版本信息

日期	版本	描述	作者
2004-01-15	v1.0	最初版本	LDAPChina.com
2004-01-29	v1.1	修正了某些容易产生歧义的语法错误，以及某些用词错误	LDAPChina.com

The LDAP Application Program Interface

LDAP 应用程序接口

本备忘录的状态（Status of this Memo）

本备忘录为 Internet 通讯提供了信息。本备忘录不指定任何类型的 Internet 标准。这个备忘录的传播是不受限制的。

目　录

The LDAP Application Program Interface LDAP 应用程序接口 2
本备忘录的状态（Status of this Memo）....2
1、介绍（Introduction）....4
2、回顾 LDAP 模型（Overview of the LDAP Model）....4
3、回顾 LDAP API 使用（Overview of LDAP API Use）....4
4、调用 LDAP 操作（Calls for performing LDAP operations）....5
4.1、打开连接（Opening a connection）....5
4.2、向目录进行认证（Authenticating to the directory）....6
4.3、关闭连接（Closing the connection）....7
4.4、查询（Searching）....7
4.5、读取条目（Reading an entry）....9
4.6、列出条目的子条目（Listing the children of an entry）....9
4.7、修改条目（Modifying an entry）....10
4.8、修改条目的 RDN（Modifying the RDN of an entry）....11
4.9、添加条目（Adding an entry）....12
4.10、删除条目（Deleting an entry）....12
5、放弃操作的调用（Calls for abandoning an operation）....13
6、获得结果的调用（Calls for obtaining results）....13
7、错误处理的调用（Calls for error handling）....14
8、解析查询条目的调用（Calls for parsing search entries）....16
8.1、遍历条目集（Stepping through a set of entries）....16
8.2、遍历条目属性（Stepping through the attributes of an entry）....17
8.3、检索属性值（Retrieving the values of an attribute）....17
8.4、检索条目名称（Retrieving the name of an entry）....18
9、安全考虑（Security Considerations）....19
10、感谢（Acknowledgements）....19
11、参考书目（Bibliography）....19
10、作者地址（Authors' Addresses）....20
附录 A、简单的 LDAP API 代码（Sample LDAP API Code）....20

1、介绍（Introduction）

本文档定义了 C 语言的 LDAP 应用程序编程接口。LDAP API 被设计成为既功能强大，又使用简单。它定义了兼容的 LDAP 同步和异步接口，以适应大量各类的应用程序。本文档首先对 LDAP 模型进行简要地回顾，然后再回顾如何使用 API 编程来获得 LDAP 信息。API 调用将详细描述，最后在附录中提供了一些范例代码来说明 API 的使用。

2、回顾 LDAP 模型（Overview of the LDAP Model）

LDAP 是轻型目录访问协议（lightweight directory access protocol），在参考文档[2]和[7]中有对其的描述。它或者可以提供一个轻量级的 X.500 目录的访问前端，或者可以提供一个独立的服务。在两种模型下，LDAP 都基于 C/S 模型，客户端创建到服务器的 TCP 连接，并通过该连接发送请求和接收响应。

LDAP 信息模型基于条目（entry），条目包含某些对象的信息（例如：一个人的信息）。条目由属性（attribute）组成，属性拥有类型和一个或多个值。每个属性都有一个语法，该语法决定哪些值允许出现在该属性中。（例如：ASCII 字符，jepg 图片，等等），并且该语法还决定这些值在目录操作中的行为（例如：在比较操作中区分大小写字母）。

条目被组织为树形结构，通常按照行政区域，地理位置和组织机构划分。每个条目的 RDN（相对分辨名）使该条目在同级条目中拥有唯一的命名，RDN 由条目中一个或多个分辨属性（distinguished attribute）的值组成。每个属性中至多有一个值可以在 RDN 中使用。例如，条目 Babs Jensen 表示一个人，commonName 属性的值可能是“Barbara Jensen”。一个条目的全局性条目名称被称为 DN（分辨名），DN 由一系列从树的根（译者注：根也是一个条目）到该条目的 RDN 连接而成。例如，如果 Babs 在 Michigan 大学工作，她的大学的 DN 条目可能是"cn=BarbarsJensen,o=University of Michigan,c=US"。LDAP 使用的 DN 格式在参考文档[4]中定义。

