版本信息

日期	版本	描述	作者
2003-04-30	v1.0	最初版本	LDAPChina.com
2003-10-17	v1.1	修订，包括错误修正	LDAPChina.com
2003-11-19	v1.2	整体修改	LDAPChina.com
2004-01-14	v1.3	完成全部翻译，并调整格式	LDAPChina.com
2004-01-28	v1.4	调整了章节格式	LDAPChina.com

Lightweight Directory Access Protocol (v3)
轻型目录访问协议(v3)

1、本备忘录的状态（Status of this Memo）

本文档定义了一个用于 Internet 通讯的 Internet 标准跟踪协议，为了发展的需要讨论和建议。对于这个协议的状况和地位请参照 Internet 官方协议标准（"Internet Official Protocol Standards"（STD 1））的当前版。这个备忘录的传播是不受限制的。

1.1、版权提示（Copyright Notice）

1.2、IESG 提示（IESG Note）

本文档描述将一种同时提供读和更新访问的目录访问协议。更新访问需要安全认证，但这个文档并不强制实现任何安全认证机制。

与 RFC2026 的 4.4.1 节相同，本规范正在被 IESG 批准期间，作为被提议的标准，可能并不限于本文档所述内容。原因如下：

a、 **鼓励在它们被发布前，实现和交互测试这些协议（带有或没有更新访问）；**

b、 **鼓励在只读的应用程序中配置和使用这些协议。**（例如，在某些应用程序中，目录的更新访问使用某些其它安全的机制而不是 LDAP，而使用 LDAPv3 被作为对目录的查询语言）；

c、 **避免阻碍别的 Internet 标准追踪协议的发展和发布。（这些协议需要** LDAPv3 的目录服务器的查询能力，而不是更新能力）

需要警告读者的是，直到强制的验证机制被标准化之前，根据本规范编写的客户端和服务器实现了更新功能的话，**它们的互操作性可能是不可靠的，或者仅提供在认证需要极度弱化的时候的互操作性。**

因此在具有强制认证的 LDAPv3 未成为一个 RFC 而被批准或发布之前，不鼓励实现者发布一个实现了更新功能的 LDAPv3 的客户端和服务器。

目　录

2、抽象（Abstract）

在本文档描述了这样一种协议，该协议可以访问支持 X.500 模型的目录产品，同时又避免 X.500 目录访问协议（DAP）过大的资源消耗。本协议特别针对提供对目录的读/写访问的管理程序和浏览程序，当这类程序与一个支持 X.500 协议的目录一同使用，本协议被认为是对 X.500 DAP 协议的补充。

在这个文档中的关键字"MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", 和"MAY" 与 RFC2119 中描述的一样。

这个 LDAP 版本的关键的几方面是：

1、 支持 LDAPv2（RFC1777）中的所有协议元素。协议直接在 TCP 上或其它传输层上运行。避开了在 X.500 DAP 上的会话层和表示层；

2、 绝大多数协议元素可以使用普通的字符串进行编码。（例如，分辨名）；

3、 可以返回其它 servers 的参照（Referrals）；

4、 SASL 机制与 LDAP 一同使用来提供相关的安全服务；

5、 通过使用 ISO 10646 字符集已使属性值和分辨名国际化；

6、 协议能够扩展以支持新的操作，对现有的操作进行扩展时可以使用 controls；

7、 模式（Schema）被公布在目录中供客户端使用。

3、模型（Models）

在 X.500 目录技术在 Internet 上的发展，已经转向减少在使用条目时的技术昂贵花销上。这个文档集继续努力定义目录的替换协议——LDAPv2 协议规范进行更新。

3.1、协议模式（Protocol Model）

被本协议所采用的总体模型是某个客户端（client）通过服务器（server）执行协议操作。在此模型中，客户端传送一个协议请求来描述将被服务器执行的操作。服务器负责在目录中执行必要的操作。在完成了操作后，服务器返回一个包含结果或错误信息的响应至客户端。

为了保持目录的使用价值与易用性相一致的目的，这个协议的一个目标是减少客户端的复杂性，以使具有目录访问能力的应用程序得以广泛发布。

注意：虽然服务器需要返回应答（只要在协议中规定了应答），但不需要同步，无论是客户端或服务器。对于多个操作的请求和应答，客户端和服务器端可以以任何顺序进行交换，只要为每个客户端的请求都最终传递了必要的应答即可。

在 LDAPv1 和 v2，没有协议规定服务器返回引用（Referral）到客户端。无论如何，这个版本的协议为了提高性能和分布性，允许返回指向别的服务器的引用。这允许服务器承受联系其他服务器处理操作的负担。

注意本文档所定义的协议的核心操作能够被映射为一个 X.500 目录抽象服务的严格子集，所以它能够被 DAP 清楚的提供。然而，在 LDAP 操作和 DAP 操作之间并不是一对一的映射关系：当一个为 X.500 目录提供网关服务的 LDAP 服务器在处理一个 LDAP 请求时，可能需要处理多个 DAP 请求。

3.2、数据模型（Data Model）

本节对 LDAP 中使用到的 X.500 的数据模型进行一个简要介绍。

LDAP 协议假设由一个或多个联合起来的服务器提供对 DIT（目录信息树：Directory Information Tree）的访问。树由条目（entry）组成。条目有名字：条目中的一个或多个属性值形成了相对分辨名（RDN），RDN 在它的兄弟节点（SIBLING）中必须是唯一的。从该节点 RDN 到树根的直属子节点的 RDN 的顺序连接构成了该节点的 DN，它在整个树中是唯一的。一个 DN 的例子：

CN=Steve Kille,O=Isode Limited,C=GB

某些服务器拥有条目的缓存或镜像拷贝，这些服务器能够被用来响应查询或比较请求，但假如请求的是修改操作，这些服务器将返回引用（referral），或者联系其他的服务器，让其他服务器处理。

条目的最大集合被称为命名上下文（naming context）。它从被一个特定服务器掌握的条目开始，包括所有它的下属节点和他们的下属节点，直到被不同服务器掌握的最终节点。DIT（目录信息树）的根是一个 DSA 特定条目（DSE：DSA-specific Entry），并且 DIT 的根并不属于命名上下文的一部分：每个服务器在根 DSE 中都有不同的属性值。（DSA 是 X.500 中为目录服务器定义的术语）。

3.2.1、条目的属性（Attributes of Entries）

条目由一个属性集组成。一个属性是一个属性类型加上一个或多个相关的值。属性类型被标识为一个简短的描述名和 OID（对象标识符：object identifier）。属性类型决定了该属性

是否能够有多个值，以及属性值所必需遵从的语法，以及在该值上所能执行的匹配方式和其他的功能。

一个属性的例子是“MAIL”。这个属性可以有多个值，它们必须是 IA5（ASCII）字符串，他们是大小写不敏感的。

模式（schema）是一个集合，该集合包括下列内容：1、属性类型定义；2、对象类定义；3、其它信息，这些信息可以帮助服务器决定如何将一个过滤器或属性值声明（attribute value assertion）与一个条目的属性进行匹配，以及是否允许增加和修改操作。用于 LDAP 的 schema 定义将在参考文档[5]和[6]中给出。附加的模式元素可能在其它文档中定义。

每个条目**必须**（MUST）有一个 objectClass 属性。objectClass 属性定义了一个条目的对象类，它与系统模式（schema）和用户模式一起决定了允许在一个条目中出现的属性。objectClass 属性的值可以被客户端修改，但它不能被删除。服务器可以不允许修改该属性以防止该条目的基本结构类被改变。（例如，不能将一个人改为一个国家）。当创建一个条目或在一个条中增加一个 objectClass 属性值时，该对象类的所有都被隐式地加入到对象的该属性中，客户端必须为这些新的父类的所有强制属性提供相应的值。

某些属性，被称为操作属性（operational attributes），服务器用使用这些属性来管理目录系统本身。他们在检索结果中不被返回，除非使用这些属性的名字显示地提出请求。某些非操作属性（如 mail），服务器将强迫进行模式（schema）和语法约束，但服务器一般不会使用它们的值。

除下列情况之外，服务器**绝不能**（MUST NOT）允许客户端向一个条目增加属性。

1、 对象类定义和控制该条目的模式（schema）允许的属性；

2、 服务器可以识别该属性是操作属性，并且该属性将用于管理目的。

注意有一个特定对象类’extensibleObjet’，该对象类允许将所有用户属性加入到一个条目中。

条目**可能**（MAY）包含下列操作属性。这些属性被服务器自动维护，不能被客户端修改：

-creatorsName：向目录添加该条目的用户的 DN

-createTimestamp：该条目被添加目录的时间

-modifiersName：最后一次修改该条目的用户的 DN

-modifyTimestamp：该条目被最后一次修改的时间

-subschemaSubentry：为该条目控制模式（schema）的 subschema 条目（或称 subentry）

的 DN。

3.2.2、子模式条目（Subschema Entries）和子条目（Subentries）

子模式条目（subschema entry）用于管理目录模式（schema）的相关信息，尤其是目录服务器支持的对象类和属性类型的信息。单个子模式条目包含目录树中特定部分由条目所使用的所有模式（schema）的定义。

符合 X.500 模型的目录服务器**应该**（SHOULD）使用 X.500 的子模式（subschema）机制实现子模式，所以这些子模式并不是普通的条目。LDAP 客户端**不应该**（SHOULD NOT）主观地认为服务器实现了 X.500 子模式的任何其他方面。一个拥有条目并且允许客户端修改这些条目的服务器**必须**（MUST）实现和提供对这些 subschema 条目的访问的方法，以便客户端可以发现哪些属性和对象类允许出现在一个条目中。强烈推荐其它的服务器（指非符合 X.500 模型的目录服务器）也实现此功能。

下面四个属性**必须**（MUST）在所有的子模式条目中都存在：

-cn：这个属性**必须**（MUST）被用来形成子模式的条目的 RDN

-objectClass：这个属性**必须**（MUST）最少有值”top”和“subschema”

-objectClasses：这个属性的每一个值指定一个服务器可识别的对象类

-attributeTypes：这个属性的每一个值指定一个服务器可识别的属性类型

这些被定义在参考文档[5]中。其它属性**可能**（MAY）出现在子模式目录中，以反映对附加功能的支持。

这些包括 matchingRules，matchingRuleUse，dITStructureRules，dITContentRules，nameForms 和 ldapSyntaxes。

服务器**应该**（SHOULD）在子模式条目中提供属性 createTimestamp 和 modifyTimestamp，以便允许客户端维护它们模式（schema）信息的缓存。

客户端**必须**（MUST）只能通过使用带有”(objectClass=subschema)”过滤条件的条目基本对象查询请求才能够检索一个子模式条目的属性（这将允许为 X.500 目录提供网关服务的 LDAPv3 服务器检索被请求的子条目（subentry）信息）。

