Handle系统协议(版本2.1)规范

英文原文：<https://www.rfc-editor.org/rfc/pdfrfc/rfc3652.txt.pdf>

本备忘录的状态

本备忘录为互联网社区提供信息。它没有规定任何类型的互联网标准。本备忘录的分发是无限制的。

版权声明

版权所有(C)互联网协会(2003)。 保留所有权利。

IESG说明

IETF和IRTF中的几个小组讨论了Handle系统及其与现有标识符系统的关系。IESG希望指出，这些讨论并没有导致IETF在所描述的Handle系统上达成一致，也没有导致它如何适合用于标识符的IETF体系结构。虽然已经讨论过将handle作为URI的一种形式，特别是作为URN，但是这些文档描述了名称空间和标识符在互联网上如何工作的另一种观点，并包含了可能与IETF共识视图不匹配的现有系统的特征。

摘要

Handle系统是一种通用的全局名称服务，它允许在公共互联网上进行安全的名称解析和管理。本文档描述了用于客户机软件访问handle解析和管理的Handle系统的协议。协议详细说明了客户机软件对于给定handle定位负责的Handle服务器的过程。它还定义了客户机和服务器之间对任意handle操作交换的消息。

目录

1.概述。。。。。。。。。。。。。。。。。。。。。。。。。。。3.

2.协议元素。。。。。。。。。。。。。。。。。。。。。。。4

2.1。约定。。。。。。。。。。。。。。。。。。。。。。。4

2.1.1。数据传输顺序………4

2.1.2。传输层…………5

2.1.3。字符格………………6

2.1.4。标准字符串类型:UTF8字符串…7

2.2。常见的元素。。。。。。。。。。。。。。。。。。7

2.2.1。邮件信封……………8

2.2.2。消息头……………11

2.2.3。消息正文………………17

2.2.4。消息凭据…………18

* 1. 消息传播。。。。。。。。。。。。。。。。。。20.

3.处理协议操作…………21

3.1。客户引导。。。。。。。。。。。。。。。。。。21

3.1.1。全局handle注册表及其服务信息………………21

3.1.2。 定位Handle系统服务组件。22

3.1.3。 选择负责任的服务器……23

3.2 查询操作。。。。。。。。。。。。。。。。。。。。。23

3.2.1之上。 查询请求…………24

3.2.2成功的查询响应……25

3.2.3。 不成功的查询响应……26

3.3。 来自服务器的错误响应……26

3.4。 服务推荐。。。。。。。。。。。。。。。。。。。。27

3.5。 客户机身份验证。。。。。。。。。。。。。。。28

3.5.1。 从服务器到客户机的挑战…29

3.5.2。 从客户机到服务器的挑战-响应…30.

3.5.3。 挑战-响应验证-请求…33

3.5.4。 挑战-响应验证-响应…33

3.6。 处理政府。。。。。。。。。。。。。。。。。。34

3.6.1。 添加handle值…………34

3.6.2 删除handle值…………35

3.6.3。 修改handle值………36

3.6.4。 创建handle……………37

3.6.5。 删除handle………………39

3.7。 命名权(NA)管理………40

3.7.1。 在命名权威下的列表handle…40

3.7.2章。 在命名权威下列出子命名权威……41

3.8。 会话和会话管理……42

3.8.1。 会话设置请求………43

3.8.2。 会话设置响应………46

3.8.3。 会话密钥交换……47

3.8.4。 会话终止…………48

* 1. 实现指南。。。。。。。。。。。。。。。。。。。48

4.1。服务器实现。。。。。。。。。。。。。。。48

4.2。客户机实现。。。。。。。。。。。。。。。49

* 1. 安全方面的考虑。。。。。。。。。。。。。。。。。49
  2. 确认。。。。。。。。。。。。。。。。。。。。50
  3. 信息参考。。。。。。。。。。。。。。50
  4. 作者的地址。。。。。。。。。。。。。。。。。。。52
  5. 完整的版权声明。。。。。。。。。。。。。。。。53

# 概述

Handle系统为互联网提供了通用的、安全的全局名称服务。它最初是由Robert Kahn和Robert Wilensky在1995年的一篇论文中构想和描述的。Handle系统定义了一个客户机服务器协议，其中客户机软件通过网络提交请求到Handle服务器。每个请求描述要在服务器上执行的操作。服务器将处理请求并返回指示操作结果的消息。本文档指定用于客户机软件访问Handle服务器以进行handle解析和管理的协议。它不包括用于管理Handle服务器的协议的描述。关于管理协议的讨论超出了本文档的范围，将在单独的文档中提供。本文档假设读者熟悉“Handle系统概述”[1]中介绍的Handle系统的基本概念，以及“Handle系统名称空间和服务定义”[2]中给出的数据模型和服务定义。

Handle系统由一组在[2]中定义的服务组件组成。从客户机角度看，Handle系统是一个保存handle的分布式数据库。Handle系统下的不同handle可以由位于不同网络位置的不同Handle服务器维护。Handle协议详细说明了对于任意给定handle，客户机定位负责的Handle服务器的过程。它还定义了客户机和服务器之间为任意handle操作交换的消息。

Handle协议的一些关键方面包括:

1. Handle协议支持handle解析和管理。协议遵循[2]中定义的数据和服务模型。
2. 客户机可以基于服务器的数字签名对任何服务器响应进行身份验证。
3. 服务器可以通过handle身份验证协议认证其客户机为handle管理员。handle身份验证协议是一个挑战-响应协议，它支持基于公钥和秘钥（secret-key）的身份验证。
4. 可以在客户机和服务器之间建立一个会话，以便在多个操作之间共享身份验证信息和网络资源(例如TCP连接)。可以建立会话密钥来实现数据完整性和机密性。
5. 可以扩展该协议以支持新操作。控制器可用于扩展现有操作。协议被定义为允许未来的向后兼容性。
6. 分布式服务架构。支持不同服务组件之间的服务引用。
7. Handle及其数据类型基于ISO-10646 (Unicode 2.0)字符集。根据Handle协议，UTF-8[3]是强制编码。

本文档中指定的Handle协议(版本2.1)与以前的版本相比有了很大的变化。由于Handle系统数据模型和服务模型中的更改，这些更改是必要的。实现此协议的服务器可以通过检查消息信封中指定的协议版本来继续支持协议的早期版本(参见2.2.1节)。

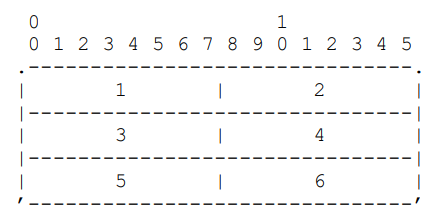
# 协议元素

## 约定

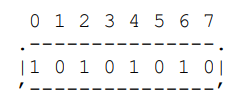
Handle协议遵循以下约定，以确保不同实现之间的互操作性。

### 2.1.1。 数据传输顺序

数据包的传输顺序遵循网络字节顺序(也称为大端字节[11])。也就是说，当一个数据图由一组字节组成时，这些字节的传输顺序遵循它们从左到右、从上到下的自然顺序，就像它们在英语中被读到的那样。例如，在下面的图中，字节按编号的顺序传送。



如果一个字节代表一个数字量，那么最左边的位就是最大的位。例如，下图表示值170 (十进制)。



类似地，当一个多字节字段表示一个数值量时，最左边的位元就是最大的位元，整个字段的最大的字节先传送。

### 2.1.2。 传输层

Handle协议的设计使得消息可以通过UDP作为单独的数据传递，也可以通过TCP连接作为连续的字节流传递。UDP和TCP的推荐端口号都是2641。

UDP使用

UDP携带的消息被限制为512字节(不包括IP或UDP头)。较长的消息必须被分割为UDP数据包，每个数据包在消息信封中携带一个适当的序列号(参见2.2.1节)。

最佳的重传策略将根据网络或服务器性能的不同而有所不同，但推荐如下:

1. 在向相同的服务器地址重复请求之前，客户机应该尝试其他服务器或服务接口。
2. 如果可能，重传间隔应该基于之前的统计。为了防止网络拥塞，应避免过于激进的重传。推荐的重传间隔为2-5秒。