LDAP 提供了认证，查询信息，修改信息，在树中添加和删除条目操作。下一节将回顾如何使用 API，并且详细描述实现上述功能的 LDAP API 调用方法。

3、回顾 LDAP API 使用（Overview of LDAP API Use）

应用程序通常按以下四个步骤使用 LDAP API：

- 打开到 LDAP 服务器的连接，调用 ldap_open()，返回连接句柄（handle），该函数允许同时打开多个连接。

- 在 LDAP 服务器和（或）X.500 DSA 上进行认证。调用 ldap_bind()及同类函数，它们支持不同的认证方法。
- 执行某些 LDAP 操作并获取相关结果。ldap_search()及同类函数能够返回结果，该结果可以使用 ldap_result2error()，ldap_first_entry()，ldap_next_entry()等函数进行解析。
- 关闭连接。调用 ldap_unbind()来关闭连接。

操作能够同步地（synchronously）或异步地（asynchronously）执行。同步调用以_s 结尾，例如：同步查询可以通过调用 ldap_search_s()来完成。异步查询可以通过调用 ldap_search()进行初始化。所有同步函数都返回一个代表操作结果的指示符。（例如：常量 LDAP_SUCCESS 或者其它错误码）。异步函数返回被初始化了的操作的消息 ID。随后调用 ldap_result()，并在该函数中使用该 ID，这样可以得到操作结果。一个异步操作可以通过调用 ldap_abandon()函数放弃其操作。

返回的结果和错误被封装于一个名为 LDAPMessage 的非透明结构中。有专门的函数对此结构进行解析，对返回的条目和属性执行转换等工作。也有专门解释错误的函数。下一节将更加详细的描述这些函数。

4、调用 LDAP 操作（Calls for performing LDAP operations）

本节将详细描述每个 LDAP 操作的 API 调用。所有调用都取得一个“连接句柄”，该句柄是一个包含每个连接信息的 LDAP structure 指针。许多函数都返回封装于 LDAPMessage 结构中的结果。这些结构或其它结构如有必要将在下面章节中描述。

4.1、打开连接（Opening a connection）

ldap_open()打开一个到 LDAP 服务器的连接。

```
typedef struct ldap {
                        /* ... opaque parameters ... */
                        int     ld_deref;
                        int     ld_timelimit;
                        int     ld_sizelimit;
                        int     ld_errno;
                        char    *ld_matched;
                        char    *ld_error;
                        /* ... opaque parameters ... */
} LDAP;
```

LDAP* ldap_open(char* hostname, int protno);

参数：

hostname：包含一个用空格分开的主机名或主机 IP 地址的列表，用来表示要连接的正正运行的 LDAP 服务器。该函数将按其在列表中的顺序，尝试连接这些主机，并且在第一个连接成功建立后停止；

portno：包含要连接的 TCP 端口号，使用常量 LDAP_PORT，可以获得缺省的 LDAP 端口。

ldap_open()返回一个“连接句柄”，一个指向 LDAP structure（译者注：LDAP structure 是指上面描述的名称为 LDAP 的“结构”数据类型，结构是指 C 语言中的关键词“structure”数据类型，而与 LDAP 协议元素无关）的指针，应该在后续的附属于该连接的调用中使用该句柄。如果连接不能打开将返回 NULL。在该连接上，ldap_bind()的调用必须在所有其它操作执行之前完成。

调用函数的程序应该假设在 LDAP structure 中的域（field）不存在任何顺序。在 LDAP structure 中有许多被内部库使用的域。在上文展示的这些域在必要时会在下面章节中进行描述。

4.2、向目录进行认证（Authenticating to the directory）

ldap_bind()及同类函数用来向目录进行验证。

```
int ldap_bind( LDAP *ld, char *dn, char *cred, int method );
int ldap_bind_s( LDAP *ld, char *dn, char *cred, int method );
int ldap_simple_bind( LDAP *ld, char *dn, char *passwd );
int ldap_simple_bind_s( LDAP *ld, char *dn, char *passwd );
int ldap_kerberos_bind( LDAP *ld, char *dn );
int ldap_kerberos_bind_s( LDAP *ld, char *dn );
```

参数：

ld：连接句柄；

dn：要绑定（bind）到的条目名；

cred：认证的信认方式；

可以使用下列认证方法之一：LDAP_AUTH_SIMPLE，LDAP_AUTH_KRBV41 或 LDAP_AUTH_KRBV42；

passwd：用于 ldap_simple_bind()，该密码将与条目的 userPassword 属性进行比较。

有三种类型的绑定（bind）调用形式，提供简单认证、提供 kerberos 认证的函数，以及两者都提供的通用函数。在 Kerberos 第四版中，认证使用通用 ldap_bind()函数，信认被忽略，因为函数假定存在有效的认证票证（ticket），并且能被用于检索适当的服务票证。