3.3、与 X.500 的关系（Relationship to X.500）

本文档根据 X.500 定义 LDAP，并将其作为 X.500 的一种访问机制。一个 LDAP 服务器在提供服务时，**必须**（**MUST**）与 ITU 推荐的 X.500（1993）系列保持一致。然而，LDAP

服务器并不需要在提供这些服务时使用任何 X.500 协议。例如，只要 LDAP 使用了 X.500 数据和服务模型，LDAP 就能被映射到任何别的目录系统中，而不会在 LDAP 接口中造成冲突。

3.4、服务器特有的数据需要（Server-specific Data Requirements）

一个 LDAP 服务器**必须**（MUST）提供关于它自身的信息和每个服务器所特有的其它信息。这些信息通过位于根 DSE（DSA-specific entry）的一个属性组来表达，DSE 的名字用零长度的 LDAPDN 来表示。当客户端执行了一个带有"(objectClass=*)"过滤条件的对根的基本对象的查询时，这些属性是可以被检索到的，当然同时它们必须遵守访问控制约束。当客户端执行了一个从根开始的子树查询时，查询结果中绝不能（MUST NOT）包括根 DSE。

服务器可以允许客户端修改这些属性。

根 DSE 的下列属性被定义在参考文档[5]的 section 5 中。附加的属性可能在其文档中定义。

-namingContexts：服务器拥有的命名上下文。命名上下文在 X.501 的 section 17 中定义

-subschemaSubentry：已被该服务器识别的子模式条目（或称子条目）

-altServer：当这个服务器不能使用时的替代服务器

-supportedExtension：已被支持的扩展操作的列表

-supportedControl：已被支持的 control 的列表

-supportedSASLMechanisms：已被支持的 SASL 安全特性的列表

-supportedLDAPVersion：已被该服务器实现的 LDAP 版本的列表

如果服务器不掌管条目并且也不知道模式（schema）信息的位置，subschemaSubentry 属性将不出现在根 DSE 中。如果服务器在一个或多个 schema 规则下掌管目录中的条目的话，则在根 DSE 中可以存在任意数量的 subschemaSubentry 属性的值。

4、协议元素（Elements of Protocol）

LDAP 协议使用 Abstract Syntax Notation 1（ASN.1）描述，并使用 ASN.1 的 BER（Basic Encoding Rules）编码方式的一个子集进行传输（transfer）。为了支持本协议在将来的特性扩展，客户端和服务器**必须**（MUST）忽略含有不可识别标记符的序列（SEQUENCE）编码的

元素。

注意对于 LDAP 协议而言，若并非利用扩展机制而对协议造成的改变，都将产生一个不同的版本号，这一点与 x.500 不同。客户端显示地指出它支持的版本号，该版本号将作绑定（bind）请求一部分（将在 section 4.2 中描述）。假如客户端没有发送绑定请求，服务器**必须**（MUST）认定客户端支持 v3 协议。（因为 v2 需要客户端首先发送绑定请求）。

客户端可以通过读取 DSE 中的 supportedLDAPVersion 属性来判定服务器支持的协议版本。支持 LDAPv3 协议及后续协议的服务器**必须**（MUST）提供该属性。仅支持 v2 协议的服务器不必支持该属性。

4.1、公共元素（Common Elements）

本节描述 LDAPMessage 封套的 PDU（协议数据单元：Protocol Data Unit）格式，以及用于协议操作的数据类型定义。

4.1.1、消息封套（Message Envelope）

为了协议交换的目的，所有协议操作被封装在一个公共封套里，LDAPMessage 有如下定义：

```
LDAPMessage ::= SEQUENCE {
        messageID           MessageID,
        protocolOp          CHOICE {
                bindRequest         BindRequest,
                bindResponse        BindResponse,
                unbindRequest       UnbindRequest,
                searchRequest       SearchRequest,
                searchResEntry      SearchResultEntry,
                searchResDone       SearchResultDone,
                searchResRef        SearchResultReference,
                modifyRequest       ModifyRequest,
                modifyResponse      ModifyResponse,
                addRequest          AddRequest,
                addResponse         AddResponse,
                delRequest          DelRequest,
                delResponse         DelResponse,
                modDNRequest        ModifyDNRequest,
                modDNResponse       ModifyDNResponse,
                compareRequest      CompareRequest,
                compareResponse CompareResponse,
```

```
                    abandonRequest    AbandonRequest,
                    extendedReq       ExtendedRequest,
                    extendedResp      ExtendedResponse },
            controls          [0] Controls OPTIONAL }
    MessageID ::= INTEGER (0 .. maxInt)

    maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --
```

LDAPMessage 的功能是提供一个公共域（common field）的封套，在所有的协议交换中都需要该公共域。目前，仅有的的共同域是消息标识（messageID）和控制（control）。

如果服务器从客户端收到一个 PDU（协议数据单元），但该 PDU 的 LDAPMessage 序列（SEQUENCE）的标签不能被识别，那么 messageID 不能被解析，协议操作（protocolOp）的标签不会作为一个请求被识别。或者如果编码的结构或数据域的长度被发现不正确的话，服务器**必须**（MUST）返回带有 protocolError 结果码（resultCode）的 disconnection 的通知（将在 4.1.1 中描述），并且立即关闭连接。对于其它服务器不能正确解析来自客户端的请求的情况，服务器也**必须**（MUST）返回一个带有 protocolError 的结果码（resultCode）的适当响应。

如果客户端收到了一个来自服务器的无法解析的 PDU（协议数据单元），客户端可以放弃该 PDU，或直接关闭连接。

4.1.1.1、Message ID

所有封装了响应的 LDAPMessage 封套都包含相关请求的 LDAPMessage 的 messageID 值。

某个请求的 messageID 值**必须**（MUST）不同于所属 LDAP 会话中的任何其它请求的 messageID 值。

在客户端得到一个请求的响应之前，绝不能（MUST NOT）在该连接中再发送另一个与先前请求具有相同 messageID 的请求，否则结果将不可预测。典型的解决方法是，客户端为每一个请求的 messageID 增长一个计数。

客户端若发送了一个放弃请求（abandonRequest），则客户端在接收到该放弃请求之后发送的另一个请求的响应之前，绝不（MUST NOT）要重用该放弃请求（abandonRequest）或该放弃请求所要放弃的操作的 messageID，因为 abandonRequest 请求没有应答。

4.1.2、string Types

LDAPString 是一个符号约定，该约定指出虽然 LDAPString 类型的字符串以 OCTET STRING 进行编码，但 LDAPString 使用 ISO 10646 字符集，使用 UTF-8 算法进行编码。注意，UTF-8 规则的字符串（与 ASCII 码相同，从 0x0000 到 0x007F）在单字节内与 ASCII 字符相同。其它字节值被用来构成变长编码的任意字符。

LDAPString ::= OCTET STRING

LDAPOID 这一符号约定指出这一字符串所允许的值是对象标识符（OBJECT IDENTIFIER）的一种以 UTF-8 编码的点-数（dotted-decimal）形式字符串。

LDAPOID ::= OCTET STRING

例如：

1.3.6.1.4.1.1466.1.2.3

4.1.3、分辨名（Distinguished Name）和相对分辨名（Relative Distinguished Name）

在按照参考文档[4]规范进行编码之后，分辨名和相对分辨名分别被定义为 LDAPDN 和 RelativeLDAPDN，如：

<distinguished-name> ::= <name>
<relative-distinguished-name> ::= <name-component>

<name>和<name-component>在参考文档[4]中定义。

LDAPDN ::= LDAPString
RelativeLDAPDN ::= LDAPString

只有属性类型可以出现在 RDN 中，属性描述的选项部分**绝不**（MUST NOT）能用于指定 DN。

4.1.4、属性类型（Attribute Type）

一个 AttributeType 只接受在规范中与该 AttributeType 相关的文本字符串作为它的值。

AttributeType ::= LDAPString

每个属性类型（AttributeType）都有一个唯一分配给它的对象标识符（OBJECT

IDENTIFIER）。这个标识符是点数间隔形式书写的，如"2.5.4.10"。

规范也可以为一个属性类型分配一个或多个文字名字。这些名字**必须**（MUST）以字母开头，并且只能包含 ASCII 字符、数字和连字符。它们是大小写不敏感的。（这些字符与 ISO 10646 字符是相同的，都是单字节介于 0x00 和 0x7F 之间的）。

如果服务器拥有某个属性类型的文本名，则在查询结果中**必须**（MUST）返回该属性类型的文本名。点-数形式的对象标识符只能在属性类型没有文本名时使用。

属性类型的文本名不是唯一的，不同的两个规范（这些规范都不在 RFC 标准追踪中）可以选择同样的名称。

一个掌管或映射条目的服务器**应该**（SHOULD）在子模式条目（subschema entry）中用 attributeTypes 属性列出所有已支持的属性类型。支持属性扩展集的服务器至少应该包括”objectClass”属性的属性类型值。客户端**可以**（MAY）从子模式条目中检索 attributeTypes 的值，来获得对象标识符和与属性类型相关的其它信息。

参考文档[5]中描述了一些在本版本 LDAP 中使用的属性类型名。服务器可以实现附加的属性类型。

4.1.5、属性描述（Attribute Description）

一个属性描述（AttributeDescription）是属性类型定义的扩展集。它有同样的 ASN.1 定义，但允许另外的选项被指定。它们也是大小写不敏感的。

AttributeDescription ::= LDAPString

AttributeDescription 的值基于下列 BNF 规范：

```
<AttributeDescription> ::= <AttributeType> [ ";" <options> ]
<options>    ::= <option> | <option> ";" <options>
<option>     ::= <opt-char> <opt-char>*
<opt-char> ::=    ASCII-equivalent letters, numbers and hyphen
```

一个合法的 AttributeDescription 的例子：

cn

userCertificate;binary

"binary"选项在本文档中定义。附加的选项可以在 IETF 标准追踪和实验性 RFC 中定义。以"x-"开头的选项保留给个人使用。任何一个选项都能够与任何一个 AttributeType 结合，但服务器可以不支持所有的组合。

带有一个或多个选项的 AttributeDescription 被当作不带选项的属性类型的子类型

（subtype）处理。存在于 AttributeDescription 中的选项绝不会互相排斥。符合 LDAPv3 的实现**必须**（MUST）以升序生成<options>列表，并且服务器**必须**（MUST）认为带有同样 AttributeType 和选项的两个 AttributeDescription 是等价的。若 AttributeDescription 中带有未被服务器实现的选项，服务器将把它看作一个不可识别的 attributeType。

数据类型 AttributeDescriptionList 描述了一个拥有零个或多个属性类型的列表。(拥有零个元素的列表在查询请求中有重要的意义）

```
AttributeDescriptionList ::= SEQUENCE OF
        AttributeDescription
```