1. 在传输大量数据时，应该尽可能地实现TCP友好的拥塞控制，如拥塞管理器[12]的接口，以避免对基于TCP的应用程序不公平地消耗带宽。拥塞控制的细节将在一个单独的文档中讨论。

TCP使用

Handle协议下的消息可以直接映射到TCP字节流。但是，每个消息的大小受到4字节无符号整数范围的限制。较长的消息可能在传输之前被分割成多个消息，并在接收端重新组装。

推荐几种连接管理策略:

1. 服务器应该支持多个连接，不应该阻塞等待TCP数据的其他活动。
2. 默认情况下，服务器应该在完成请求后关闭连接。但是，如果请求保持连接打开，服务器应该假设客户机将启动连接关闭。

2.1.3。 字符案例

handle是基于ISO-10646字符集的字符串，必须用UTF-8编码。默认情况下，handle字符在Handle协议下被视为大小写敏感的。但是，Handle服务可以以不区分大小写处理ASCII字符的方式实现。例如，全局handle注册表(GHR)提供了一个Handle服务，其中ASCII字符以不区分大小写的方式处理。这表明任何命名权威中的ASCII字符都是不区分大小写的。

在不区分大小写的Handle服务器下创建handle时，应该保留它们的原始大小写。为了避免混淆，服务器应该避免创建字符串与现有handle匹配的handle，忽略大小写差异。例如，如果已经创建了handle“X/Y”，则服务器应该拒绝创建handle“x/y”或其任何大小写变化的任何请求。

### 2.1.4。 标准字符串类型:UTF8字符串

handle在Handle协议下以**UTF8字符串**的形式传输。在整个文档中，UTF8字符串表示由一个4字节无符号整数和一个UTF-8编码的字符串组成的数据类型。前导整数指定字符串的字节数。

## 常见的元素

系统协议下交换的每条消息由四部分组成(图2.2)。其中一些部分(例如，消息正文)可能是空的，这取决于协议操作。

消息信封必须始终存在。它的固定大小是20个字节。消息信封不携带任何应用层信息，主要用于帮助传递消息。

消息信封中的内容不受消息凭据中的数字签名的保护。

消息头也必须始终存在。它的大小固定为24字节，并保存客户机和服务器之间交换的所有消息的公共数据字段。其中包括操作代码、响应代码和每个协议操作的控制选项。消息头中的内容受消息凭据中的数字签名保护。

消息体包含特定于每个协议操作的数据。它的格式根据消息头中的操作代码和响应代码而变化。消息体可能是空的。消息正文中的内容受消息凭据中的数字签名保护。

消息凭据为客户机和服务器之间交换的任何消息提供传输安全机制。非空消息凭据可以包含来自消息发起者的数字签名或基于预先建立的会话密钥的单向消息验证码(MAC)。消息凭据可用于对客户机和服务器之间的消息进行身份验证。它还可以用来检查数据传输后的完整性。

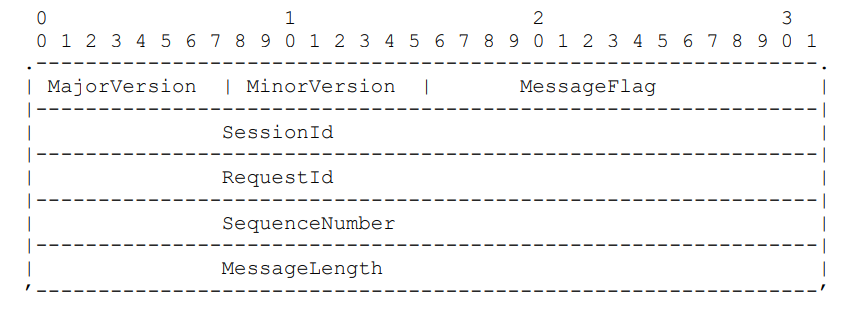
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
| |--------------- | | 。 |  |  |
| | |  | | | ;用于正确消息传递的消息信封。 | |
| | | 消息信封 | | | ;在消息凭据中，不受数字签名保护。 | |
| | |  | | |  | |
| | |  | | |  |  |
| |--------------- | | | |  |  |
| | |  | | | ;所有handle操作的公共数据字段。 | |
| | | 消息头 | | |  |  |
| | |  | | |  |  |
| |--------------- | | | |  |  |
| | |  | | | ;定义请求/响应每个的数据字段。 | |
| | | 消息体 | | |  |  |
| | |  | | |  |  |
| |--------------- | | | |  |  |
| | |  | | | ;包含关于消息头和消息体的 | |
| | | 消息凭据 | | | ;数字签名或消息认证码(MAC)。 | |
| | |  | | |  | |
| |--------------- | | ' |  |  |

图2.2:Handle协议下的消息格式

### 2.2.1。 消息信封

每个消息都以Handle协议下的消息信封开头。如果必须在消息传输之前截断消息，则每个截断的部分也必须以消息信封开头。

消息信封允许在接收端重新组装消息。它的固定大小为20个字节，由7个字段组成:



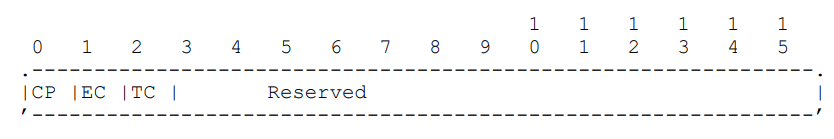
#### 2.2.1.1。 <MajorVersion>和<MinorVersion>

<MajorVersion>和<MinorVersion>用于标识Handle协议的版本。它们中的每一个都被定义为一个单字节无符号整数。该规范定义了协议版本，其<MajorVersion>为2，<MinorVersion>为1。

<MajorVersion>和<MinorVersion>是为了允许将来的向后兼容性而设计的。<MajorVersion>的差异表示协议格式的主要差异，而使用<MajorVersion>的一方必须升级其软件以确保精确通信。<MinorVersion>中的增量是在协议中添加了附加功能而没有对消息格式进行任何重大更改时产生的。

#### 2.2.1.2 . <MessageFlag>

<MessageFlag>由两个定义如下的字节组成:



第0位是CP(压缩)标志，它指示消息(不包括消息信封)是否被压缩。如果将CP位设置为1，则消息将被压缩。否则，消息不会被压缩。Handle协议使用与FTP协议[8]相同的压缩方法。

第1位是EC(加密)标志，指示消息(不包括消息信封)是否加密。EC位应该只在已建立的会话下设置，其中有一个会话密钥。如果将EC位设置为1，则使用会话密钥对消息进行加密。否则消息是不加密的。

第2位是TC (TrunCated)标志，它指示这是否是截断的消息。当通过UDP协议传输大消息时，消息截断是最常见的。消息截断(或碎片)的详细信息将在第2.3节中讨论。

3到15位目前是保留的，必须设置为0。

#### 2.2.1.3 . <SessionId>

<SessionId>是一个4字节无符号整数，用于标识客户机和服务器之间的通信会话。

Session和它的<SessionId>是由服务器分配的，可以是在客户机发出显式请求时分配，也可以是在期望多个消息交换满足客户机请求时分配。例如，如果必须对客户机进行身份验证，服务器将在其响应中分配唯一的<SessionId>。客户机可以显式地要求服务器将会话设置为类似于SSL[4]的虚拟专用通信通道。来自没有建立会话的客户机的请求必须将其<SessionId>设置为0。服务器必须为每个新会话分配一个惟一的非零<SessionId>。它还负责在一段时间后终止那些不使用的会话。

对于在已建立的会话下交换的消息，客户机和服务器都必须维护相同的<SessionId>。一条<SessionId>为0的消息表示没有建立任何会话。

会话及其状态信息可以在多个handle操作之间共享。它们也可以在多个TCP连接上共享。一旦建立了会话，客户机和服务器都必须根据<SessionId>维护它们的状态信息。状态信息可以包括对话的阶段、另一方的身份验证信息和为消息加密或身份验证而建立的会话密钥。这些细节将在第3.8节中讨论。

#### 2.2.1.4。<RequestId>

来自客户机的每个请求由<RequestId>标识，这是客户机设置的4字节无符号整数。每个<RequestId>必须与来自同一客户机的所有其他未完成请求相同。<RequestId>允许客户机跟踪其请求，来自服务器的任何响应必须包含正确的<RequestId>。