这些函数的同步版本以_s 作为名称的结尾。如果绑定操作成功，这些函数返回常量 LDAP_SUCCESS 作为结果，如果操作失败，则返回 LDAP 错误码。阅读下面关于错误处理的章节可以了解到更多可能出现的错误的信息，以及如何解释他们。

这些函数的异步版本返回已初始化了的绑定操作的消息 ID（message id）。随后通过调用 ldap_result()，可以得到该绑定操作的结果，该函数将在下面章节描述。当发生错误时，这些函数将返回-1，并且对 LDAP structure 中的 ld_errno 域进行适当设置。

值得注意的是，在绑定调用没有成功完成之前，不应该在此连接上尝试任何其它操作。再次进行绑定调用可以用于在同一连接上执行重新认证（re-authenticate）。

4.3、关闭连接（Closing the connection）

ldap_unbind()用于从目录中解除绑定（unbind），并关闭连接。

```
int ldap_unbind( LDAP *ld );
```

参数：

ld：连接句柄。

ldap_unbind()同步地进行工作，从目录中解除绑定，关闭连接，并且在 ldap_unbind()返回 LDAP_SUCCESS（或者返回其它 LDAP 错误码，如果该请求无法被送达 LDAP 服务器）之前释放 ld structure。在 ldap_unbind()调用之后，该 ld 连接句柄将变为无效。

4.4、查询（Searching）

ldap_search()及同类函数用于查询 LDAP 目录，并为每个匹配条目返回被请求的属性集。它有三种形式：

```
struct timeval {
        long    tv_sec;
        long    tv_usec;
};
int ldap_search(
        LDAP    *ld,
        char    *base,
        int     scope,
```

```
                char        *filter,
                char        *attrs[],
                int         attrsonly
        );
        int ldap_search_s(
                LDAP                *ld,
                char                *base,
                int                 scope,
                char                *filter,
                char                *attrs[],
                int                 attrsonly,
                LDAPMessage         **res
        );
        int ldap_search_st(
                LDAP                *ld,
                char                *base,
                int                 scope,
                char                *filter,
                char                *attrs[],
                int                 attrsonly,
                struct timeval      *timeout,
                LDAPMessage         **res
        );
```

参数：

ld：连接句柄；

base：条目的 DN，查询从该条目开始；

scope：表示查询的范围，可以为下列值之一：LDAP_SCOPE_BASE，LDAP_SCOPE_ONELEVEL 或 LDAP_SCOPE_SUBTREE；

filter：描述一个查询过滤器的字符串，该字符串在 RFC 1558 参考文档[3]中描述；

attrs：一个 NULL 结尾的字符串数组，它描述了每个匹配条目所返回的属性。若该参数值为 NULL，则所有有效的属性均被检索；

attrsonly：一个布尔值。如果属性类型和属性值均需要返回，则该参数值为 0；如果仅需返回属性类型，则该参数值为非 0；

timeout：当调用 ldap_search_st()时，该参数指定本地查询的超时值（即客户端指定的超时值）；

res：对于同步调用，在调用结束时，该参数将包含查询的结果。

在 ld 连接句柄中，有三个域用于控制如何执行查询。它们是：

ld_sizelimit：限制查询返回的条目的数量。值为 0 时表示没有限制；

ld_timelimit：限制查询所花费的时间（以秒为单位）。值为 0 时表示没有限制；

ld_deref：指定在查询期间，应该如何处理别名（alias），可以为下列值之一：LDAP_DEREF_NEVER，LDAP_DEREF_SEARCHING，LDAP_DEREF_FINDING 或 LDAP_DEREF_ALWAYS。其中 LDAP_DEREF_SEARCHING 表示在查询期间将别名解除引用（将别名转换为 DN），而在定位查询的基准对象（base object）时，不解除引用。LDAP_DEREF_FINDING 表示在定位基准对象时将别名解除引用（将别名转换成 DN），而在查询期间不解除引用。

异步查询通过调用 ldap_search()函数进行初始化。该函数返回已被初始化的查询的消息 ID（message id）。随后调用 ldap_result()，可以得到查询的结果。该结果可以使用结果解析函数进行解析，结果解析函数将在稍后详述。当发生错误时，这些函数将返回-1，并且对 LDAP structure 中的 ld_errno 域进行适当设置。

同步查询通过调用 ldap_search_s()或者 ldap_search_st()执行。除了 ldap_search_st()多一个附加参数以定义查询超时外，这两个函数是相同的。两个函数都返回查询结果的标识，该标识的值或者为 LDAP_SUCCESS 或者某些错误标识（参阅错误处理一节）。查询返回的条目（如果有返回的条目的话）包含在 res 参数中。该参数对于调用者来说是非透明的。条目，属性，值等，应该通过调用解析函数获得，这些函数将在下面章节描述。如果包含在 res 中的结果不在使用时，应该调用 ldap_msgfree()释放空间，该函数将在下面章节描述。