4.1.5.1、二进制选项（Binary Option）

如果”binary”选项存在于一个 AttributeDescription 中，它将忽略任何为该属性定义的（在参考文档[5]中）基于字符的编码表达形式。取而代之的是，属性将使用 BER 编码的二进制值形式进行传输。这个二进制值的语法是一个 ASN.1 数据类型定义，属性类型定义的语法部分将参照该 ASN.1 数据类型定义。

在协议中是否存在"binary"选项仅会影响属性值的传输；服务器以唯一一种形式存储所有属性。如果一个客户端请求服务器以二进制格式返回一个属性，但是服务器不能生成该格式，则服务器**必须**（MUST）把该属性类型看作不可识别的属性类型。类似地，如果客户端请求该属性时使用了不带 binary 选项的属性名称，那么客户端**绝不**（MUST NOT）能期望服务器以二进值格式返回该属性。

“binary”选项被确定为与如下所描述的属性一起使用，该属性的语法是一个复杂的 ASN.1 数据类型，并且客户端需要该 ASN.1 数据类型的值的结构。例如：证书（Certificate）和证书列表（CertificateList）这类的数据。

4.1.6、属性值（attribute Value）

AttributeValue 域把 AttributeValue 数据类型的字符串编码看作它的值，还是把包含二进制值的 OCTET STRING 看作它的值，将依赖于”binary”选项是否存在于该 AttributeValue 的 AttributeDescription 中。

AttributeValue ::= OCTET STRING

注意对这种编码的体积没有限制；这样协议值可以包括几兆字节的属性（如：照片）。

可以定义包含特殊字符或不可打印字符的属性。如果属性的语法不能被识别，实现程序**必须**（MUST）既不能简单显示也不能尝试对该值进行 ASN.1 编码。实现程序可以尝试寻找

该条目的子模式，从该子模式中检索 attributeTypes 的值。

客户端**绝不**（MUST NOT）能在一个请求中发送包含不符合属性语法定义的属性值。

4.1.7、属性值判定（Attribute Value Assertion）

本文档的 AttributeValueAssertion 类型定义与 X.500 目录标准对其的定义类似。它包含一个属性描述和一个适用于所判定类型的匹配规则判定值。

```
AttributeValueAssertion ::= SEQUENCE {
        attributeDesc     AttributeDescription,
        assertionValue    AssertionValue}


AssertionValue ::= OCTET STRING
```

若"binary"选项存在于 attributeDesc 中，则是告知服务器 assertionValue 是一个判定值的二进制编码。

对于在参考文档[5]中描述的所有以字符串为值的用户属性，判定值语法与值语法相同。客户端可以在比较请求和查询过滤器中把属性值当作判定值看待。

注意，对于其它属性或者非相等（non-equality）性匹配规则而言，判定语法可以与值语法不同。它们可以拥有仅包含值（value）部分（而不包含描述部分）的判定语法。例子请见 X.501 参考文档[6]的 section 20.2.1.8。

4.1.8、属性（Attribute）

一个属性由一个类型和该类型的一个或多个值组成(虽然属性在存储时最少必须有一个值，但由于访问控制的约束，当在协议中传输时，该集合可以为空。这些信息在 section 4.5.2，描述 PartialAttributeList 类型时描述)。

```
Attribute ::= SEQUENCE {
        type       AttributeDescription,
        vals       SET OF AttributeValue }
```

在属性值集合中每个属性值都是不同的（没有重复）。在属性值集合中没有定义属性值的顺序问题，它依赖于具体的实现，但**绝不**（MUST NOT）能成为依赖条件。

4.1.9、匹配规则标识符（Matching Rule Identifier）

匹配规则定义了服务器如何将查询过滤器中的 AssertionValue 与一个抽象数据值进行比较的方法。匹配规则定义了判定值的语法和判定时服务器端执行的流程。

在 LDAP 协议中，一个 X.501 匹配规则被标识为对象标识符的可打印格式。这种格式或者是参考文档[5]中定义的字符串中的一个，或者是一个点-数间隔的字符串，例如："caseIgnoreIA5Match"或"1.3.6.1.4.1.453.33.33"。

MatchingRuleId ::= LDAPString

为了支持在带有 extensibleMatch 的查询过滤器中使用匹配规则，服务器**必须**（MUST）在子模式条目（subschema entry）中列出服务器已实现的所有匹配规则，该列表放在子模式条目的 matchingRules 属性中。服务器也**应该**（SHOULD）在 matchingRuleUse 中列出可以使用该匹配规则的属性。更多信息在参考文档[5]的 section4.4 中描述。

4.1.10、结果信息（Result Message）

在本协议中 LDAPResult 中是服务器发送到客户端的一个结构。该结构可以返回一个请求是否执行成功。

对于不同种类的请求，为了说明某个协议操作请求的最终状态，服务器将返回包含 LDAPResult 类型域的响应。

```
LDAPResult ::= SEQUENCE {
        resultCode      ENUMERATED {
                        success                      (0),
                        operationsError              (1),
                        protocolError                (2),
                        timeLimitExceeded            (3),
                        sizeLimitExceeded            (4),
                        compareFalse                 (5),
                        compareTrue                  (6),

                        authMethodNotSupported       (7),
                        strongAuthRequired           (8),
                                -- 9 reserved --
                        referral                     (10),  -- new
                        adminLimitExceeded           (11),  -- new
                        unavailableCriticalExtension (12),  -- new
                        confidentialityRequired      (13),  -- new
```

```
                        saslBindInProgress      (14),    -- new
                        noSuchAttribute         (16),
                        undefinedAttributeType  (17),
                        inappropriateMatching   (18),
                        constraintViolation     (19),
                        attributeOrValueExists  (20),
                        invalidAttributeSyntax  (21),
                                -- 22-31 unused --
                        noSuchObject            (32),
                        aliasProblem            (33),
                        invalidDNSyntax         (34),
                        -- 35 reserved for undefined isLeaf --
                        aliasDereferencingProblem (36),
                                -- 37-47 unused --
                        inappropriateAuthentication (48),
                        invalidCredentials      (49),
                        insufficientAccessRights (50),
                        busy                    (51),
                        unavailable             (52),
                        unwillingToPerform      (53),
                        loopDetect              (54),
                                -- 55-63 unused --
                        namingViolation         (64),
                        objectClassViolation    (65),
                        notAllowedOnNonLeaf     (66),
                        notAllowedOnRDN         (67),
                        entryAlreadyExists      (68),
                        objectClassModsProhibited (69),
                                -- 70 reserved for CLDAP --
                        affectsMultipleDSAs     (71), -- new
                                -- 72-79 unused --
                        other                   (80) },
                        -- 81-90 reserved for APIs --
        matchedDN       LDAPDN,
        errorMessage    LDAPString,
        referral        [3] Referral OPTIONAL }
```

除了 success、compareFalse 和 compareTrue 以外，所有结果码均被认为操作未能全部完成。

绝大多数结果码都基于 X.511 错误数据类型定义的问题声明。结果码 16 到 21 指出属性问题（AttributeProblem），结果码 32、33、34 和 36 指出名称问题（NameProblem），结果码 48、49 和 50 指出安全问题（SecurityProblem），结果码 51 到 54 指出服务问题（ServiceProblem），

结果码 64 到 69、71 指出更新问题（UpdateProblem）。

如果客户端收到一个未出现在上面列表中的结果码，该结果码将被看作未知错误。

该结构的错误信息（errorMessage）域可以（服务器可选）被用来返回一个包含文本的、可读的诊断信息的字符串（该字符串中不应出现终止控制符和页面格式符）。该错误诊断信息并未标准化，**绝不**（MUST NOT）能依赖于该返回值进行任何实现。如果服务器不返回文本的诊断信息，LDAPResult 的错误信息（errorMessage）域**必须**（MUST）包含一个零长度的字符串。

针对结果码 noSuchObject，aliasProblem，invalidDNSyntax 和 aliasDereferencingProblem，matchedDN 域的值被设为在目录中所能匹配的最低条目（lowest entry）（对象或别名）的名称。如果尝试定位该条目时，没有别名被转换为 DN，那么 matchedDN 将是所提供的名称删减形式；如果别名被转换到 DN，那么 matchedDN 将是结果名称的删减形式（定义于 X.511 参考文档[8]）。对于所有其它的结果码，matchedDN 字段将被设为零长度的字符串。

4.1.11、引用（Referral）

引用错误声明与之联系的服务器不拥有请求中的目标条目。若 LDAPResult.resultCode 的值是 referral，则 referral 域将存在于 LDAPResult 中。但在所有其它结果码中不会出现 referral 域。引用错误可以包含一个对其它服务器（或服务器集合）的引用，这些服务器可能通过 LDAP 或其它协议进行访问。引用可以出现在任何其它操作请求的响应中（除了 unbind 和 abandon，它们没有响应）。Referral 中**必须**（MUST）至少存在一个 URL。

如果有一个单级（singleLevel）或一个整个子树（wholeSubtree）的查询，它的查询范围跨越多个命名上下文（naming contexts），并且需要联系多个不同的服务器才能完成该操作的话，则不返回引用（referral）。而应该返回继续引用（continuation references）（在 section 4.5.3 中描述）。

```
Referral ::= SEQUENCE OF LDAPURL    -- one or more

LDAPURL ::= LDAPString -- limited to characters permitted in URLs
```

假如客户端希望继续该操作，它**必须**（MUST）根据返回的引用访问这些服务器。所有的 URLs **必须**（MUST）拥有能被用来继续操作的同等能力。（如何实现多服务器访问的机制，不属于本文档讨论的范围）。

实现了 LDAP 协议的服务器，它的 URLs 的格式的书写根据参考文档[9]。如果一个别名被转换到了新的目标对象的名称，URL **必须**（MUST）存在<dn>部分，该<dn>包含新目标对象的名称。如果<dn>部分存在，客户端**必须**（MUST）在它接下来的继续处理操作中使

用这个名字；如果<dn>部分不存在，客户端将使用与原先请求中相同的名字来进行继续处理操作。某些服务器（例如：参与分布式索引的服务器）可能为一个查询操作在引用（referral）中提供不同的过滤器。如果在一个 LDAPURL 中出现了该 URL 的过滤器部分，客户端**必须**（MUST）在接下来的继续查询请求中使用这个过滤器；如过滤器没有出现，客户端将使用与原先查询相同的过滤器进行继续查询请求。新请求的其它方面可能与产生该 referral 的请求相同或不同。

注意出现在 DN 或查询过滤器（search filter）中的 UTF-8 字符，对于 URLs 来说可能是不合法的（如：空格），所以必须使用%（在 RFC1738 参考文档[7]中定义）字符转意。