#### 2.2.1.5。<SequenceNumber>

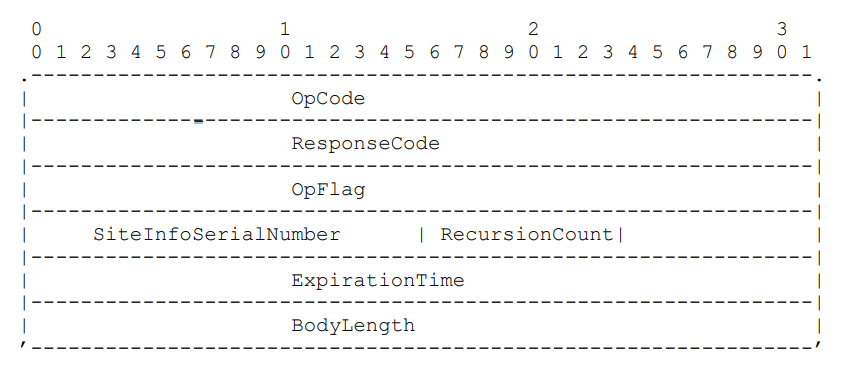
Handle协议下的消息在传输过程中可能会被截断(例如，在UDP协议下)。<SequenceNumber>是一个4字节的无符号整数，用作跟踪原始消息的每个截断部分的计数器。消息接收者可以根据<SequenceNumber>重新组装原始消息。对于每个消息，<SequenceNumber>必须以0开头。每个截断的消息必须在消息信封中设置其TC标志。未被截断的消息必须将其<SequenceNumber>设置为0。

#### 2.2.1.6。<MessageLen>

一个4字节无符号整数，指定任何消息的字节的总数，不包括消息信封中的字节。在Handle协议下交换的任何单个消息的长度都受到4字节无符号整数范围的限制。较长的数据可以用公共的<RequestId>作为多个消息进行传输。

### 2.2.2。 消息头

消息头包含任何协议操作之间的公共数据元素。它的固定大小为24个字节，由8个字段组成。



每条未被截断的消息都必须有一个消息头。如果必须截断消息以进行传输，则消息头必须出现在消息的第一个截断部分。

这与消息信封不同，消息信封出现在消息的每个截断部分。

#### 2.2.2.1。<OpCode>

<OpCode>代表操作代码，它是一个指定预期操作的四字节无符号整数。下表列出了所有实现必须支持的<OpCode>，以便符合基本协议规范。每个操作代码都有一个符号名，在整个文档中都使用这个符号名，以便于引用。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | | | |  |
| Op\_Code | | | 符号名称 | 备注 | | |  |
| --------- | | | ------------- | ------ | | |  |
| 0 | | | OC\_RESERVED | 保留 | | |  |
| 1 | | | OC\_RESOLUTION | Handle查寻 | | | |
| 2 | | | OC\_GET\_SITEINFO | 获得HS\_SITE值 | | | |
| 100 | | | OC\_CREATE\_HANDLE | 创建新的处理 | | | |
| 101 | | | OC\_DELETE\_HANDLE | 删除现有的处理 | | | |
| 102 | | | OC\_ADD\_VALUE | 添加处理值(s) | | | |
| 103 | | | OC\_REMOVE\_VALUE | 删除处理值(年代) | | | |
| 104 | | | OC\_MODIFY\_VALUE | 修改处理值(s) | | | |
| 105 | | | OC\_LIST\_HANDLE | 列表handle | | | |
| 106 | | | OC\_LIST\_NA | 列表sub-naming当局 | | | |
| 200 | | | OC\_CHALLENGE\_RESPONSE | 应对挑战 | | | |
| 201 | | | OC\_VERIFY\_RESPONSE | 验证挑战的反应 | | | |
| 300 | | |  |  | | | |
| ： | | | {300-399保留用于Handle服务器管理} | | | | |
| 399 | | |  |  | | | |
| 400 | OC\_SESSION\_SETUP | | | 会话设置请求 | | | |
| 401 | OC\_SESSION\_TERMINATE | | | 会 | 会话终止请求 | | |
| 402 | OC\_SESSION\_EXCHANGEKEY | | | 会 | 会话密钥交换 | | |

每个<OpCode>s的详细描述可以在本文档的第3节中找到。通常，客户机使用<OpCode>来告诉服务器他们想要完成什么样的handle操作。来自服务器的响应必须保持与原始请求相同的<OpCode>，并使用<ResponseCode>来指示结果。

#### 2.2.2.2 . <ResponseCode>

<ResponseCode>是一个4字节的无符号整数，由服务器给出以指示服务请求的结果。Handle协议中使用的<ResponseCode>的列表在下表中定义。每个响应代码都有一个符号名，在整个文档中都使用这个符号名，以便于参考。

|  |  |  |
| --- | --- | --- |
| 代码》 | 符号名称 | 备注 |
| --------- | ------------- | ------ |
| 0 | RC\_RESERVED | 用于请求 |
| 1 | RC\_SUCCESS | 成功响应 |
| 2 | RC\_ERROR | 一般的错误 |
| 3. | RC\_SERVER\_BUSY | 服务器太忙，无法响应 |
| 4 | RC\_PROTOCOL\_ERROR | 损坏或未被承认的消息 |
| 5 | RC\_OPERATION\_DENIED | 不支持的操作 |
| 6 | RC\_RECUR\_LIMIT\_EXCEEDED | 太多的递归请求 |

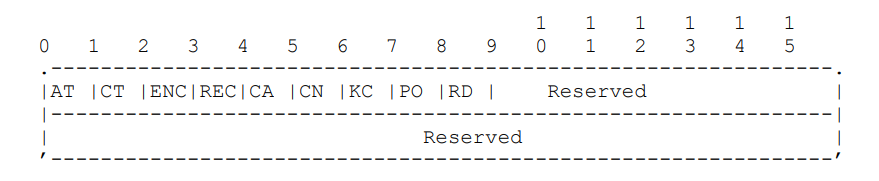
|  |  |  |
| --- | --- | --- |
| 100 | RC\_HANDLE\_NOT\_FOUND | Handle未找到 |
| 101 | RC\_HANDLE\_ALREADY\_EXIST | Handle已经存在 |
| 102 | RC\_INVALID\_HANDLE | 编码(或语法)错误 |
| 200 | RC\_VALUE\_NOT\_FOUND | 值未找到 |
| 201 | RC\_VALUE\_ALREADY\_EXIST | 值已经存在 |
| 202 | RC\_VALUE\_INVALID | 无效的handle值 |
| 300 | RC\_EXPIRED\_SITE\_INFO | SITE\_INFO已经过时了 |
| 301 | RC\_SERVER\_NOT\_RESP | 服务器不负责 |
| 302 | RC\_SERVICE\_REFERRAL | 服务器推荐 |
| 303 | RC\_NA\_DELEGATE | 进行命名权威委托 |
| 400 | RC\_NOT\_AUTHORIZED | 未授权或允许 |
| 401 | RC\_ACCESS\_DENIED | 无法查阅资料 |
| 402 | RC\_AUTHEN\_NEEDED | 需要身份验证 |
| 403 | RC\_AUTHEN\_FAILED | 未能进行身份验证 |
| 404 | RC\_INVALID\_CREDENTIAL | 无效的证书 |
| 405 | RC\_AUTHEN\_TIMEOUT | 身份验证超时 |
| 406 | RC\_UNABLE\_TO\_AUTHEN | 无法身份验证 |
| 500 | RC\_SESSION\_TIMEOUT | 会话过期 |
| 501 | RC\_SESSION\_FAILED | 无法建立会话 |
| 502 | RC\_NO\_SESSION\_KEY | 还没有可用的会话 |
| 503 | RC\_SESSION\_NO\_SUPPORT | 会话不支持 |
| 504 | RC\_SESSION\_KEY\_INVALID | 无效的会话密钥 |
| 900 | RC\_TRYING | 请求正在处理 |
| 901 | RC\_FORWARDED | 请求转发到另一个服务器 |
| 902 | RC\_QUEUED | 请求排队等待稍后处理 |

10000以下的响应码预留给系统使用。任何响应代码低于10000但未在上面列出的消息都应视为未知错误。10000以上的响应代码是用户定义的，可以用于应用程序特定的目的。