4.5、读取条目（Reading an entry）

LDAP 不支持直接读取操作。但可以使用查询来模拟这种操作，将要读取的条目的 DN 设置到 base 参数，scope 参数设置为 LDAP_SCOPE_BASE，并且将 filter 参数设置为 "(objectclass=*)"。attrs 参数包含需要返回的属性的列表。

4.6、列出条目的子条目（Listing the children of an entry）

LDAP 不支持直接的列表操作。但可以使用查询来模拟这种操作。将需要列出的条目的 DN 设置到 base，scope 参数设置为 LDAP_SCOPE_ONELEVEL，并且将 filter 参数设置为 "(objectclass=*)"。attrs 参数包含每个子条目需要返回的属性的列表。

4.7、修改条目（Modifying an entry）

ldap_modify()和 ldap_modify_s()函数用于修改一个已存在的 LDAP 条目。

```
typedef struct ldapmod {
        int                     mod_op;
        char                    *mod_type;
        union {
                char                    **modv_strvals;
                struct berval   **modv_bvals;
        } mod_vals;
} LDAPMod;
#define mod_values          mod_vals.modv_strvals
#define mod_bvalues         mod_vals.modv_bvals
int ldap_modify( LDAP *ld, char *dn, LDAPMod *mods[] );
int ldap_modify_s( LDAP *ld, char *dn, LDAPMod *mods[] );
```

参数：

ld：连接句柄；

dn：要修改的条目名称；

mods：条目修改项的 NULL 结尾数组（NULL-terminated array）。

在 LDAPMod structure 中的域有如下意义：

mod_op：要执行的修改操作类型。应该为下列值之一：LDAP_MOD_ADD，LDAP_MOD_DELETE，LDAP_MOD_REPLACE。该域同时也指出了包含在联合（union。译者注：union 是 C 语言中的一种数据类型，请参见上面的 ldapmod structure）mod_vals 中的值的类型。若希望使用 mod_bvalues 形式，则该域需与 LDAP_MOD_BVALUES 进行或操作；否则，将使用 mod_values 形式；

mod_type：要修改的属性的类型；

mod_vals：如果存在的话，则是将要 add，delete，replace 的值。mod_values 和 mod_bvalues 不可同时使用，通过让 mod_op 和 LDAP_MOD_BVALUES 常量进行或操作，来选择应使用哪个。mod_values 是一个 0 结尾（zero-terminated）字符串组成的 NULL 结尾数组，而 mod_bvalues 是一个 berval structure 组成的 NULL 结尾数组，它可以用于传送二进制值，例如：图片等。

对于 LDAP_MOD_ADD 修改操作，给定的值被添加至条目，必要时创建相关的属性。对于 LDAP_MOD_DELETE 修改操作，给定的值将从条目中被删除，如果该值对应的属性已没有任何值，则该属性也被删除。如果希望删除条目的某个属性，mod_vals 域应设为

NULL。（译者注：这里的描述也为多值属性的存在提供了说明，对于单值属性，删除了值，同时也将删除该属性，对于多值属性，只有当删除了所有值时，该属性才将被删除。）对于 LDAP_MOD_REPLACE 修改操作，属性会在修改后有可列表的值，并根据需要创建属性。所有的修改操作按列出的顺序执行。

ldap_modify_s()根据修改操作返回 LDAP 错误码。此代码可以由 ldap_perror()及同类函数进行解释。

ldap_modify()返回由本次调用初始化的请求的消息 ID，或者在出错时返回-1。操作结果可以通过调用 ldap_result()获得。

4.8、修改条目的 RDN（Modifying the RDN of an entry）

ldap_modrdn 和 ldap_modrn_s()函数用于修改 LDAP 条目的名称。

```
int ldap_modrdn(
                    LDAP      *ld,
                    char      *dn,
                    char      *newrdn,
                    int       deleteoldrdn
            );
            int ldap_modrdn_s(
                    LDAP      *ld,
                    char      *dn,
                    char      *newrdn,
                    int       deleteoldrdn
            );
```