其它类型的 URLs 可以被返回，只要操作能够使用那种协议执行。

4.1.12、控制（Control）

控制是一种指定扩展的信息的方法。随某个请求一起发送的控制，仅作用于该请求，并且不被保存。

```
Controls ::= SEQUENCE OF Control

Control ::= SEQUENCE {
        controlType             LDAPOID,
        criticality             BOOLEAN DEFAULT FALSE,
        controlValue            OCTET STRING OPTIONAL }
```

controlType 域必须是一个可以唯一标识该控制的对象标识符的 UTF-8 编码的点-数表达形式。这阻止了控制名之间的冲突。

Criticality 域的值为 TRUE 或 FALSE。

如果服务器可以识别某个控制类型，并且该控制适用于某操作，服务器在执行该操作时可以使用该控制。

如果服务器无法识别该控制类型，并且 criticality 域的值为 TRUE，则服务器**绝不**（MUST NOT）能执行该操作，并且**必须**（MUST）返回值为 unsupportedCriticalExtension 的结果码（resultCode）。

如果该控制不适用于该操作，并且 criticality 域的值为 TRUE，服务器**绝不**（MUST NOT）能执行该操作，并且**必须**（MUST）返回值为 unsupportedCriticalExtension 的结果码（resultCode）。

如果该控制不可被识别或不适用，但 criticality 域的值为 FALSE，服务器**必须**（MUST）忽略该控制。

controlValue 包含了与该控制相关的所有信息，并且该控制定义它的格式。服务器**必须**（MUST）准备处理 controlValue 八进制字符串中的特殊情况的内容，包括零字节。除非一个控制的类型值不存在，否则 controlValue 不能不存在。

本文档不定义任何控制。控制可能在其它文档中定义。一个控制的定义组成如下：

- 分配给控制的对象标识符

- 定义控制是否总是关键的，或者总是不关键的，或者在客户端选项中是关键的

- 控制的 controlValue 内容的格式

服务器在根 DSE 的 supportedControl 属性中列出可识别的控制。

4.2、绑定操作（Bind Operation）

绑定操作的功能是允许验证信息在客户端和服务器之间交换。

绑定请求定义如下：

```
BindRequest ::= [APPLICATION 0] SEQUENCE {
        version                 INTEGER (1 .. 127),
        name                    LDAPDN,
        authentication          AuthenticationChoice }

AuthenticationChoice ::= CHOICE {
        simple                  [0] OCTET STRING,
                                -- 1 and 2 reserved
        sasl                    [3] SaslCredentials }

SaslCredentials ::= SEQUENCE {
        mechanism               LDAPString,
        credentials             OCTET STRING OPTIONAL }
```

绑定请求的参数：

- version：版本号声明在协议会话中使用的协议版本。这个文档描述了 LDAPv3。注意这里没有版本协商，客户端将把该参数设为它希望的版本。如果客户端请求 v2，并且服务器支持 v2，则服务器不返回任何 v3 中定义的协议域（protocol field）。（注意不是所有的 LDAP 服务器都支持 v2，因此这些服务器不能产生与 v2 相关的语法属性）

- name：客户端希望使用其进行绑定的目录对象名。如果认证已经在一个更底的层次上执行，或者正在使用一种在信认中包含 LDAPDN 的 SASL 认证的机制时，name 域为了匿名绑定的目的可以接收空值（零长度字符串）。

- authentication：用于认证 name 的信息，如果需要的话，在绑定请求中提供。

一旦收到了 bind 请求，在必要的情况下，协议服务器将认证该请求客户端。服务器将返回一个绑定响应（bind response）给客户端，该响应声明了认证的状态。

当执行操作时，认证操作使用该认证信息。认证可能受到来自 LDAP 绑定请求外部的因素的影响，例如底层的安全服务。

4.2.1、绑定请求的顺序（Sequencing of the Bind Request）

对于某些 SASL 认证机制，客户端必须调用多次 BindRequest。在任何阶段如果客户端希望取消绑定处理，它**可以**（MAY）解除绑定（unbind）并且关闭连接。客户端在一个多阶段绑定处理中的某两个绑定请求之间，**绝不**（MUST NOT）能执行其它操作。

下列方法允许客户端取消 SASL 绑定：1、发送一个 SaslCredentials 的 mechanism 域中带有不同值的 BindRequest；2、发送一个 AuthenticationChoice 值不是 sasl 的 BindRequest。

如果客户端发送了一个 mechanism 域（mechanism 域是 sasl 域所属类型中的一个域）为空字符串的请求，服务器**必须**（MUST）返回一个 resultCode 值为 authMethodNotSupported 的 BindResponse。如果客户端希望重新尝试同样的 SASL 机制，这将允许客户端取消一个绑定协商。

与 LDAPv2 不同，客户端不必在连接中第一个 PDU（协议数据单元）中发送绑定请求。客户端可以请求任何操作，服务器**必须**（MUST）将把这些操作看作未被认证。如果服务器需要客户端在浏览或修改目录之前前进行绑定，服务器**可以**（MAY）使用带有 operationsError 结果的响应来拒绝一个非 bind、unbind 或扩展操作的请求。

如果客户端在发送一个请求之前没有进行绑定，并且它接到了 operationsError 信息，它可以在此时发送一个绑定请求。如果这样做也失败了或者客户端选择不在当前连接上进行绑定，那么客户端将关闭该连接，然后重新打开连接，并首先发送一个带绑定请求的 PDU（协议数据单元）以便重新开始。这将有助于实现与支持其它 LDAP 版本的服务器之间的交互。

客户端**可以**（MAY）在一个连接上发送多个绑定请求来改变信认。一个后续的绑定处理操作将导致服务器放弃在该连接上所有的操作。（这简化了服务器的实现）。先前绑定中的认证随后将被忽略，如果后续的绑定失败，连接将被视作匿名。如果一个 SASL 传输加密（transfer encryption）和完整性机制（integrity mechanism）已被协商，并且该机制不支持从一个身份到另一个身份的认证的改变，那么客户端**必须**（MUST）负责建立一个新的连接。

4.2.2、认证和其它安全服务（Authentication and Other Security Services）

简单认证（simple authentication）选项提供最小的认证功能，它的 authentication 域仅由明文的 password 组成。注意，如果底层没有执行认证和加密的话，不建议在开放网络上使用明文的 password。见"Security Considerations"一节，获得更多关于安全方面的信息。

如果没有执行认证，则**必须**（MUST）选择简单认证选项，并且 password 长度**必须**（MUST）为零。（LDAPv2 的客户端经常这样做）一般情况下 DN 的长度也是零。

sasl 选项的值可以是任何可与 SASL 一同使用的机制。mechanism 域包含机制的名称。credentials 字段包含用于认证的任意数据，这些数据被用 OCTET STRING 封装。注意与使用 SASL 某些 Internet 应用协议不同，LDAP 并不是基于文本的，这样一来，就不会在认证上执行 base64 的转换。

如果某个基于 SASL 的完整性和保密性服务被允许使用，它将影响：1、位于最后一个带有 success 的结果码的 BindResponse 响应之后的由服务器发送的信息；2、位于最后一个带有 success 的结果码的 BindResponse 响应之后的由客户端接收的信息。

客户端可以通过使用 SASL 扩展机制，请求服务器使用一个来自底层的认证信息。

4.2.3、绑定响应（Bind Response）

绑定响应定义如下：

```
BindResponse ::= [APPLICATION 1] SEQUENCE {
        COMPONENTS OF LDAPResult,
        serverSaslCreds       [7] OCTET STRING OPTIONAL }
```

BindResponse 的组成非常简单，它只包括声明认证请求的处理状态的信息。如果绑定成功了，resultCode 的值为 success，否则将是：

- operationsError：服务器碰到一个内部错误

- protocolError：不可识别的版本号或不正确的 PDU（协议数据单元）结构

- authMethodNotSupported：不可识别的 SASL 机制名

- strongAuthRequired：服务器需要执行 SASL 机制的认证

- referral：这个服务器不能接收这个绑定请求，并且服务器应当尝试另一个服务器

- saslBindInProgress：服务器需要客户端发送一个新的绑定请求，该绑定请求带有同样

的 sasl 机制，以继续该认证处理。

- inappropriateAuthentication：服务器需要试图匿名绑定或未提供信认的客户端提供某种信认。

- invalidCredentials：password 错误或不能处理 SASL 信认（credentials）

- unavailable：服务器正在关闭

如果服务器不支持客户端所请求的协议版本，它**必须**（MUST）将 resultCode 的值设置为 protocolError。

如果客户端收到了 resultCode 值为 protocolError 的 BindResponse 响应，它**必须**（MUST）关闭连接，并认为服务器将不希望接收进一步操作。（这是为了与早期的 LDAP 版本地取得兼容，早期版本中绑定总是第一个操作，没有协商过程）。

ServerSaslCreds 被用来作为 SASL 定义的（SASL-defined）绑定机制的一部分，ServerSaslCreds 允许客户端验证与之通讯的服务器的身份，或者允许客户端执行"challenge-response"认证。如果客户端选择使用使用 password 进行绑定，或者使用不需要服务器向客户端返回信息的 SASL 机制，那么该域不被包括在结果中。

4.3、解除绑定操作（Unbind Operation）

Unbind 操作的功能是终止一个协议会话。解除绑定操作定义如下：

UnbindRequest ::= [APPLICATION 2] NULL

解除绑定操作被定义为没有响应。一旦 UnbindRequest 被发送，协议客户端可以认为协议会话已经终止。一旦接收到 UnbindRequest，协议服务器可以认为发送该请求的客户端已经终止了会话，该会话内所有已发出的请求可以被放弃，并且可以关闭连接。

4.4、主动提示（Unsolicited Notification）

一个主动提示是服务器主动发送给某客户端的 LDAPMessage，主动提示并不是对服务器所接收到的 LDAPMessage 的响应：它被用于提示服务器内的或客户端和服务器之间的连接内的特殊情况。该提示是属于建议性的，服务器将不期望接收到来自客户端的响应。

主动请求被结构化为 LDAPMessage，但 messageID 是 0，并且 protocolOp 是 extendedResp（扩展响应）形式。它拥有 ExtendedResponse（扩展响应）的 responseName 域。主动请求的 LDAPOID 值**必须**（MUST）是唯一的，并且不能用于其它情况。

本文档定义了一个主动提示。见 4.1.1 节。

4.1.1、关闭连接提示（Notice of Disconnection）

该主动提示用于下面情况：服务器通知客户端，由于一个错误，客户器将关闭连接。

注意这个提示不是客户端发送的解除绑定（unbind）请求的应答：解除绑定时，服务器**必须**（MUST）按照 section 4.3 的步骤执行。而该提示的意图是帮助客户端辨别一个错误和一个短暂的网络失败之间的区别。当由于网络失败连接关闭时，客户端**绝不**（MUST NOT）能主观判定修改目录的请求成功了还是失败了。