这些<ResponseCode>的详细描述可以在本文档的第3节中找到。通常，来自客户机的任何请求都必须将其<ResponseCode>设置为0。来自服务器的响应消息必须具有非零的<ResponseCode>来指示结果。例如，将<ResponseCode>设置为RC\_SUCCESS的服务器的响应消息表示服务器已经成功地满足了客户机的请求。

#### 2.2.2.3。<OpFlag>

<OpFlag>是一个32位的位掩码，它定义了协议操作的各种控制选项。下图显示了<OpFlag>字段中每个选项标志的位置。



AT - 权威位。将AT位设置为1的请求指示应该将请求定向到主要服务站点(而不是任何镜像站点)。将AT位设置为1的响应消息表示消息是从主服务器(在主服务站点内)返回的。

CT – 证明位。将CT位设置为1的请求要求服务器使用其数字签名对其响应进行签名。将CT位设置为1的响应表示消息已签名。如果请求的CT位设置为1，则服务器必须对其响应进行签名。如果服务器未能在其响应中提供有效签名，则客户端应放弃响应并将请求视为失败。

ENC - 加密位。ENC位设置为1的请求要求服务器使用预先建立的会话密钥加密其响应。

REC - 递归位。如果请求必须由另一个Handle服务器处理，则将REC位设置为(1)的请求要求服务器代表客户机转发查询。服务器可以通过将请求转发到适当的Handle服务器并将任何结果传递回客户机来满足请求。服务器还可以通过发送一个将<ResponseCode>设置为RC\_SERVER\_NOT\_RESP的响应来拒绝任何此类请求。

CA - 缓存身份验证。将CA位设置为1的请求要求缓存服务器(如果有的话)代表客户机对任何服务器响应进行身份验证(例如，验证服务器的签名)。将CA位设置为1的响应表示缓存服务器已经对响应进行了身份验证。

CN - 连续位。将CN位设置为1的消息告诉消息接收者，属于同一请求(或响应)一部分的更多消息将随后发送。如果请求(或响应)的数据太大，无法容纳任何单个消息，并且必须将其分割为多个消息，就会发生这种情况。

KC - 保持连接位。设置了KC位的消息要求消息接收者保持TCP连接打开(在响应发送回来之后)。这允许将相同的TCP连接用于多个handle操作。

PO – 仅公开位。仅供查询操作使用。将PO位设置为1的查询请求表明客户机只请求具有PUB\_READ权限的句柄值。将PO位设置为0的请求请求所有的handle值，而不考虑它们的读权限。如果有任何handle值需要ADMIN\_READ权限，则服务器必须对客户机进行身份验证，使其成为handle管理员。

RD – 请求摘要位。将RD位设置为1的请求要求服务器在其响应中包含消息请求摘要。将RD位设置为1的响应消息指示消息正文中的第一个字段包含原始请求的消息摘要。消息摘要可用于检查服务器响应的完整性。这些细节将在本文后面讨论。

<OpFlag>字段中的所有其他位都是保留的，必须设置为0。

通常，服务器必须遵守请求中指定的<OpFlag>。如果无法满足所请求的选项，服务器应该返回一个错误消息，并使用上一节定义的适当<ResponseCode>。

#### 2.2.2.4。<SiteInfoSerialNumber>

<SiteInfoSerialNumber>是一个双字节无符号整数。请求中的<SiteInfoSerialNumber>是指客户机(访问服务器)使用的HS\_SITE值的<SerialNumber>。服务器可以检查请求中的<SiteInfoSerialNumber>，以确定客户机是否有最新的服务信息。

如果可能，服务器应该满足客户机的请求，即使客户机使用的服务信息已经过期。但是，响应消息应该在<SiteInforSerialNumber>字段中指定服务信息的最新版本。具有过时服务信息的客户机可以从全局handle注册表更新服务信息。如果服务器由于服务信息过期而无法满足客户机的请求，则应该拒绝该请求并返回一条错误消息，并将<ResponseCode>设置为RC\_EXPIRED\_SITE\_INFO。

#### 2.2.2.5。<RecursionCount>

<RecursionCount>是一个单字节无符号整数，它指定了服务递归的数量。如果服务器必须将客户机的请求转发给另一台服务器，就会发生服务递归。任何直接来自客户机的请求必须将其<RecursionCount>设置为0。如果服务器必须代表客户机发送递归请求，则必须将<RecursionCount>增加1。来自服务器的任何响应都必须保持与请求中的相同的<RecursionCount>。为了防止服务递归的无限循环，服务器应该是可配置的，以在<RecursionCount>达到某个值时停止发送递归请求。

#### 2.2.2.6。<ExpirationTime>

<ExpirationTime>是一个4字节的无符号整数，它指定了相对于1970年1月1日GMT的消息应该被认为过期的时间(以秒为单位)。如果预期没有到期，则将其设置为零。

#### 2.2.2.7。<BodyLength>

<BodyLength>是一个4字节无符号整数，它指定消息体中的字节数。<BodyLength>不计算消息头中的字节或消息凭据中的字节。

### 2.2.3消息体

消息正文始终位于消息头之后。消息体中的字节数可以通过消息头中的<BodyLength>确定。消息体可能是空的。消息体的确切格式取决于消息头中的<OpCode>和<ResponseCode>。本文档第3节描述了每个<OpCode>和<ResponseCode>下的消息体的详细信息。

对于任何响应消息，如果消息头的RD位(在<OpFlag>中)设置为1，则消息正文必须从原始请求的消息摘要开始。消息摘要的定义如下:

<RequestDigest>::= <DigestAlgorithmIdentifier> <MessageDigest>

这里

**<**DigestAlgorithmIdentifier**>**

一个字节，用于识别生成消息摘要的算法。如果将字节设置为1，则使用MD5[9]算法生成摘要。如果字节设置为2，则使用SHA-1[10]算法。

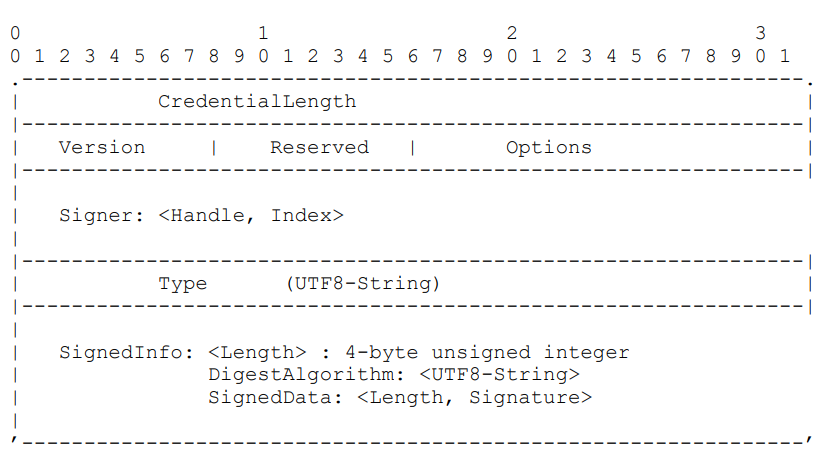
**<**MessageDigest**>**

消息摘要本身。它是根据原始请求的消息头和消息体计算的。字段的长度根据摘要算法确定。MD5算法的长度为16字节。对于SHA-1，长度是20字节。

消息体在传输过程中可能被截断成多个部分(例如，通过UDP)。此类消息的收件人可以根据消息信封中的<SequenceNumber>从每个部分重新组装消息正文。

### 2.2.4消息凭据

消息凭据主要用于携带由消息发送者签名的任何数字签名。如果建立了会话密钥，它还可以携带消息验证码(MAC)。消息凭据用于保护消息头和消息正文中的内容在传输期间不受篡改。消息凭据的格式设计为在语义上与PKCS#7[5]兼容。每个消息凭据由以下字段组成:



这里

**<**CredentialLength**>**

一个4字节无符号整数，指定消息凭据中的字节数。如果消息没有消息凭据，则必须将其设置为零。

<Version>

一个字节，用于标识消息凭据的版本号。本文档中指定的版本号为零。

<Reserved>

必须设置为零的一个字节。

<Options>

两个字节保留给不同的加密选项。

<Signer>::= <HANDLE>

<INDEX>

一个根据<handle>和<index>对handle值的引用。handle值可以包含用于验证数字签名的公钥或X.509证书。

<Type>

一个UTF8字符串，表示<SignedInfo>字段中的内容类型(如下所述)。如果<SignedInfo>包含消息摘要，则可能包含HS\_DIGEST;如果<SignedInfo>包含消息验证码(MAC)，则可能包含HS\_MAC。如果<SignedInfo>包含数字签名，<Type>字段将指定签名算法标识符。例如，将<Type>字段设置为HS\_SIGNED\_PSS， <SignedInfo>字段将包含使用RSA-PSS算法[16]生成的数字签名。如果<Type>字段设置为HS\_SIGNED， <SignedInfo>字段将包含从DSA公钥对生成的数字签名。

**<**SignedInfo**>**:: = <Length>

**<**DigestAlgorithm**>**

**<**SignedData**>**

这里

<Length>

一个4字节的无符号整数，它指定了<SignedInfo>字段中的字节数。

**<**DigestAlgorithm**>**

一个UTF8字符串，它引用用于生成数字签名的摘要算法。例如，值“SHA-1”表示使用SHA-1算法生成签名的消息摘要。

**<**SignedData**>**:: = <LENGTH>

<SIGNATURE>

这里

<LENGTH>

一个4字节的无符号整数，它指定了<SIGNATURE>中的字节数。

<SIGNATURE>

在消息头和消息体上包含数字签名或MAC。签名的语法和语义依赖于<Type>字段以及<Signer>字段中引用的公钥。例如，如果<Type>字段是“HS\_SIGNED”，而<Signer>字段引用的公钥是DSA[6]公钥，则签名将是参数R和S的ASN.1字节字符串表示形式，如[7]中所述。如果<Signer>字段引用包含X.509证书的handle值，则签名应该按照RFC 3279和RFC 3280[14,15]进行编码。

消息凭据可以包含使用预先建立的会话密钥生成的消息验证码(MAC)。在这种情况下，<Signer>字段必须将其<handle>设置为零长度的UTF8字符串，将其<INDEX>设置为消息信封中指定的<SessionId>。<Signature>字段必须在其<SIGNATURE>字段中包含MAC。MAC是会话密钥、<Message Header>、<MessageBody>和会话密钥连接的单向散列的结果。

响应消息中的消息凭据可以包含由服务器签名的数字签名。可以在客户机用于向服务器发送请求的服务信息中找到服务器的公钥。在这种情况下，客户机应该忽略<Signer>字段中的任何引用，并使用服务信息中的公钥来验证签名。

消息凭据还可以用于不可抵赖性目的。如果消息凭据包含服务器的数字签名，则会发生这种情况。签名可以作为证据来证明服务器已经响应客户的请求提供了服务。

消息凭据为客户机和服务器之间的任何消息的安全传输提供了一种机制。任何消息其消息头和消息体符合其消息凭据的消息都表明该消息确实来自其发送者，并确保该消息在传输过程中没有被篡改。

## 消息传输

在传输过程中，一条大的消息可能会被截断成多个包。例如，为了适应UDP包的大小限制，消息发布方必须在传输之前将任何大消息截断为多个UDP包。消息接收方必须在进一步处理之前重新组装来自这些截断的信息包的消息。必须对整个消息进行截断但消息信封除外。在发送之前，必须在每个截断的信息包前面插入一个新的消息信封。例如，包含以下内容的大型消息

.----------------------------------.

| 消息信封 | 消息头，消息体, 凭据 |

’---------------------------------’

可截断为:

.---------------------------------.

| 消息信封 1 | Truncated\_Packet 1 |

’--------------------------------’

.---------------------------------.

| 消息信封 2 | Truncated\_Packet 2 |

’--------------------------------’

……

.---------------------------------.

| 消息信封 N | Truncated\_Packet N |

’--------------------------------’

其中，“Truncated\_packet 1”，“Truncated\_packet 2”，…，而“Truncated\_packet N”则是截断消息头、消息正文和消息凭据的结果。每个“消息信封i”(在每次截断之前插入)必须将其TC标志设置为1并保持适当的序列计数(在<SequenceNumber>中)。每个“消息信封i”还必须设置其<MessageLength>以反映包的大小。这些被截断的信息包的接收者可以根据它们的<SequenceNumber>将这些信息包连接起来，从而重新组装消息。

# Handle协议操作

本节根据客户机和服务器之间交换的消息描述每个协议操作的细节。它还根据消息头中的每个<OpCode>和<ResponseCode>定义消息体的格式。

## 3.1 客户机引导

### 3.1.1全局handle注册表及其服务信息

全局Handle注册中心(GHR)的服务信息允许客户机与GHR联系，以找到负责其handle的服务组件。服务信息是一组分配给根handle“0.NA/0.NA”的HS\_SITE值，也称为根服务信息。根服务信息可以随客户机软件一起发布，也可以从http://www.handle.net的Handle系统网站下载。

根服务信息的更改由HS\_SITE值中的<SerialNumber>标识。GHR的服务器可以通过检查客户机请求中的<SerialNumber>来发现客户机使用的根服务信息是否过期。如果响应消息的<ResponseCode>是RC\_EXPIRED\_SITE\_INFO，客户机应该更新根服务信息。客户机可以从根handle获取最新的根服务信息。GHR必须使用过时的服务信息(在客户机请求中标识)中指定的公钥对根服务信息进行签名，以便客户机可以验证签名。

### 3.1.2定位Handle系统服务组件

Handle系统下的每个handle由一个惟一的Handle服务组件(例如，LHS)管理。对于任何给定的handle，都可以从其命名权威handle找到负责任的服务组件(及其服务信息)。在解析任何给定handle之前，客户机需要通过从GHR查询命名权威handle来查找负责任的服务组件。

例如，为了查找handle“1000/abc”对应的LHS，客户机软件可以向GHR查询分配给命名权威handle“0.NA/1000”的HS\_SITE(或HS\_SERV)值。HS\_SITE值集提供管理命名权威“1000”下每个handle的LHS的服务信息。如果没有找到HS\_SITE值，客户机可以检查是否有分配给命名权威handle的HS\_SERV值。 HS\_SERV值提供用于维护LHS服务信息的服务handle。 服务handle用于管理不同命名权威共享的服务信息。

客户机请求的命名权威handle可能不驻留在GHR中。 当进行命名权威委托时，就会发生这种情况。 命名权威委托发生在命名权威委托LHS管理其所有子命名权威时。 在这种情况下，委托的命名权威必须包含服务信息，即管理其子命名权威的LHS的一组HS\_NA\_DELEGATE值。

所有顶级命名权威handle都必须由GHR注册和管理。当GHR上的服务器接收到对已委托给LHS的命名权威的请求时，它必须将<ResponseCode>设置为RC\_NA\_DELEGATE的消息和来自最近的祖先命名权威的HS\_NA\_DELAGATE值。客户机可以查询HS\_NA\_DELAGATE值所描述的用于委托的命名权威handle的LHS。在实践中，通过在委托时复制自己，祖先命名权威应该使自己对GHR中的任意handle服务器可用。这将防止在查询中的命名权威和祖先命名权威未散列到同一Handle服务器时在Handle服务器(服务站点内)之间进行任何交叉查询。

### 3.1.3 选择负责的服务器

每个Handle服务组件都是根据一组HS\_SITE值定义的。每个HS\_SITE值在服务组件中定义一个服务站点。一个服务站点可能由一组Handle服务器组成。对于任何给定的handle，服务组件中负责的handle服务器都可以通过以下步骤找到:

1. 选择一个首选的服务站点。

每个服务站点都是根据HS\_SITE值定义的。HS\_SITE值可以包含<Description>或其他属性(在<AttributeList>下)来帮助选择。客户机必须为任何管理操作选择主服务站点。

1. 在服务站点中找到负责的服务器。

可以这样做:将handle中的每个ASCII字符转换为大写。根据HS\_SITE值中给出的<HashOption>，计算转换handle字符串的MD5散列。将哈希结果的最后4个字节作为带符号的整数。对HS\_SITE值中给出的整数<NumOfServer>的绝对值取模（Modulo）。结果是HS\_SITE值中列出的<ServerRecord>的序列号。例如，如果取模的结果是2，那么应该选择<HS\_SITE>中列出的第三个<ServerRecord>。<ServerRecord>为给定handle定义了负责的handle服务器。