参数：

ld：连接句柄；

dn：要修改 RDN 的条目的名称；

newrdn：条目的新 RDN；

deleteoldrdn：布尔值，若非 0 表示删除原 RDN 值，若为 0 表示原 RDN 值应该作为条目的非分辨值保留。

ldap_modrdn_s()为同步函数，返回操作输出的 LDAP 错误码。

ldap_modrdn()为异步函数，返回由本次调用初始化的操作的消息 ID，或者在出错时返回-1。操作结果可以通过调用 ldap_result()获得。

4.9、添加条目（Adding an entry）

ldap_add()和 ldap_add_s()用于向 LDAP 目录添加条目。

```
int ldap_add( LDAP *ld, char *dn, LDAPMod *attrs[] );
int ldap_add_s( LDAP *ld, char *dn, LDAPMod *attrs[] );
```

参数：

ld：连接句柄；

dn：要添加的条目名称；

attrs：条目属性，使用 LDAPMod structure 指定，该 structure 已在 ldap_modify()函数的说明中定义过。mod_type 和 mod_vals 域应该填写，除非需要与常量 LDAP_MOD_BVALUES 进行或操作（用于选择 mod_vals 中的 mod_bvalues 形式），否则 mod_op 域可以被忽略。

注意：要添加的条目的父条目必须存在。

ldap_add_s()为同步函数，返回操作输出的 LDAP 错误码。

ldap_add()为异步函数，返回由本次调用初始化的操作的消息 ID，或者在出错时返回-1，操作结果可以通过调用 ldap_result()获得。

4.10、删除条目（Deleting an entry）

ldap_delete()和 ldap_delete_s()用于从 LDAP 目录中删除条目。

```
int ldap_delete( LDAP *ld, char *dn );
int ldap_delete_s( LDAP *ld, char *dn );
```

参数：

ld：连接句柄；

dn：要删除的条目名称。

注意：要删除的条目必须为叶条目（也就是说，它不能有子条目）。LDAP 不支持删除整个子树的操作。

ldap_delete_s()为同步函数，返回操作输出的 LDAP 错误代码。

ldap_delete()为异步函数，返回由本次调用初始化的操作的消息 ID，或者在出错时返回-1，操作结果可以通过调用 ldap_result()获得。

5、放弃操作的调用（Calls for abandoning an operation）

ldap_abandon()用于放弃一个正在处理的操作。

```
int ldap_abandon( LDAP *ld, int msgid );
```

ldap_abandon()放弃消息 ID 为 msgid 的操作。如果放弃操作成功返回 0，否则返回-1。在成功调用 ldap_abandon()后，给定消息 ID 的结果不会由 ldap_result()调用返回。

6、获得结果的调用（Calls for obtaining results）

ldap_result()用于获得先前已初始化的异步操作的结果。ldap_msgfree()用于释放先前调用 ldap_result()或者同步查询函数获得的结果。

```
int ldap_result(
        LDAP                *ld,
        int                 msgid,
        int                 all,
        struct timeval      *timeout,
        LDAPMessage         **res
);
int ldap_msgfree( LDAPMessage *res );
```

参数：

ld：连接句柄；

msgid：需要返回结果的操作的消息 ID，或者如果所有的结果都希望获得时，该参数可以使用 LDAP_RES_ANY 常量；

all：布尔参数，仅对查询结果有意义。如果非 0 则表示在返回查询结果之前，所有的查询结果都应该已被检索到。如果为 0，一旦检索到一个查询结果（条目），就立刻返回；

timeout：指定等待结果返回的超时时间。该参数设置为 NULL 将导致 ldap_result()一直等待，直到结果可用。timeout 值为 0 秒指定轮询行为（polling behavior）；

res：对于 ldap_result()函数，该参数包含操作结果，对于 ldap_msgfree()函数，该参数包含要被释放的结果链（result chain），该结果链是从先前调用的 ldap_result()，ldap_search_s()或 ldap_seatch_st()中获得的。

在成功完成后，ldap_result()将在 res 参数中返回结果的类型，该类型将是下列常量中的之一：

LDAP_RES_BIND

LDAP_RES_SEARCH_ENTRY
LDAP_RES_SEARCH_RESULT
LDAP_RES_MODIFY
LDAP_RES_ADD
LDAP_RES_DELETE
LDAP_RES_MODRDN
LDAP_RES_COMPARE

如果超时，ldap_result()返回 0，在出错时返回-1。在上述两种情况下，ld structure 的 ld_errno 域将会得到相应的设置。

ldap_msgfree()释放指向 res 的结果，并且返回由该函数释放了的消息的类型。

7、错误处理的调用（Calls for error handling）

下面的调用用于解释由其他 LDAP API 函数返回的错误。

```
int ldap_result2error(
                        LDAP            *ld,
                        LDAPMessage     *res,
                        int             freeit
                );
char* ldap_err2string(int err);
void ldap_perror(LDAP* ld, char* msg);
```