该提示的响应名（responseName）是 1.3.6.1.4.1.1466.20036，不存在 response 域，并且 resultCode 指出关闭连接的原因。

下列 resultCode 值将被用于这个提示：

- protocolError：服务器收到了来自客户端的数据，其 LDAPMessage 结构无法解析

- strongAuthRequired：服务器检测到保护客户端和服务器之间通讯的底层安全服务意外失败或者已存在安全危及。

- unavailable：服务器将停止接受新的连接并且在当前所有连接上停止接受新操作，同时在一段较长时间内服务器不可用。客户端可以利用其它的服务器。

发出该提示后，服务器**必须**（MUST）关闭连接。接收到这个提示后，客户端**绝不**（MUST NOT）能再继续发送任何的请求，并立刻关闭连接。

4.5、查询操作（Search Operation）

查询操作允许客户端请求服务器执行一个指定的查询。查询操作可被用于：1、从单个条目读取属性；2、从指定条目的直接下级读取属性；3、从条目的整个子树中读取属性。

4.5.1、查询请求（Search Request）

查询请求定义如下：

```
SearchRequest ::= [APPLICATION 3] SEQUENCE {
        baseObject      LDAPDN,
        scope           ENUMERATED {
                baseObject              (0),
                singleLevel             (1),
                wholeSubtree            (2) },
        derefAliases    ENUMERATED {
```

```
                    neverDerefAliases       (0),
                    derefInSearching        (1),
                    derefFindingBaseObj     (2),
                    derefAlways             (3) },
            sizeLimit           INTEGER (0 .. maxInt),
            timeLimit           INTEGER (0 .. maxInt),
            typesOnly           BOOLEAN,
            filter              Filter,
            attributes          AttributeDescriptionList }

    Filter ::= CHOICE {
            and                 [0] SET OF Filter,
            or                  [1] SET OF Filter,
            not                 [2] Filter,
            equalityMatch       [3] AttributeValueAssertion,
            substrings          [4] SubstringFilter,
            greaterOrEqual      [5] AttributeValueAssertion,
            lessOrEqual         [6] AttributeValueAssertion,
            present             [7] AttributeDescription,
            approxMatch         [8] AttributeValueAssertion,
            extensibleMatch [9] MatchingRuleAssertion }

    SubstringFilter ::= SEQUENCE {
            type                AttributeDescription,
            -- at least one must be present
            substrings          SEQUENCE OF CHOICE {
                    initial [0] LDAPString,
                    any         [1] LDAPString,
                    final       [2] LDAPString } }

    MatchingRuleAssertion ::= SEQUENCE {
            matchingRule        [1] MatchingRuleId OPTIONAL,
            type                [2] AttributeDescription OPTIONAL,
            matchValue          [3] AssertionValue,
            dnAttributes        [4] BOOLEAN DEFAULT FALSE }
```

查询请求的参数：

- baseObject：需要查询的基准对象条目的 LDAPDN。

- scope：查询执行范围的指示器。该域的可选值的语义与 X.511 查询操作所定义的 scope 域的可选值的语义相同。

- derefAliases：指定查询中别名对象（alias object）处理方法的指示器。这个字段可能的值是：

neverDerefAliases：在查询基准对象下级范围或定位基准对象（base object）时不将别名解除引用（dereference）（转化为 DN）；

derefInSearching：在查询基准对象的下级范围时将别名解除引用，但在定位基准对象时不将别名解除引用；

derefFindingBaseObj：在定位基准对象时将别名解除引用，但在查询基准对象的下级范围时不将别名解除引用；

derefAlways：在查询或定位基准对象时均将别名解除引用。

- sizelimit：限制将返回的查询结果的最大条目数。该域值为 0 时，表示客户端没有为该查询请求 sizelimit 限制。服务器可以强制一个返回条目的最大数值。

- timelimit：限制执行一个查询的最大时间值（以秒为单位）。该域值为 0 时，表示客户端没有为该查询请求 timelimit 限制。

- typesOnly：指出在查询结果中是包含属性类型和值，还是仅包含属性类型。该域值为 TRUE，将仅返回属性类型（无值）。该域值为 FALSE 时，将返回属性类型和属性值。

- filter：一个过滤器定义了查询为匹配一个指定条目所必须满足的条件。

“and”，“or”和“not”选项可以被用于组成复合过滤器。在“and”，“or”选项中，**必须**（MUST）存在至少一个过滤器元素。其它选项与在该查询范围内条目的某个属性值匹配。（实现者提示：'not'过滤器是隐式标记模块的标记选择的一个例子。在 BER 中，该标签将被视作是显示的）

服务器**必须**（MUST）按照 x.511 的 section 7.8.1 中所描述的三值逻辑来判定过滤器。总之，一个过滤器或者被评估为“TURE”，或者“FALSE”，或者“Undefined”。如果对于某个特定条目，过滤器被评估为 TRUE，则该条目的属性将作为查询结果的一部分被返回（也要受到应用程序访问控制的约束）。如果过滤器被评估为 FALSE 或 Undefined，那么查询将忽略该条目。

在一个带有“and”选项的过滤器中，如果所有过滤条件均评估为 TRUE，则该过滤器为 TRUE；如果至少一个过滤条件为 FALSE，则该过滤器为 FALSE；否则，如果至少一个过滤条件为 Undefined，则该过滤器为 Undefined。在一个带有“or”选项的过滤器中，如果所有的过滤条件均评估为 FALSE，则该过滤器为 FALSE；如果至少一个过滤条件为 TRUE，则该过滤器为 TRUE；否则，如果至少一个过滤条件为 Undefined，则该过滤器为 Undefined。在一个带有“not”选项的过滤器中，若过滤条件为 FALSE，则过滤器为 TRUE，若过滤条件为 TRUE，则过滤器为 FALSE，若过滤条件为 Undefined，则过滤器为 Undefined。

若特定属性描述中的属性或子类型存在于某个条目中，则 present 匹配评估为 TRUE，

否则为 FALSE（也包括在 present 评估中有不可被识别的属性描述）。

ExtensibleMatch 是新出现在这一把版本的 LDAP 的。若不填写 matchingRule，则**必须**（MUST）填写 type 域，并且该 type 执行相等判定运算（type 实际上是属性）。若不填写 type 域并且填写了 matchingRule 域，matchValue 将与条目中所有支持该 matchingRule 的属性进行比较，并且 matchingRule 决定了判定值的语法。（若它与该条目中至少一个属性匹配，过滤器返回值为 TRUE；若它与条目中的任何一个属性都不匹配，过滤器返回值为 FALSE；若 matchingRule 不能被识别或 assertionValue 无法被解析，过滤器返回值为 Undefined。）若 type 和 matchingRule 域都被填写了，则 matchingRule **必须**（MUST）是一个 type 允许使用的匹配规则，否则过滤器返回值为 Undefined。若 dnAttributes 的值为 TRUE，该匹配将应用到组成条目 DN 的所有属性上，同时，若 DN 中的至少一属性在该过滤器中被评估为 TRUE，则该匹配也被评估为 TRUE。（编辑者提示：填写 dnAttributes 域以便避免同一类的匹配规则存在多个版本。例如在字符匹配中，一个规则应用到条目，而另一个规则应用到条目和 DN 属性中）。

当服务器不能决定是否判定值（assertion value）与一个条目匹配时，该过滤器项将被评估为 Undefined。若服务器不能识别位于 equalityMatch, substrings, greaterOrEqual, lessOrEqual, approxMatch 或 extensibleMatch 过滤器中的一个属性描述，则在 extensibleMatch 中的一个匹配规则 id 也不能被服务器识别，判定值不能被服务器解析，或者过滤请求类型尚未被服务器实现，那么 filter 返回值为 Undefined。例如，一个服务器没有识别属性类型 shoeSize，一个（shoeSize=*）的过滤器将评估为 FALSE，并且（shoeSize=12），（shoeSize>=12）和（shoeSize<=12）将评估为 Undefined。

若属性描述或匹配规则 id 没有被识别，或者判定值不能被解析，服务器**绝不能**（MUST NOT）返回错误。更多的 filter 处理细节描述在 X.511 的 section 7.8 中定义。

- attributes：返回结果中，匹配查询过滤器的每个条目的的属性列表。可以使用两个特殊的值：一个没有属性的空列表，和属性描述字符串”*”。这两个值都表示需要返回所有的用户属性。（"*"允许客户端请求除特殊操作属性（operational attributes）之外的所有属性）。

属性在列表中**必须**（MUST）最多只被命名一次，并且在条目中至多被返回一次。若列表中存在不能识别的属性描述，服务器将忽略它们。

如果客户端希望所有属性都不返回，它可以指定一个仅包含属性 OID "1.1"的列表。这个 OID 可以任意选择，不要与任何正在使用的属性相对应。

客户端的实现者应该注意，即使所有的用户属性都已被请求，某些条目的属性仍可能不包含在查询结果中，这是由于访问控制或其它约束造成的。进一步，服务器将不返回操作属性（operational attributes），除非这些属性的名字存在于列表中，例如 objectClasses 或 attributeTypes。这是因为某些操作属性的值可能非常庞大。（LDAP 使用的操作属性列表在

参考文档[5]中给出。）

注意，客户端可以使用带有检查 objectClass 属性是否存在的过滤器的一级 LDAP 查询操作来模拟类似 X.500 的"list"操作，同时客户端也可以使用带有相同过滤器的基准对象 LDAP 查询操作来模拟类似 X.500 的"read"操作。一个提供了 X.500 网关的服务器可以不使用 read 和 list 操作，但若它使用了，则必须提供与 X.500 查询操作同样的语义。

4.5.2、查询结果（Search Result）

服务器基于查询请求完成的返回结果是一个查询响应（Search Responses），查询响应或者包含 SearchResultEntry，SearchResultReference，ExtendedResponse 或 SearchResultDone 中的一个数据类型。

```
SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
        objectName          LDAPDN,
        attributes          PartialAttributeList }

PartialAttributeList ::= SEQUENCE OF SEQUENCE {
        type       AttributeDescription,
        vals       SET OF AttributeValue }
```

实现者应当注意 PartialAttributeList 可以有 0 个元素（如果没有请求任何条目的属性，或者没有属性能被返回），并且 vals 也可能有 0 个元素 。（如果只请求了类型，或者所有的值都被排除在结果之外。）

```
SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL

-- 至少一个 LDAPURL 元素应该出现

SearchResultDone ::= [APPLICATION 5] LDAPResult
```