## 3.2 查询操作

查询操作由客户机向负责的handle服务器发送查询请求和服务器向客户机返回查询结果组成。查询请求用于检索分配给任何给定handle的handle值。

### 3.2.1查询请求

任何查询请求的消息头必须将其<OpCode>设置为OC\_RESOLUTION(在2.2.2.1节中定义)，并将<ResponseCode>设置为0。

任何查询请求的消息体定义如下:

<查询请求的消息体> :: = <Handle>

**<**IndexList**>**

**<**TypeList**>**

这里

<Handle>

一个UTF8字符串(如2.1.4节中定义的)，它指定要解析的handle。

**<**IndexList**>**

一个4字节无符号整数，后跟一个4字节无符号整数数组。第一个整数表示整数数组中的整数数量。整数数组中的每个数字都是handle值索引，也对查询出的handle值引用。客户机可将第一个整数设置为0(后面是一个空数组)，以请求所有的handle值，而不管它们的索引是什么。

**<**TypeList**>**

一个4字节无符号整数，后跟一个UTF8字符串列表。第一个整数表示下面列表中UTF8字符串的数量。列表中的每个UTF8字符串指定一个数据类型。这告诉服务器返回数据类型在列表中列出的所有handle值。如果UTF8字符串以“.”(0x2E)字符结尾，服务器必须返回数据类型位于UTF8字符串中指定的类型层次结构下的所有handle值。如果第一个整数是0，则<TypeList>可能不包含UTF8字符串。在这种情况下，服务器必须返回所有的handle值，不管它们的数据类型是什么。

如果查询请求没有指定任何索引或数据类型，并且设置了PO标志(在消息头中)，则服务器将返回具有PUBLIC\_READ权限的所有handle值。客户机也可以在请求时不设置PO标志，在这种情况下，服务器将返回具有PUBLIC\_READ权限的所有handle值，以及具有ADMIN\_READ权限的所有handle值。如果查询通过值索引请求特定的handle值，而该值没有PUBLIC\_READ权限，那么即使该请求设置了PO标志，服务器也应该接受该请求(并对客户机进行身份验证)。

如果一个查询包含一个非空的<IndexList>，而一个空的<TypeList>，那么服务器应该只返回索引列在<IndexList>中的handle值。同样，如果一个查询包含一个非空的<TypeList>，但是一个空的<IndexList>，服务器应该只返回那些数据类型列在<TypeList>中的handle值。

当<IndexList>和<TypeList>字段都非空时，服务器应该返回索引列在<IndexList>中的所有handle值，以及数据类型列在<TypeList>中的所有handle值。

### 3.2.2成功查询响应

任何查询响应的消息头都必须将其<OpCode>设置为OC\_RESOLUTION。成功的查询响应必须将其<ResponseCode>设置为RC\_SUCCESS。

成功查询响应的消息体定义如下:

<成功查询响应>::= [<RequestDigest>]

<Handle>

**<**ValueList**>**

这里

**<**RequestDigest**>**

2.2.3节定义的可选字段。

<Handle>

一个UTF8字符串，它指定客户机查询的handle。

**<**ValueList**>**

一个4字节无符号整数，后跟一个handle值列表。该整数指出了列表中handle值的数目。每个handle值的编码遵循[2]中给出的规范(参见3.1节)。如果没有满足查询的handle值，则将该整数设置为0。

### 3.2.3 失败的查询响应

如果服务器不能满足客户机的请求，它必须返回一个错误消息。来自服务器的任何错误消息的一般格式在本文档3.3节中指定。

例如，如果查询的handle在数据库中不存在，服务器必须返回错误消息。错误消息将有一个空的消息体，并将其<ResponseCode>设置为RC\_HANDLE\_NOT\_FOUND。

注意，如果服务器不负责查询的handle，则服务器不应该返回RC\_HANDLE\_NOT\_FOUND消息。查询的handle可能存在，但由另一个Handle服务器(在另一个Handle服务下)管理。发生这种情况时，服务器应该发送一个服务引用(参见3.4节)，或者简单地返回一条错误消息，将<ResponseCode>设置为RC\_SERVER\_NOT\_RESP。

如果服务器太忙而无法处理请求，服务器可能会返回一条错误消息，并将<ResponseCode>设置为RC\_SERVER\_BUSY。与RC\_HANDLE\_NOT\_FOUND一样，RC\_SERVER\_BUSY消息也有一个空消息体。

如果请求请求一个既没有PUBLIC\_READ权限也没有ADMIN\_READ权限的特定handle值(通过handle值索引)，服务器应该返回一个RC\_ACCESS\_DENIED消息。

在解析期间，Handle服务器可能要求其客户机作为handle管理员进行身份验证。如果查询中的任意handle值具有ADMIN\_READ权限，但没有PUBLIC\_READ权限，则会发生这种情况。客户机身份验证的详细信息将在本文档后面描述。

## 3.3服务器错误响应

如果处理请求时遇到错误，Handle服务器将返回错误消息。来自服务器的任何错误响应必须保持与原始请求中的相同<OpCode>(在消息头中)。每个错误条件由唯一的<ResponseCode>标识，如本文档第2.2.2.2节所定义。

错误消息的消息体可能为空。否则，它包括以下数据字段(除非另有说明):

<来自服务器的错误响应消息体> ::= [<RequestDigest>]

**<**ErrorMessage**>**

**[ <**IndexList**>** ]

这里

**<**RequestDigest**>**

2.2.3节定义的可选字段。

**<**ErrorMessage**>**

解释错误的UTF8字符串。

**<**IndexList**>**

一个可选的字段。当不为空时，它由一个4字节无符号整数和一组handle值索引组成。第一个整数表示列表中的索引数量。列表中的每个索引是一个4字节无符号整数，它引用导致错误的handle值。例如，一个服务器被要求添加三个索引为1、2和3的handle值，以及已经存在索引为1和2的handle值。在这种情况下，服务器可以返回一条错误消息，将<REsponseCode>设置为RC\_VALUE\_ALREADY\_EXIST，并将索引1和2添加到<IndexList>。请注意，服务器没有义务返回可能导致错误的handle值索引的完整列表。

## 3.4服务引用

Handle服务器可以接收由其他Handle服务器或服务管理的handle请求。当发生这种情况时，服务器可以选择返回一个将客户机定向到正确的Handle服务的引用消息，或者简单地返回一个错误消息，并将<ResponseCode>设置为RC\_SERVER\_NOT\_RESP。当handle的所有权从一个Handle服务转移到另一个Handle服务时，也会发生服务引用。任何本地Handle服务也可以使用它将其服务委托给多个服务层。

服务引用的消息头必须与原始请求中的消息头保持相同的<OpCode>，并将其<ResponseCode>设置为RC\_SERVICE\_REFERRAL。

服务引用的消息体定义如下:

<服务引用的消息体> ::= [<RequestDigest>]

<ReferralHandle>

[ **<**ValueList**> ]**

这里

**<**RequestDigest**>**

2.2.3节定义的可选字段。

**<**ReferralHandle**>**

一个UTF8字符串，标识含有引用信息（即要引用的handle服务的服务信息）的handle(例如，一个服务handle)。如果<ReferralHandle>被设置为“0.NA/0.NA”，它正将客户机提交到GHR。

**<**ValueList**>**

一个可选字段，如果提供<ReferralHandle>，则该字段必须为空。当不为空时，它由一个4字节无符号整数组成，后跟一个HS\_SITE值列表。该整数指定列表中HS\_SITE值的数量。

与由任何数据类型的handle值组成的常规查询响应不同，服务引用的<ValueList>中只能有零个或多个HS\_SITE值。如果<ValueList>中的HS\_SITE值不是由任意handle维护的，<ReferralHandle>可能包含一个空的UTF8字符串。

客户机必须小心避免由服务引用引起的任何循环。客户机还负责对从服务引用获得的服务信息进行身份验证。如果<ReferralHandle>被设置为“0.NA/0.NA”，客户机应该始终使用自己的GHR服务信息副本。