参数：

ld：连接句柄；

res：由 ldap_result()返回的或由一个同步 API 操作调用返回的 LDAP 操作结果；

freeit：布尔参数，表示 res 参数是否应该被释放（非 0 表示释放，0 表示不释放）；

err：LDAP 错误码，该错误码或者由 ldap_result2error()返回，或者由某个同步 API 操作调用返回；

msg：在 LDAP 错误信息之前显示的信息。

ldap_result2error()用于将 LDAP 结果消息（result message）转换为数字形式的错误码，该 LDAP 结果消息或者从 ldap_result()中获得，或者从一个同步 API 操作调用返回的 res 参数中获得。同时该函数也解析结果消息中的 ld_matched 和 ld_error 部分，并且将它们放入连接句柄信息中。所有的同步操作函数在返回结果前调用 ldap_result2error()，可以确保这些域得到正确地设置。连接结构（connection structue）中的相关字段是：

ld_matched：如果返回 LDAP_NO_SUCH_OBJECT 错误，则该参数包含了匹配 DN 的范

围（the extent of the DN matched）；

ld_error：此参数包含了由 LDAP 服务器返回的结果中的错误消息；

ld_errno：表示操作输出的 LDAP 错误码，它的值为下列常量之一：

LDAP_SUCCESS
LDAP_OPERATIONS_ERROR
LDAP_PROTOCOL_ERROR
LDAP_TIMELIMIT_EXCEEDED
LDAP_SIZELIMIT_EXCEEDED
LDAP_COMPARE_FALSE
LDAP_COMPARE_TRUE
LDAP_STRONG_AUTH_NOT_SUPPORTED
LDAP_STRONG_AUTH_REQUIRED
LDAP_NO_SUCH_ATTRIBUTE
LDAP_UNDEFINED_TYPE
LDAP_INAPPROPRIATE_MATCHING
LDAP_CONSTRAINT_VIOLATION
LDAP_TYPE_OR_VALUE_EXISTS
LDAP_INVALID_SYNTAX
LDAP_NO_SUCH_OBJECT
LDAP_ALIAS_PROBLEM
LDAP_INVALID_DN_SYNTAX
LDAP_IS_LEAF
LDAP_ALIAS_DEREF_PROBLEM
LDAP_INAPPROPRIATE_AUTH
LDAP_INVALID_CREDENTIALS
LDAP_INSUFFICIENT_ACCESS
LDAP_BUSY
LDAP_UNAVAILABLE
LDAP_UNWILLING_TO_PERFORM
LDAP_LOOP_DETECT
LDAP_NAMING_VIOLATION
LDAP_OBJECT_CLASS_VIOLATION
LDAP_NOT_ALLOWED_ON_NONLEAF
LDAP_NOT_ALLOWED_ON_RDN
LDAP_ALREADY_EXISTS
LDAP_NO_OBJECT_CLASS_MODS
LDAP_RESULTS_TOO_LARGE
LDAP_OTHER
LDAP_SERVER_DOWN
LDAP_LOCAL_ERROR
LDAP_ENCODING_ERROR
LDAP_DECODING_ERROR

LDAP_TIMEOUT
LDAP_AUTH_UNKNOWN
LDAP_FILTER_ERROR
LDAP_USER_CANCELLED
LDAP_PARAM_ERROR
LDAP_NO_MEMORY

ldap_err2string()用于将一个数字形式的 LDAP 错误码转换成为以 NULL 结尾的包含错误描述的字符串消息，该错误码由 ldap_result2error()函数或者一个同步 API 操作调用返回。它返回静态数据的指针。

ldap_perror()用于将 msg 参数提供的消息打印为标准错误，该参数将位于 ld 连接句柄的 ld_errno 域中的错误声明之后。

8、解析查询条目的调用（Calls for parsing search entries）

下列调用将用于解析由 ldap_search()及同类函数返回的条目。这些条目以一个非透明的结构返回，仅应该通过调用下面所述的函数进行访问。这些函数用于遍历所返回的条目，遍历某条目的属性，检索条目名称，以及检索某条目中给定属性的属性值。

8.1、遍历条目集（Stepping through a set of entries）

ldap_first_entry()和 ldap_next_entry()函数用于遍历查询结果中的条目集。

ldap_count_entries()用来计算返回条目的数量。

```
LDAPMesage *ldap_first_entry( LDAP *ld, LDAPMessage *res );
LDAPMesage *ldap_next_entry( LDAP *ld, LDAPMessage *entry );
int ldap_count_entries( LDAP *ld, LDAPMessage *res );
```