一旦收到一个查询请求，服务器将对 DIT（目录信息树）执行必要的查询。

如果 LDAP 会话在面向连接的传输（如：TCP）上操作，服务器将返回给客户端一系列独立的 LDAP 消息响应。它们可能是 0 个或多个包含 SearchResultEntry 的响应，每个对应一个查询到的条目。也可能是 0 个或多个包含 SearchResultReference 的响应，每个对应查询中该服务器未进行搜索的区域。SearchResultEntry 和 SearchResultReference PDU（协议数据单元）的顺序可以任意。当服务器已经返回了所有 SearchResultReference 响应和所有 SearchResultEntry 响应之后，服务器将返回包含 SearchResultDone 的一个响应，它包含一个成功的指示或所有发生的错误的细节。

随 SearchResultEntry 返回的每一个条目中包含在查询请求的 attributes 域中指定的所有

属性，如有必要不包含所有相关属性的值，返回的属性也受到访问控制和其它管理策略的影响。某些属性可以以二进制格式返回（使用 AttributeDescription 指定在响应中有二进制选项存在）。

某些属性可能被服务器构建，并出现在一个 SearchResultEntry 属性列表中，虽然它们不是条目所存储的属性。即使在访问控制中允许修改，客户端也**绝不**（MUST NOT）能主观地认为所有属性都可以修改。

ExtendedResponse 格式的 LDAPMessage 响应被保留，它用来返回与客户端请求的控制（control）相关的信息，这些内容可能在本文档的将来版本中定义。

4.5.3、在查询结果中的继续引用（Continuation References in the Search Result）

如果服务器能够定位到 baseObject 引用的条目，但是无法查询到 baseObject 及下级范围的所有条目，服务器可以返回一个或多个 SearchResultReference，每个包含一个可以继续该操作的其它服务器集合的引用。假如服务器没有定位 baseObject 并且没有查询到任何条目，那么它**绝不**（MUST NOT）能返回任何 SearchResultReference，。这种情况下，服务器应该返回一个包括 referral resultCode（引用结果码）的 SearchResultDone 作为响应。

当拥有子范围命名上下文的其它服务器的索引信息不存在，而不能提供本服务器时，SearchResultReference 响应将不受查询过滤器的影响，并且总是在范围之内时返回。

SearchResultReference 与 Referral 具有同样的数据类型。实现 LDAP 协议的服务器的 URL 格式按照参考文档[9]书写。<dn>部分**必须**（MUST）出现在 URL 中，指出新的目标对象名。客户端**必须**（MUST）在下一个请求中使用这个名称。某些服务器（例如：在一个分布式索引交换系统的一部分）可以在 SearchResultReference 的 URL 中提供一个不同的过滤器。如果 URL 的 filter 部分出现在 LDAP URL 中，客户端**必须**（MUST）在下一个请求中使用该过滤器来处理该查询；如果没有指定 URL 的 filter 部分，客户端将使用相同的过滤器。新查询请求的其它方面可以与刚才产生该继续引用（references）的请求相同或不同。

只要该操作能够使用那个协议来执行，其它类型的 URL 也可以被返回。

在 SearchResultReference 中未搜索的子树的名称不需要是基准对象的下属。

为了完成该查询，客户端**必须**（MUST）为每个返回的 SearchResultReference 发出一个新的查询操作。注意 section 4.11 将描述的放弃（abandon）操作仅应用于一个客户端和一个服务器连接上的特定操作，如果客户端对多个不同服务器发出了多个查询操作，它**必须**（MUST）单独取消每个操作。

4.5.3.1、例子（Example）

例如，假设已连接服务器（hosta）拥有条目"O=MNN,C=WW"和条目"CN=Manager,O=MNN,C=WW"。该服务器知道或者 LDAP 服务器（hostb）或（hostc）拥有条目"OU=People,O=MNN,C=WW"（一个是主服务器，另一个是映像服务器），并且 LDAP 服务器（hostd）拥有子树"OU=Roles,O=MNN,C=WW"。如果在"O=MNN,C=WW"之上的子树查询请求被发送到已连接服务器，它可能返回如下信息：

O=MNN,C=WW 的 SearchResultEntry

SearchResultEntry for CN=Manager,O=MNN,C=WW
SearchResultReference {
ldap://hostb/OU=People,O=MNN,C=WW
ldap://hostc/OU=People,O=MNN,C=WW
}
SearchResultReference {
ldap://hostd/OU=Roles,O=MNN,C=WW
}
SearchResultDone (success)

客户端实现者应该注意，紧随着 SearchResultReference 信息，可能会产生一个附加的 SearchResultReference。继续上面的例子，如果客户端连接至服务器（hostb），并请求查询子树"OU=People,O=MNN,C=WW"，服务器可能返回如下信息：

OU=People,O=MNN,C=WW 的 SearchResultEntry

SearchResultReference {
ldap://hoste/OU=Managers,OU=People,O=MNN,C=WW
}
SearchResultReference {
ldap://hostf/OU=Consultants,OU=People,O=MNN,C=WW
}
SearchResultDone (success)

如果已连接服务器并不拥有该查询的基准对象（base object），它将给客户端返回一个引用（referral）。例如，如果客户端向 hosta 发送一个"O=XYZ,C=US"子树查询请求，服务器可能仅返回一个包含一条引用（referral）的 SearchResultDone。

SearchResultDone (referral) {
ldap://hostg/
}

4.6、修改操作（Modify Operation）

查询操作允许客户端请求服务器执行一个指定的查询。修改操作允许客户端请求服务器执行一个条目的修改。修改请求定义如下：

```
ModifyRequest ::= [APPLICATION 6] SEQUENCE {
        object          LDAPDN,
        modification    SEQUENCE OF SEQUENCE {
                operation       ENUMERATED {
                                        add     (0),
                                        delete  (1),
                                        replace (2) },
                modification    AttributeTypeAndValues } }

AttributeTypeAndValues ::= SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
```

修改请求的参数：

- object：被修改的对象，该域的值包含被修改条目的 DN。服务器在决定被修改的对象时将不执行任何别名解除引用（将别名转换成 DN）。

- modification：在条目上执行的一系列修改的列表。整个条目修改列表**必须**（MUST）被视为一个原子操作，并按照所列出的顺序执行。然而某个修改（modification）可能会违反了目录模式（directory schema），则在整个修改列表全部执行完毕后，经过修改的条目**必须**（MUST）符合目录模式的要求。每个 modification 结构中的 operation 域的可接受如下相关语义：

add：在给定的属性上增加列出的值，如属性不存在，则创建该属性。

delete：从给定的属性上删除列出的值，如果没有列出要删除属性的值，则删除整个属性；或者如果列出了要删除属性的所有值，也删除整个属性。

replace：在给定的属性中，用列出的新值替换所有的原值，如果属性不存在，则创建该属性。如果某个属性存在，一个无值的 replace 将导致该属性被删除；如果该属性不存在，则该 replace 被忽略。

服务器基于修改请求完成的修改结果是一个修改响应（Modify Response）：

```
ModifyResponse ::= [APPLICATION 7] LDAPResult
```

一旦收到一个修改请求，服务器将对 DIT（目录信息树）执行必要的修改操作。服务器将返回给客户端一条单独的修改响应，该响应或者表示 DIT 的修改已成功完成，或者描述修改失败的原因。值得注意的是，由于在应用修改请求中的修改列表时有原子性要求，所以当客户端接收到任何类型的错误的修改响应时，DIT 应该没有执行任何修改；并且当客户端接收到成功执行完毕的修改响应时，所有修改请求应该已执行完毕。如果连接失败，修改是否发生将不可预知。

修改操作不能用于删除一个条目中的任何分辨值，这些值组成了该条目的 RDN（相对分辨名）。这种尝试将导致服务器返回错误 notAllowedOnRDN。专门用于重命名条目名称的修改 DN 操作将在 section 4.9 中描述。

如果某个相等判断匹配过滤器在某个属性类型中没有定义，客户端**绝不**（MUST NOT）可以使用 modification 的"delete"形式去尝试从条目中删除那个属性的单个值，而**必须**（MUST）使用"replace"形式代替。

值得注意的是，由于 LDAP 中作了简化，所以在 LDAP 的 ModifyRequest 中的 modification 并不是 DAP 的 ModifyEntry 操作中的 EntryModifications 的直接镜像（direct mapping），并且不同的 LDAP-DAP 网关可以使用不同方法来代表这种转换。如果这种方法成功的话，该操作对条目的最终影响**必须**（MUST）是相同的。

4.7、添加操作（Add Operation）

添加操作允许客户端请求向目录中增加一个条目。增加请求定义如下：

```
AddRequest ::= [APPLICATION 8] SEQUENCE {
        entry           LDAPDN,
        attributes      AttributeList }
AttributeList ::= SEQUENCE OF SEQUENCE {
        type    AttributeDescription,
        vals    SET OF AttributeValue }
```

添加请求的参数如下：

- entry：被添加条目的 DN。注意，在定位被添加的条目时，服务器不进行任何别名到 DN 的转换。

- attributes：组成被添加条目内容的属性列表。客户端**必须**（MUST）在该列表中包含分辨值（distinguished value）（这些值组成了条目 RDN），objectClass 属性和其它任何对象类列出的强制属性的值。客户端**绝不**（MUST NOT）能提供 createTimestamp 或 creatorsName 属性，因为这些属性由服务器自动生成。

为了 AddRequest 执行成功，**绝不**（MUST NOT）能存在与 AddRequest 中 entry 域同名

的条目。被添加条目的父条目**必须**（MUST）存在。例如，假设客户端尝试添加条目"CN=JS,O=Foo,C=US"，但条目"O=Foo,C=US"不存在，条目"C=US"存在，则服务器将返回 noSuchObject 错误，noSuchObject 中 matchedDN 域的值为"C=US"的。如果父条目存在，但并不在该服务器所拥有的命名上下文（naming context）中，那么该服务器**应该**（SHOULD）返回一个拥有该父条目的服务器的引用（referral）。

服务器**不应该**（SHOULD NOT）在目录中限制哪些位置可以添加条目，某些服务器**可能**（MAY）允许管理员限制哪些类（class）的条目可以被添加到目录中。

一旦收到一个添加请求，服务器将尝试执行添加请求。添加操作的结果将以添加响应(Add Response)的形式返回给客户端，定义如下：

AddResponse ::= [APPLICATION 9] LDAPResult

success 响应表明新的条目已存在于目录中了。

4.8、删除操作（Delete Operation）

删除操作允许客户端请求从目录中删除一个条目。删除请求定义如下：

DelRequest ::= [APPLICATION 10] LDAPDN

删除请求由被删除的条目的 DN 组成。注意，在解析将被删除的目标条目时，服务器不会将别名转换到 DN，并且在此操作中只有叶条目（leaf entry）（没有下级的条目）可以被删除。

