfre (psM sg) ;
42：              }
43：          }
44：          //LDAPC  tro for de  ted ite  arc
45：          LDAPC  tro  tro =
46：          {
47：              "1.2.840.113556.1.4.529", //Exte ded DN OID
48：              {0, NULL} ,                  //BERval tructure, n  data
49：              true                      //c  tro  ritical
50：          } ;
```

```
52:            //array of po  ters to  rver c  tro
53:            LDAPC  tro * apsServerC  tro [] = {&s  tro, NULL};
54:
55:            //Is ue  arc
56:            uErr = ldap_s arc _ext_s(
57:                psLdap,                          //s
58:                szDo  ai DN,                //bas
59:                LDAP_SCOPE_ONELEVEL,//s  pe
60:                "bje tC as =*",  //fi ter
61:                NULL,                        //attrs
62:                fal ,                           //attrs  y?
63:                apsServerC  tro ,       //s rver c  tro
64:                NULL,                        //c  t c  tro
65:                NULL,                        //ti      t
66:                0,                               //sze   t
67:                &psM sg                          //re ults
68:            ) ;
69:            if(uErr == LDAP_SUC  ESS)
70:            {
71:                for(LDAPM e  age* psEntry = ldap_first_e try(psLdap, psM sg) ;
72:                     psEntry != NULL ;
73:                     psEntry = ldap_n xt_e try(psLdap, psEntry) )
74:                {
75:                    c ar* pszDN = ldap_get_dn(psLdap, psEntry) ;
76:                    c ut << pszDN << e dl;
77:                }
78:
79:                ldap_m gfre (psM sg) ;
80:            }
81:        }
82:
83:         ldap_unbi d(psLdap) ;
84:    }
85:
86:    return 0;
87:}
```

程序清单 22.4 中的程序首先通过调用 ldap_i t() 初始化一个 LDAP 会话，然后使用前登录用户的证书对会话进行认证。

程序在第 19～43 行从 ro tDSE 检索域控制器的域名。

程序在第 45～50 行为扩展名字控制初始化 LDAPC tro 结构，在第 53 行初始化 LDAPC tro 指针数组。

在第 56 行，程序使用 LDAPC tro 数组调用 ldap_s arc _ext_s()。在第 71～77 行程序打印由搜索返回的每一个项的扩展可区别名字。

程序在第 79 行通过调用 ldap_m gfre () 释放搜索结果，在第 83 行使用 ldap_unbi 结束 LDAP 会话。

22.2　获取搜索结果——一次一页

最令人恼怒的,就是从目录中要检索的项数大于域控制器的管理限制许可。缺省情况下控制器一次搜索返回的项数最多为 262 144——但是很有可能管理员将该数组设置的更例如 1000。这样造成的后果就是搜索只能返回满足过滤器限制条件的前面 1000 个对象会返回更多的对象。

第二个问题是性能问题。如果你发出了一个将返回大量项的同步搜索请求,相应 LDAP 程序可能要等待几分钟才能得到返回结果。很可能要检索几千个项同时要检索每项的所有属性,如果这样的话,应用程序将会用尽内存。

微软 LDAP 客户库提供了一种分页搜索的功能来避免这些问题。分页搜索与常规的索类似,除了服务器按照相对较小的块(或者称之为页)返回搜索项目之外。执行一个分页索避免了超过服务器上的管理限制的问题,它同时也限制了搜索所要求的客户端内存量

为什么会这样呢?为什么管理限制不应用到分页搜索之上呢?实际上管理限制已应到分页搜索之上。但是,一个分页搜索实际上是一系列较小的搜索,每一个搜索在前一个索结束的地方重新开始搜索。所以,每一个搜索刚好能够满足服务器上的管理限制。

注意: 微软的分页搜索协议实现描述在 Internet 草案 draft-ietf-asid-ldapv3-ispl paged-03.txt 中。

22.2.1 执行分页搜索的两个方法

微软将分页搜索作为对常规 LDAP 搜索的扩展而实施。这个控制的 OID 为 1.2.840 113556.1.4.319,它提供的附加数据如下:包含一个整数的 BER 编码序列,它的含义为每返回的项数;一个空的 8 位组字符串。但是,你自己不能建造扩展搜索,这是因为微软没有供需要初始化 LDAPControl 结构的 BER 编码函数。

幸运的是,微软提供了几个辅助函数可以为我们处理低层的细节。执行分页搜索共需五个函数。你可以用两种方法执行分页搜索。最简单的方法是使用 ldap_search_init_page()、ldap_get_next_page()和 ldap_get_paged_count()函数执行分页搜索。你并不需要把为搜索建立 LDAPControl 结构。

第二种方法有一点麻烦。首先需要使用 ldap_create_page_control()函数为分页搜索立 LDAPControl 结构。然后使用 ldap_search_ext()发出搜索请求,处理返回的结果:返回果中包含一个从服务器返回的 LDAPControl 结构。LDAPControl 结构中的 BER 值包含个代表服务器上搜索状态的 cookie。在下一次分页搜索调用时必须将这个 cookie 传递给务器。整个处理过程有点困难并且容易出错,它不提供辅助函数。我将把 ldap_create_page_control()/ldap_search_ext()方法留给你作为一个练习。

22.2.2 建立分页搜索

```
PLDAPSearch ldap_search_init_page(PLDAP ExternalHandle, LPCTSTR
  DistinguishedName, ULONG ScopeOfSearch, LPCTSTR SearchFilter,
  PCHAR AttributeList[], ULONG AttributesOnly, PLDAPControl *ServerControls,
  PLDAPControl *ClientControls, ULONG PageTimeLimit, ULONG TotalSizeLimit,
  PLDAPSortKey *SortKeys)
```