参数：

ld：连接句柄；

res：查询结果，通过调用一个同步查询函数或 ldap_result()获得；

entry：上一个调用 ldap_first_entry()函数或 ldap_next_entry()函数所返回的条目。

当不再存在可以被返回的条目时，ldap_first_entry()和 ldap_next_entry()将返回 NULL。若在遍历条目时发生了错误，也将返回 NULL，这种情况下，ld 连接句柄的 ld_errno 域将被设置用于表示错误码。

ldap_count_entries()返回包含于一个条目链中的条目数量，如果调用此函数时，将 ldap_first_entry()或 ldap_next_entry()所返回的条目作为第二个参数，它也可以用于计算链中

剩余的条目数量。

8.2、遍历条目属性（Stepping through the attributes of an entry）

ldap_first_attribute()和 ldap_next_attribute()函数用于遍历一个条目返回的属性类型列表。

```
char *ldap_first_attribute(
                LDAP            *ld,
                LDAPMessage     *entry,
                void            **ptr
        );
        char *ldap_next_attribute(
                LDAP            *ld,
                LDAPMessage     *entry,
                void            *ptr
        );
```

参数：

ld：连接句柄；

entry：要遍历其属性的条目，该条目由 ldap_first_entry()或 ldap_next_entry()返回；

ptr：在 ldap_first_attribute()中，该参数是一个内部使用的指针地址，该地址追踪条目的当前位置。在 ldap_next_attribute()中，该参数是由先前的 ldap_first_attribute()调用返回的指针。

当到达属性的结尾或者发生错误时，ldap_first_attribute()和 ldap_next_attribute()将返回 NULL，在这种情况下，ld 连接句柄中的 ld_errno 域将被设置用于表示错误码。

这两个函数都返回一个指针，该指针将指向一个包含当前属性名的单连接缓冲区（per-connection buffer）。该指针应该被视为静态数据。ldap_first_attribute()将会分配空间并且在 ptr 参数中返回一个 BerElement 指针，该指针用于追踪当前位置。该指针应该传给后续调用的 ldap_next_attribute()，以便遍历条目属性。

返回的属性名可以传给 ldap_get_values()及同类函数的调用，以便检索相关的值。

8.3、检索属性值（Retrieving the values of an attribute）

ldap_get_values() 和 ldap_get_values_len() 用于从条目中检索给定属性的值。ldap_count_values()和 ldap_count_values_len()用于计算返回值的数量。ldap_value_free()和

ldap_value_free_len()用于释放值。

```
typedef struct berval {
        unsigned long    bv_len;
        char             *bv_val;
};
char **ldap_get_values(
        LDAP             *ld,
        LDAPMessage      *entry,
        char             *attr
);
struct berval **ldap_get_values_len(
        LDAP             *ld,
        LDAPMessage      *entry,
        char             *attr
);
int ldap_count_values( char **vals );
int ldap_count_values_len( struct berval **vals );
int ldap_value_free( char **vals );
int ldap_value_free_len( struct berval **vals );
```

参数：

ld：连接句柄；

entry：需要检索其值的条目，该条目由 ldap_first_entry()或 ldap_next_entry()返回；

attr：需要检索其值的属性，该属性由 ldap_first_attribute()或 ldap_next_attribute()返回，或者是一个由调用者提供的字符串（例如："mail"）；

vals：由先前调用的 ldap_get_values()或 ldap_get_values_len()返回的值。

上面提供了两种形式的调用。第一种形式仅适合于与非二进制字符串数据一同使用，第二种以_len 结尾的形式可以与任何类型的数据一同使用。

应注意，返回的值是使用 malloc 进行空间分配的，所以不再使用时，应该调用 ldap_value_free()或 ldap_value_free_len()释放。

8.4、检索条目名称（Retrieving the name of an entry）

ldap_get_dn()用于检索条目名称。ldap_explode_dn()用于将名称拆分为各个组成部分（component parts）。ldap_dn2ufn()用于将名称转换为更加“用户友好”（user friendly）的格式。

```
char *ldap_get_dn( LDAP *ld, LDAPMessage *entry );
```

```
char **ldap_explode_dn( char *dn, int notypes );
char *ldap_dn2ufn( char *dn );
```