基于接收到的删除请求，服务器将删除操作的结果以删除响应（Delete Response）的形式返回，删除响应定义如下：

DelResponse ::= [APPLICATION 11] LDAPResult

一旦接收到一个删除请求，服务器将尝试执行条目删除请求。删除操作的结果将以删除响应的形式返回给客户端。

4.9、修改 DN 操作（Modify DN Operation）

修改 DN 操作允许客户商修改目录中的条目名称的最左边（重要性最小）的部分，或者把一个条目的整个子树移至目录中的新位置。修改 DN 的请求定义如下：

```
ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry           LDAPDN,
        newrdn          RelativeLDAPDN,
```

```
        deleteoldrdn    BOOLEAN,
        newSuperior     [0] LDAPDN OPTIONAL }
```

修改 DN 的请求的参数定义如下：

- entry：将修改的条目的分辨名（DN）。该条目可以有也可以没有下级条目。

- newrdn：该 RDN 将组成条目新名称的最左边元素。

- deleteoldrdn：一个布尔参数，决定是否原 RDN 属性值仍作为条目的属性保留，或者从条目中删除。

- newSuperior：如果存在，拥有该 DN 的条目将变为要修改条目的直接上级。

基于接收到的修改 DN 请求，服务器将修改名称操作的结果以修改 DN 响应（Modify DN Response）的形式返回，定义如下：

```
ModifyDNResponse ::= [APPLICATION 13] LDAPResult
```

一旦接收到一个 ModifyDNRequest，服务器将尝试执行改变名称。改变名称的结果将以修改 DN 响应的形式返回给客户端。

例如，如果在"entry"参数中将条目命名为"cn=John Smith,c=US"，"newrdn"参数值为"cn=John Cougar Smith"，"newSuperior"参数空缺，则这个操作将尝试把条目的名称修改为"cn=John Cougar Smith,c=US"。如果已存在一个同名条目，则该操作将失败，并返回错误码 entryAlreadyExists 的失败信息。

若 deleteoldrdn 参数值为 TRUE，则组成原 RDN 的值从条目中被删除。若 deleteoldrdn 参数值为 FALSE，则组成原 RDN 的值将被保留，但只能作为该条目的非分辨属性值（non-distinguished attribute value）。如果 deleteoldrdn 的设置导致条目中的模式（schema）不一致，则服务器可以不执行该操作，并返回一个错误码。

值得注意的是，X.500 限制 ModifyDN 操作仅能对包含在一个单独服务器上的条目产生影响。如果 LDAP 服务器被镜像到 DAP，则这种限制将被应用，并且如果这种错误发生的话，应该返回 affectsMultipleDSAs 结果码（resultCode）。总之，客户端**绝不**（MUST NOT）能期待在多个服务器之间执行条目和子树的任意移动。

4.10、比较操作（Compare Operation）

比较操作允许客户端对目录中的一个条目进行判定比较。比较请求定义如下：

```
CompareRequest ::= [APPLICATION 14] SEQUENCE {
        entry           LDAPDN,
        ava             AttributeValueAssertion }
```

比较请求的参数：

- entry：要进行比较的条目的名称。

- ava：要与条目中的一个属性进行比较的判定值。

基于接收到的比较请求，服务器将比较操作的结果以比较响应（Compare Response）的形式返回，定义如下：

CompareResponse ::= [APPLICATION 15] LDAPResult

一旦接收到一个比较请求，服务器将尝试执行被请求的比较。比较的结果将以比较响应的形式返回给客户端。注意，比较的错误和比较结果在同一个结构中返回。

值得注意的是，某些目录系统可以建立访问控制，用于允许可以比较某个属性的值（如userPassword），但不可以读取该值。在一个查询结果中，该属性的类型可以被返回，但值为空。

4.11、放弃操作（Abandon Operation）

放弃操作的功能是允许客户端请求服务器终止一个操作。放弃请求定义如下：

AbandonRequest ::= [APPLICATION 16] MessageID

MessageID **必须**（MUST）是在同一连接中先前的请求的操作的 ID。

（放弃请求有自己的 Mssage id。该 ID 与要放弃的操作的 ID 不同。）

放弃操作没有定义响应。一旦发出了放弃请求，客户端可以认为在放弃请求的 Message ID 中指定的操作已经被放弃了。如果服务器正在传输某个查询的响应时，接收到了放弃该查询的请求，服务器**必须**（MUST）立即终止该传输条目，并且**绝不**（MUST NOT）能返回 SearchResponseDone。当然，服务器**必须**（MUST）保证仅传输被适当编码的 LDAPMessage PDU。

客户端**绝不**（MUST NOT）能对同一操作多次发送放弃请求，并且**必须**（MUST）时刻准备从它所放弃的请求中接收结果（在请求放弃之前，这些结果以由服务器发出了。）

服务器**必须**（MUST）丢弃不能识别的放弃请求的 Message ID，因为这样的操作无法被放弃，或者该操作已经被放弃了。

4.12、扩展操作（Extended Operation）

在 LDAPv3 版本中加入了扩展机制，可以允许为服务定义在本协议其它地方并不有效

的附加的操作，例如数字签名（digitally sign）操作和结果。

扩展操作允许客户端使用预定义的语法和语义发送请求和接收响应。这些可能在 RFC 中定义，也可能由特定的实现拥有。每个请求**必须**（MUST）有一个分配给它的唯一的对象标识符（OBJECT IDENTIFIER）。

```
ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
        requestName      [0] LDAPOID,
        requestValue     [1] OCTET STRING OPTIONAL }
```

requestName 是一个与请求对应的对象标识符的点-数（dotted-decimal）表示。requestValue 是某种由请求定义的格式的信息，用八进制字符串（OCTET STRING）封装。

服务对将使用包含 ExtendedResponse 的 LDAPMessage 来响应扩展请求。

```
ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
        COMPONENTS OF LDAPResult,
        responseName     [10] LDAPOID OPTIONAL,
        response         [11] OCTET STRING OPTIONAL }
```

如果服务器无法识别请求名，它**必须**（MUST）仅返回包含 protocolError 结果码的 response 域的 LDAPResult。

5、协议元素的编码和传输（Protocol Element Encodings and Transfer）

这里定义了一个底层服务。客户端和服务器**应该**（SHOULD）实现在 5.2.1 中描述的位于 TCP 之上的 LDAP 映射（mapping）。

5.1、基于 BER 传输服务上的映射（Mapping Onto BER-based Transport Services）

LDAP 的协议元素为了交换目的，使用 ASN.1 [3]的 BER（Basic Encoding Rules）[11]进行编码。然而，由于使用 BER 某些 BER 元素时会付出高额的开销，所以在 LDAP 协议元素的 BER 编码中加入了如下附加的限制：

（1） 使用长度编码的定形式（definite form）。

（2） 八进制字符串（OCTET STRING）值仅被编码成原形式（primitive form）。

（3） 如果某布尔（BOOLEAN）类型的值是 true，则编码**必须**（MUST）把它的八进

制内容设为十六进制"FF"。

（4） 如果某类型的值是它的缺省值，它**必须**（MUST）不填写。在本协议定义中只有某些 BOOLEAN 和 INTEGER 类型有缺省值。

除非有其它提示，否则上述限制并不应用于封装在八进制字符串（OCTET STRING）值里的 ASN.1 类型，例如属性值。

5.2、传输协议（Transfer Protocols）

本协议被设计成为运行于面向连接（connection-oriented）的、可靠的传输之上，数据流中每个字节中的全部 8 个字位都是非常重要的。

5.2.1、Transmission Control Protocol（TCP）

LDAPMessage 协议数据单元（PDU）被直接映射为 TCP 字节流（bytestream）。建议运行于 TCP 之上的服务器**可以**（MAY）在端口 389 上提供一个协议监听器。服务器也可以在其他端口上提供监听器。客户端**必须**（MUST）支持在任意有效的 TCP 端口上与服务器进行联系。

6、实现指南（Implementation Guidelines）

本文档描述了一个 Internet 协议。

6.1、服务器实现（Server Implementations）

服务器**必须**（MUST）有能力识别所有在参考文档[5]中定义的强制属性类型名（attribute type name），并实现所有语法。服务器也**可以**（MAY）识别附加的属性类型名。

6.2、客户端实现（Client Implementations）

请求引用（referrals）的客户端**必须**（MUST）保证它们不在服务器之间循环（loop）。它们绝不（MUST NOT）能对同一个服务器，使用相同的请求（目标条目名、范围和过滤器均相同）进行重复请求。某些客户端可以使用一个计数器，该计数器为在一个操作中每次发生的引用（referral）处理增加计数。这种客户端**必须**（MUST）能够处理一个在根（root）条目和叶（leaf）条目之间的命名上下文至少有十层的目录信息树（DIT）。

在没有与服务器事先约定的情况下，客户端**不应该**（SHOULD NOT）认为服务器支持任何在 section 6.1 参考之外的特殊模式（schema）。不同的模式中的不同属性类型可以有相同的名称。客户端能够检索由 subschemaSubentry 属性参照的子模式条目（subschema entry），subschemaSubentry 属性位于服务器的根 DSE 条目中或位于该服务器拥有的条目中。

7、安全考虑（Security Considerations）

当使用一个面向连接的传输，本协议版本为 LDAPv2 认证机制提供了手段（LDAPv2 使用明文密码的简单认证），也提供了 SASL 机制。SASL 允许为完整性和保密性服务进行协商。

如果客户端选择这样做的话，SASL 也允许服务器能够返回它的凭证（credential）给客户端。

在底层的传输服务不能保证机密的情况下，强烈不建议使用明文密码，并且明文密码的使用也可能导致密码泄漏给未授权的地方。

当使用 SASL 时，应该注意的是 BindRequest 的 name 域不能保证不被修改。如果客户端的分辨名（一个 LDAPDN）由信用协商（negotiation of the credentials）认可，它将优先于未被保护的 name 域中的任何值。

某个服务器实现了缓存属性，以及缓存通过 LDAP 得到的条目，那么该服务器**必须**（MUST）保证当信息被提供给多个客户端时，访问控制能够被维护，这是由于服务器可能存在访问控制策略，该策略可以防止查询结果中的条目或属性返回至特定的授权客户端以外的其它客户端中。例如，缓存仅向这样的客户端提供结果信息，即该客户端的请求引起结果信息被缓存。

8、感谢（Acknowledgements）

本文档是对 RFC 1777 的更新，由 Wengyik Yeong，Tim Howes，和 Steve Kille 编写。在本文档中包含的设计思想基于 ASID 和其它 IETF 工作组的讨论。在这些工作组里每个人的贡献都应得到感谢。

9、参考书目（Bibliography）

[1] ITU-T Rec. X.500, "The Directory: Overview of Concepts, Models and Service", 1993。

[2] Yeong, W., Howes, T., and S. Kille, "Lightweight Directory Access Protocol", RFC 1777,