参数：

ld：连接句柄；

entry：需要检索名称的条目，该条目由 ldap_first_entry()或 ldap_next_entry()返回；

dn：要被拆分的 dn，由 ldap_get_dn()返回；

notypes：布尔型参数，如果非 0 表示应该剥去 dn 各组成部分中的类型信息。（也就是说："cn=Babs"应成为"Babs"）。

若在解析 dn 时发生错误，ldap_get_dn()将返回 NULL，这种情况下，ld 连接句柄中的 ld_errno 域将被设置用于表示错误。该函数返回一个指针，该指针使用 malloc 分配空间，当该指针不再使用时，调用者应该调用 free()进行释放。应注意，返回的 DN 的格式在参考文档[4]中给出。

ldap_explode_dn()返回一个 char * 数组，该数组包含 DN 的 RDN 的组成部分，是否带有类型信息，取决于 notypes 参数的设定。当这个返回的数组不再使用时，应该调用 ldap_value_free()释放。

ldap_dn2ufn()将 DN 转换为用户友好的格式（在参考文档[5]中描述）。返回的 UFN 是使用 malloc 进行空间分配的，所以当它不再使用时，应该调用 free()进行释放。

9、安全考虑（Security Considerations）

LDAP 支持在连接认证（connection authentication）期间的最小限度的安全。

10、感谢（Acknowledgements）

本材料基于国家科学基金（National Science Foundation）支持的工作之上，授权号为 NCR-9416667。

11、参考书目（Bibliography）

[1] The Directory: Selected Attribute Syntaxes. CCITT, Recommendation X.520.

[2] Howes, T., Kille, S., Yeong, W., and C. Robbins, "The String Representation of Standard Attribute Syntaxes", University of Michigan, ISODE Consortium, Performance Systems International, NeXor Ltd., RFC 1778, March 1995.

[3] Howes, T., "A String Representation of LDAP Search Filters", RFC 1558, University of Michigan, December 1993.

[4] Kille, S., "A String Representation of Distinguished Names", RFC 1779, ISODE Consortium, March 1995.

[5] Kille, S., "Using the OSI Directory to Achieve User Friendly Naming", RFC 1781, ISODE Consortium, March 1995.

[6] S.P. Miller, B.C. Neuman, J.I. Schiller, J.H. Saltzer, "Kerberos Authentication and Authorization System", MIT Project Athena Documentation Section E.2.1, December 1987

[7] Yeong, W., Howes, T., and S. Kille, "Lightweight Directory Access Protocol," RFC 1777, Performance Systems International, University of Michigan, ISODE Consortium, March 1995.

10、作者地址（Authors' Addresses）

Tim Howes
University of Michigan
ITD Research Systems
535 W William St.
Ann Arbor, MI 48103-4943
USA
Phone: +1 313 747-4454
EMail: tim@umich.edu

Mark Smith
University of Michigan
ITD Research Systems
535 W William St.
Ann Arbor, MI 48103-4943
USA
Phone: +1 313 764-2277
EMail: mcs@umich.edu

附录 A、简单的 LDAP API 代码（Sample LDAP API Code）

```
#include <ldap.h>

main()
{
```

```
LDAP            *ld;
LDAPMessage     *res, *e;
int             i;
char            *a, *dn;
void            *ptr;
char            **vals;

/* open a connection */
if ( (ld = ldap_open( "dotted.host.name", LDAP_PORT ))
        == NULL )
        exit( 1 );

/* authenticate as nobody */
if ( ldap_simple_bind_s( ld, NULL, NULL ) != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_simple_bind_s" );
        exit( 1 );
}

/* search for entries with cn of "Babs Jensen",
        return all attrs    */
if ( ldap_search_s( ld, "o=University of Michigan, c=US",
    LDAP_SCOPE_SUBTREE, "(cn=Babs Jensen)", NULL, 0, &res )
    != LDAP_SUCCESS ) {
        ldap_perror( ld, "ldap_search_s" );
        exit( 1 );
}

/* step through each entry returned */
for ( e = ldap_first_entry( ld, res ); e != NULL;
    e = ldap_next_entry( ld, e ) ) {
        /* print its name */
        dn = ldap_get_dn( ld, e );
        printf( "dn: %s0, dn );
        free( dn );

        /* print each attribute */
        for ( a = ldap_first_attribute( ld, e, &ptr );
                a != NULL;
            a = ldap_next_attribute( ld, e, ptr ) ) {
                printf( "attribute: %s0, a );

                /* print each value */
                vals = ldap_get_values( ld, e, a );
```

```
                        for ( i = 0; vals[i] != NULL; i++ ) {
                                printf( "value: %s0, vals[i] );
                        }
                        ldap_value_free( vals );
                }
        }
        /* free the search results */
        ldap_msgfree( res );

        /* close and free connection resources */
        ldap_unbind( ld );
}
```