March 1995。

[3] ITU-T Rec. X.680, "Abstract Syntax Notation One (ASN.1) - Specification of Basic Notation", 1994。

[4] Kille, S., Wahl, M., and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", RFC 2253, December 1997。

[5] Wahl, M., Coulbeck, A., Howes, T., and S. Kille, "Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions", RFC 2252, December 1997。

[6] ITU-T Rec. X.501, "The Directory: Models", 1993。

[7] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, December 1994。

[8] ITU-T Rec. X.511, "The Directory: Abstract Service Definition", 1993。

[9] Howes, T., and M. Smith, "The LDAP URL Format", RFC 2255, December 1997。

[10] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997。

[11] ITU-T Rec. X.690, "Specification of ASN.1 encoding rules: Basic, Canonical, and Distinguished Encoding Rules", 1994。

[12] Meyers, J., "Simple Authentication and Security Layer", RFC 2222, October 1997。

[13] Universal Multiple-Octet Coded Character Set (UCS) - Architecture and Basic Multilingual Plane, ISO/IEC 10646-1 : 1993。

[14] Yergeau, F., "UTF-8, a transformation format of Unicode and ISO 10646", RFC 2044, October 1996。

10、作者地址（Authors' Addresses）

Mark Wahl
Critical Angle Inc.
4815 W Braker Lane #502-385
Austin, TX 78759
USA
Phone: +1 512 372-3160
EMail: M.Wahl@critical-angle.com

Tim Howes
Netscape Communications Corp.
501 E. Middlefield Rd., MS MV068
Mountain View, CA 94043
USA
Phone: +1 650 937-3419
EMail: howes@netscape.com

Steve Kille
Isode Limited
The Dome, The Square
Richmond
TW9 1DT
UK
Phone: +44-181-332-9091
EMail: S.Kille@isode.com

附录 A、完整的 ASN.1 定义（Complete ASN.1 Definition）

Lightweight-Directory-Access-Protocol-V3 定义

IMPLICIT TAGS ::=

开始

```
LDAPMessage ::= SEQUENCE {
        messageID           MessageID,
        protocolOp          CHOICE {
                bindRequest         BindRequest,
                bindResponse        BindResponse,
                unbindRequest       UnbindRequest,
                searchRequest       SearchRequest,
                searchResEntry      SearchResultEntry,
                searchResDone       SearchResultDone,
                searchResRef        SearchResultReference,
                modifyRequest       ModifyRequest,
                modifyResponse      ModifyResponse,
                addRequest          AddRequest,
                addResponse         AddResponse,
                delRequest          DelRequest,
                delResponse         DelResponse,
                modDNRequest        ModifyDNRequest,
```

```
                    modDNResponse      ModifyDNResponse,
                    compareRequest     CompareRequest,
                    compareResponse CompareResponse,
                    abandonRequest     AbandonRequest,
                    extendedReq        ExtendedRequest,
                    extendedResp       ExtendedResponse },
            controls          [0] Controls OPTIONAL }

MessageID ::= INTEGER (0 .. maxInt)

maxInt INTEGER ::= 2147483647 -- (2^^31 - 1) --

LDAPString ::= OCTET STRING

LDAPOID ::= OCTET STRING

LDAPDN ::= LDAPString

RelativeLDAPDN ::= LDAPString

AttributeType ::= LDAPString

AttributeDescription ::= LDAPString

AttributeDescriptionList ::= SEQUENCE OF
        AttributeDescription

AttributeValue ::= OCTET STRING

AttributeValueAssertion ::= SEQUENCE {
        attributeDesc     AttributeDescription,
        assertionValue    AssertionValue }

AssertionValue ::= OCTET STRING

Attribute ::= SEQUENCE {
        type       AttributeDescription,
        vals       SET OF AttributeValue }

MatchingRuleId ::= LDAPString

LDAPResult ::= SEQUENCE {
        resultCode         ENUMERATED {
```

```
        success                      (0),
        operationsError              (1),
        protocolError                (2),
        timeLimitExceeded            (3),
        sizeLimitExceeded            (4),
        compareFalse                 (5),
        compareTrue                  (6),
        authMethodNotSupported       (7),
        strongAuthRequired           (8),
                -- 9 reserved --
        referral                     (10),  -- new
        adminLimitExceeded           (11),  -- new
        unavailableCriticalExtension (12),  -- new
        confidentialityRequired      (13),  -- new
        saslBindInProgress           (14),  -- new
        noSuchAttribute              (16),
        undefinedAttributeType       (17),
        inappropriateMatching        (18),
        constraintViolation          (19),
        attributeOrValueExists       (20),
        invalidAttributeSyntax       (21),
                -- 22-31 unused --
        noSuchObject                 (32),
        aliasProblem                 (33),
        invalidDNSyntax              (34),
        -- 35 reserved for undefined isLeaf --
        aliasDereferencingProblem    (36),
                -- 37-47 unused --

        inappropriateAuthentication  (48),
        invalidCredentials           (49),
        insufficientAccessRights     (50),
        busy                         (51),
        unavailable                  (52),
        unwillingToPerform           (53),
        loopDetect                   (54),
                -- 55-63 unused --
        namingViolation              (64),
        objectClassViolation         (65),
        notAllowedOnNonLeaf          (66),
        notAllowedOnRDN              (67),
        entryAlreadyExists           (68),
        objectClassModsProhibited    (69),
```

```
                                        -- 70 reserved for CLDAP --
                        affectsMultipleDSAs     (71), -- new
                                        -- 72-79 unused --
                        other                   (80) },
                        -- 81-90 reserved for APIs --
                matchedDN       LDAPDN,
                errorMessage    LDAPString,
                referral        [3] Referral OPTIONAL }

        Referral ::= SEQUENCE OF LDAPURL

        LDAPURL ::= LDAPString -- limited to characters permitted in URLs

        Controls ::= SEQUENCE OF Control

        Control ::= SEQUENCE {
                controlType             LDAPOID,
                criticality             BOOLEAN DEFAULT FALSE,
                controlValue            OCTET STRING OPTIONAL }

        BindRequest ::= [APPLICATION 0] SEQUENCE {
                version                 INTEGER (1 .. 127),
                name                    LDAPDN,
                authentication          AuthenticationChoice }

        AuthenticationChoice ::= CHOICE {
                simple                  [0] OCTET STRING,
                                        -- 1 and 2 reserved
                sasl                    [3] SaslCredentials }

        SaslCredentials ::= SEQUENCE {
                mechanism               LDAPString,
                credentials             OCTET STRING OPTIONAL }

        BindResponse ::= [APPLICATION 1] SEQUENCE {

    COMPONENTS OF LDAPResult,
                serverSaslCreds     [7] OCTET STRING OPTIONAL }

        UnbindRequest ::= [APPLICATION 2] NULL

        SearchRequest ::= [APPLICATION 3] SEQUENCE {
                baseObject      LDAPDN,
```

```
        scope               ENUMERATED {
                baseObject                  (0),
                singleLevel                 (1),
                wholeSubtree                (2) },
        derefAliases        ENUMERATED {
                neverDerefAliases           (0),
                derefInSearching            (1),
                derefFindingBaseObj         (2),
                derefAlways                 (3) },
        sizeLimit           INTEGER (0 .. maxInt),
        timeLimit           INTEGER (0 .. maxInt),
        typesOnly           BOOLEAN,
        filter              Filter,
        attributes          AttributeDescriptionList }

Filter ::= CHOICE {
        and                 [0] SET OF Filter,
        or                  [1] SET OF Filter,
        not                 [2] Filter,
        equalityMatch       [3] AttributeValueAssertion,
        substrings          [4] SubstringFilter,
        greaterOrEqual      [5] AttributeValueAssertion,
        lessOrEqual         [6] AttributeValueAssertion,
        present             [7] AttributeDescription,
        approxMatch         [8] AttributeValueAssertion,
        extensibleMatch [9] MatchingRuleAssertion }

SubstringFilter ::= SEQUENCE {
        type                AttributeDescription,
        -- at least one must be present
        substrings          SEQUENCE OF CHOICE {
                initial [0] LDAPString,
                any         [1] LDAPString,
                final       [2] LDAPString } }

MatchingRuleAssertion ::= SEQUENCE {
        matchingRule        [1] MatchingRuleId OPTIONAL,
        type                [2] AttributeDescription OPTIONAL,
        matchValue          [3] AssertionValue,
        dnAttributes        [4] BOOLEAN DEFAULT FALSE }

SearchResultEntry ::= [APPLICATION 4] SEQUENCE {
        objectName          LDAPDN,
```

```
        attributes          PartialAttributeList }

PartialAttributeList ::= SEQUENCE OF SEQUENCE {
        type       AttributeDescription,
        vals       SET OF AttributeValue }

SearchResultReference ::= [APPLICATION 19] SEQUENCE OF LDAPURL

SearchResultDone ::= [APPLICATION 5] LDAPResult

ModifyRequest ::= [APPLICATION 6] SEQUENCE {
        object              LDAPDN,
        modification        SEQUENCE OF SEQUENCE {
                operation           ENUMERATED {
                                            add         (0),
                                            delete      (1),
                                            replace (2) },
                modification        AttributeTypeAndValues } }

AttributeTypeAndValues ::= SEQUENCE {
        type       AttributeDescription,
        vals       SET OF AttributeValue }

ModifyResponse ::= [APPLICATION 7] LDAPResult

AddRequest ::= [APPLICATION 8] SEQUENCE {
        entry               LDAPDN,
        attributes          AttributeList }

AttributeList ::= SEQUENCE OF SEQUENCE {
        type       AttributeDescription,
        vals       SET OF AttributeValue }

AddResponse ::= [APPLICATION 9] LDAPResult

DelRequest ::= [APPLICATION 10] LDAPDN

DelResponse ::= [APPLICATION 11] LDAPResult

ModifyDNRequest ::= [APPLICATION 12] SEQUENCE {
        entry               LDAPDN,
        newrdn              RelativeLDAPDN,
        deleteoldrdn        BOOLEAN,
```

```
                newSuperior         [0] LDAPDN OPTIONAL }

        ModifyDNResponse ::= [APPLICATION 13] LDAPResult

        CompareRequest ::= [APPLICATION 14] SEQUENCE {
                entry               LDAPDN,
                ava                 AttributeValueAssertion }

        CompareResponse ::= [APPLICATION 15] LDAPResult

        AbandonRequest ::= [APPLICATION 16] MessageID

        ExtendedRequest ::= [APPLICATION 23] SEQUENCE {
                requestName         [0] LDAPOID,
                requestValue        [1] OCTET STRING OPTIONAL }

        ExtendedResponse ::= [APPLICATION 24] SEQUENCE {
                COMPONENTS OF LDAPResult,
                responseName        [10] LDAPOID OPTIONAL,
                response            [11] OCTET STRING OPTIONAL }
```

完