## 3.5。 客户机身份验证

当使用ADMIN\_READ查询任意handle值但没有PUBLIC\_READ权限时，要求客户机将自己验证为handle管理员。对于任何需要管理员权限的handle管理请求，也需要客户机身份验证。这包括添加、删除或修改handle或handle值。

客户机身份验证由客户机和服务器之间交换的多个消息组成。这些消息包括从服务器到客户机的验证客户机的挑战、来自客户机对服务器挑战的响应，以及验证请求和响应消息（如果使用密钥验证）。身份验证期间交换的消息通过服务器分配的唯一<SessionId>关联。对于每个身份验证会话，服务器需要维护状态信息，其中包括服务器的挑战、来自客户机的挑战响应以及原始客户机请求。

身份验证从服务器的响应消息开始，其中包含对客户机的挑战。客户机必须使用挑战-响应消息来响应挑战。服务器通过验证挑战-响应中的数字签名来验证挑战-响应，或者通过向另一个Handle服务器(这里称为验证服务器，它为管理员维护着秘密密钥)发送验证请求。挑战和挑战响应的目的是向服务器证明客户机拥有handle管理员的私钥(或秘密密钥)。如果身份验证失败，将返回一个错误响应，并将<ResponseCode>设置为RC\_AUTHEN\_FAILED。

在成功的客户机身份验证之后，服务器还必须确保为请求授权了管理员。如果管理员有足够的权限，服务器将处理请求并发回结果。如果管理员没有足够的权限，服务器将返回一条错误消息，<ResponseCode> set to RC\_NOT\_AUTHORIZED。

以下部分提供了身份验证过程中交换的每个消息的详细信息。

### 3.5.1从服务器到客户机的挑战

挑战的消息头必须保持与原始请求相同的<OpCode>，并将<ResponseCode>设置为RC\_AUTH\_NEEDED。服务器必须在消息信封中分配一个非零的惟一<SessionId>来跟踪身份验证。它还必须在消息头中设置<OpFlag>(参见2.2.2.3节)的RD标志，而不管原始请求是否设置了RD位。

服务器挑战的消息体定义如下:

<服务器挑战的消息体> ::= <RequestDigest>

<Nonce>

这里

**<**RequestDigest**>**

请求消息的消息摘要，如2.2.3节中定义的。

**<**Nonce**>**

一个4字节无符号整数，后跟一个由服务器通过安全随机数生成器生成的随机字符串。该整数指定随机字符串中的字节数。随机字符串的大小应该不小于20个字节。

请注意，如果客户机没有请求服务器签署挑战，服务器将不会签署。如果客户机担心它是否与正确的服务器对话，它可以要求服务器签署<Challenge>。如果客户机请求服务器对<Challenge>进行签名，但是未能验证服务器的签名，那么客户机应该放弃服务器的响应，并重新向服务器发出请求。

### 3.5.2从客户机到服务器的挑战-响应

CHALLENGE\_RESPONSE的消息头必须将其<OpCode>设置为OC\_CHALLENGE\_RESPONSE，将其<ResponseCode>设置为0。它还必须保持来自服务器的挑战中指定的相同<SessionId>(在消息信封中)。

CHALLENGE\_RESPONSE请求的消息体定义如下:

**<**CHALLENGE\_RESPONSE> ::= <AuthenticationType>

**<**KeyHandle**>**

**<**KeyIndex**>**

**<**ChallengeResponse**>**

这里

**<**AuthenticationType**>**

一个UTF8字符串，用于标识客户机使用的身份验证密钥的类型。例如，如果客户机选择使用秘密密钥进行身份验证，则该字段被设置为“HS\_SECKEY”。如果使用公钥，则将字段设置为“HS\_PUBKEY”。

**<**KeyHandle**>**

一个UTF8字符串，用于标识持有handle管理员公钥或密钥的handle。

**<**KeyIndex**>**

一个4字节无符号整数，指定持有管理员公钥或密钥的handle值(<KeyHandle>)的索引。

**<**ChallengeResponse**>**

包含来自服务器的消息验证码(MAC)或数字签名。如果<AuthenticationType>是"HS\_SECKEY"， <ChallengeResponse>由一个字节组成，后面跟一个MAC。字节标识用于生成MAC的算法。例如，如果将第一个字节设置为0x01, MAC通过下面算法生成

MD5\_Hash(<SecretKey> + <ServerChallenge> + <SecretKey>)

其中，<SecretKey>是管理员的秘密密钥，由<KeyHandle>和<KeyIndex>引用。<ServerChallenge>是服务器挑战的消息主体部分。如果将<ChallengeResponse>中的第一个字节设置为0x02，则使用以下命令生成MAC

SHA-1\_Hash(<SecretKey> + <ServerChallenge> + <SecretKey>)

一个更安全的方法是使用HMAC[17]进行<ChallengeResponse>。可以使用<SecretKey>和<ServerChallenge>生成HMAC。将第一个字节设置为0x11的<ChallengeResponse>表明HMAC是使用MD5算法生成的。同样，将第一个字节设置为0x12的<ChallengeResponse>表明HMAC是使用SHA-1算法生成的。

如果<AuthenticationType>是"HS\_PUBKEY"，则<ChallengeResponse>包含服务器挑战消息正文部分的数字签名。签名通过两个步骤生成:首先，在要签名的blob上计算一个单向散列值。其次，散列值是使用私钥签名的。签名由一个UTF8字符串组成，该字符串指定用于签名的摘要算法，然后是服务器挑战时的签名。<KeyHandle>和<KeyIndex>指向了可以用来验证签名的管理员的公钥。

Handle管理员是根据分配给handle的HS\_ADMIN值定义的。每个HS\_ADMIN值定义授予管理员的权限集。它还提供对可用于对管理员进行身份验证的身份验证密钥的引用。如果HS\_ADMIN值的<AdminRef>字段引用持有身份验证密钥的handle值，则可以直接进行引用。还可以通过管理员组间接引用身份验证密钥。在本例中，<AdminRef>字段可能引用HS\_VLIST类型的handle值。HS\_VLIST值通过handle值引用列表定义管理员组，每个handle值引用引用一个handle管理员的身份验证密钥。

对于具有多个HS\_ADMIN值的handle，服务器必须检查每个具有足够权限的handle，以查看其<AdminRef>字段是否与<KeyHandle>和<KeyIndex>匹配。如果没有找到匹配，但是定义了管理员组，服务器必须检查<KeyHandle>和<KeyIndex>是否属于具有足够权限的任何管理员组。管理员组可以包含另一个管理员组作为成员。在浏览这些组时，服务器必须小心避免无限循环。

如果<KeyHandle>和<KeyIndex>没有被任何HS\_ADMIN值引用，或者没有被具有足够权限的管理员组引用，服务器将返回一条错误消息，并将<ResponseCode>设置为RC\_NOT\_AUTHORIZED。否则，服务器将继续对客户机进行如下身份验证:

如果<AuthenticationType>是"HS\_PUBKEY"，服务器将根据<KeyHandle>和<KeyIndex>检索管理员的公钥。公钥可用于针对服务器的<Challenge>验证<ChallengeResponse>。如果<ChallengeResponse>与<Challenge>匹配，服务器将继续处理原始请求并返回结果。否则，服务器将返回一条错误消息，并将<ResponseCode>设置为RC\_AUTHENTICATION\_FAILED。

如果<AuthenticationType>是"HS\_SECKEY"，则服务器必须向验证服务器发送验证请求;也就是说，管理<KeyHandle>引用的handle的Handle服务器。验证请求及其响应将在以下部分中定义。验证服务器将代表Handle服务器对<ChallengeResponse>和<Challenge>进行验证。

### 3.5.3挑战-响应验证请求

VERIFICATION\_REQUEST的消息头必须将其<OpCode>设置为OC\_VERIFY\_CHALLENGE，而<ResponseCode>设置为0。

验证请求的消息体定义如下:

**<**VERIFICATION\_REQUEST> ::=<KeyHandle>

<KeyIndex>

<Challenge>

<ChallengeResponse>

这里

<KeyHandle>

一个UTF8字符串，它引用持有管理员密钥的handle。

<KeyIndex>

一个4字节无符号整数，是持有管理员密钥的handle值的索引。

<Challenge>

服务器挑战的消息体，如3.5.1节所述。

**<**ChallengeResponse**>**

为了服务器的<Challenge>，来自客户机的<ChallengeRespons>响应，如3.5.2节中定义的那样。

任何挑战-响应 验证-请求必须在消息头中设置其CT位。这是为了确保验证服务器将在下一节中指定的验证-响应上签名。

### 3.5.4挑战-响应 验证-响应

验证响应告诉请求Handle服务器<ChallengeRespons>是否与验证请求中的<Challenge>匹配。

验证响应的消息头必须将其<ResponseCode>设置为RC\_SUCCESS，无论<ChallengeRespons>是否与<Challenge>匹配。<OpFlag>字段中的RD标志也应该设置(为1)，因为<RequestDigist>将在消息正文中是强制性的。

验证响应的消息体定义如下:

<挑战-响应 验证-响应>::= <RequestDigest>

< VerificationResult **>**

这里

**<**RequestDigest**>**

包含验证请求的消息摘要。

**<**VerificationResult**>**

如果<ChallengeRespons>与<Challenge>匹配，则将其设置为1的字节组。否则它必须设置为0。

验证服务器如果不能执行验证，可能会返回一个错误，并将<ResponseCode>设置为RC\_AUTHEN\_FAILED(例如，<KeyHandle>不存在，或者<KeyHandle>和<KeyIndex>指的是无效的handle值)。当发生这种情况时，执行客户机身份验证的服务器应该将相同的错误消息转发回客户机。

## 3.6。Handle管理

Handle系统协议支持一组handle管理功能，包括添加、删除和修改handle或handle值。在完成任何管理请求之前，服务器必须对客户机进行身份验证，使其成为被授权进行管理操作的handle管理员。handle管理只能由主handle服务器执行。

### 3.6.1添加handle值

客户机通过向负责的Handle服务器发送ADD\_VALUE请求来向现有handle添加值。ADD\_VALUE请求的消息头必须将其<OpCode>设置为OC\_ADD\_VALUE。

ADD\_VALUE请求的消息体编码如下:

**<**ADD\_VALUE请求的消息体>::= <Handle>

<ValueList>

这里

<Handle>

指定handle的UTF8字符串。

**<**ValueList**>**

一个4字节无符号整数，后跟一个handle值列表。该整数指示列表中handle值的数目。

接收ADD\_VALUE请求的服务器必须首先使用ADD\_VALUE特权对客户机进行身份验证。身份验证成功后，服务器将继续将<ValueList>中的每个值添加到<Handle>。如果成功，服务器将向客户机返回一条RC\_SUCCESS消息。

每个ADD\_VALUE请求必须作为一个事务执行。如果在<ValueList>中添加任何值引起错误，则必须回滚整个操作。对于任何失败的ADD\_VALUE请求，<ValueList>中的值都不应该添加到<Handle>。服务器还必须向客户机发送解释错误的响应。例如，如果<ValueList>中的值具有与现有handle值之一相同的索引，则服务器将返回一条错误消息，该错误消息将<ResponseCode>设置为RC\_VALUE\_ALREADY\_EXISTS。

ADD\_VALUE请求也可以用来添加handle管理员。如果ADD\_VALUE请求中的<ValueList>包含任何HS\_ADMIN值，就会发生这种情况。在完成这些请求之前，服务器必须使用ADD\_ADMIN特权对客户机进行身份验证。

如果请求handle不存在，ADD\_VALUE请求将导致错误。发生这种情况时，服务器将返回一条错误消息，并将<ResponseCode>设置为RC\_HANDLE\_NOT\_EXIST。

### 3.6.2删除handle值

客户机通过向负责的Handle服务器发送REMOVE\_VALUE请求来删除现有的handle值。REMOVE\_VALUE请求的消息头必须将其<OpCode>设置为OC\_REMOVE\_VALUE。

任何REMOVE\_VALUE请求的消息体编码如下:

**<**REMOVE\_VALUE请求的消息体>::=<Handle>

<IndexList>

这里

<Handle>

一个UTF8字符串，指定需要删除其值的handle。

**<**IndexList**>**

一个4字节无符号整数，后跟一组handle值索引。每个索引引用要从<handle>中删除的handle值。该整数指定列表中的索引数量。每个索引也被编码为4字节无符号整数。

接收REMOVE\_VALUE请求的服务器必须首先使用REMOVE VALUE特权对客户机进行身份验证。身份验证成功后，服务器将继续从**<**handle**>**中删除<IndexList>中指定的handle值。如果成功，服务器将向客户机返回一条RC\_SUCCESS消息。

每个REMOVE\_VALUE请求必须作为一个事务执行。如果删除<IndexList>中指定的任何值会引发错误，则必须回滚整个操作。对于任何失败的REMOVE\_VALUE请求，<IndexList>中引用的值都不应该从<Handle>中删除。服务器还必须向客户机发送解释错误的响应。例如，尝试删除任何没有PUB\_WRITE或ADMIN\_WRITE权限的handle值都会导致RC\_ACCESS\_DENIED错误。注意，要求删除不存在handle值的REMOVE\_VALUE请求不会被视为错误。

REMOVE\_VALUE请求也可以用来删除handle管理员。如果<IndexList>中的任何索引引用HS\_ADMIN值，就会发生这种情况。在完成这些请求之前，服务器必须使用REMOVE\_ADMIN特权对客户机进行身份验证。

### 3.6.3修改handle值

客户机可以通过向负责的Handle服务器发送MODIFY\_VALUE请求来修改现有的handle值。MODIFY\_VALUE请求的消息头必须将其<OpCode>设置为OC\_MODIFY\_VALUE。

任何MODIFY\_VALUE请求的消息体定义如下:

**<**MODIFY\_VALUE响应的消息体> ::= <Handle>

<ValueList>

这里

<Handle>

一个UTF8字符串，指定需要修改其值的handle。

**<**ValueList**>**

一个4字节无符号整数，后跟一个handle值列表。该整数指定列表中handle值的数目。<ValueList>中的每个值指定一个handle值，该handle值将用相同的索引替换现有的handle值。

接收到MODIFY\_VALUE请求的服务器必须首先使用MODIFY\_VALUE特权对客户机进行身份验证。成功验证之后，服务器将继续替换<ValueList>中列出的那些handle值，前提是每个handle值都具有PUB\_WRITE或ADMIN\_WRITE权限。如果成功，服务器必须使用RC\_SUCCESS消息通知客户机。

每个MODIFY\_VALUE请求必须作为一个事务执行。如果替换<ValueList>中列出的任何值引起错误，则必须回滚整个操作。对于任何失败的MODIFY\_VALUE请求，都不应该替换<ValueList>中的值。服务器还必须向客户机返回解释错误的响应。例如，如果MODIFY\_VALUE请求删除一个既没有PUB\_WRITE权限也没有ADMIN\_WRITE权限的handle值，服务器必须返回一条错误消息，消息中<ResponseCode>设置为 RC\_ACCESS\_DENIED。任何替换不存在handle值的MODIFY\_VALUE请求也将被视为一个错误。在这种情况下，服务器将返回一条错误消息，将<ResponseCode>设置为RC\_VALUE\_NOT\_FOUND。

MODIFY\_VALUE请求也可以用来更新handle管理员。如果<ValueList>中的值和要替换的值都是HS\_ADMIN值，就会发生这种情况。在完成这样的请求之前，服务器必须使用MODIFY\_ADMIN特权对客户机进行身份验证。将非HS\_ADMIN值替换为HS\_ADMIN值是错误的。在这种情况下，服务器将返回一条错误消息，消息中<ResponseCode>设置为 RC\_VALUE\_INVALID。

### 3.6.4创建handle

客户机可以通过向负责的Handle服务器发送CREATE\_HANDLE请求来创建新的handle。任何CREATE\_HANDLE请求的消息头必须将其<OpCode>设置为OC\_CREATE\_HANDLE。

任何CREATE\_HANDLE请求的消息体定义如下:

**<**CREATE\_HANDLE响应的消息体>::= <Handle>

<ValueList>

这里

<Handle>

指定handle的UTF8字符串。

**<**ValueList**>**

一个4字节无符号整数，后跟一个handle值列表。该整数指示列表中handle值的数目。<ValueList>至少应该包含一个定义handle管理员的HS\_ADMIN值。

只有具有CREATE\_HANDLE特权的命名权威管理员才能在命名权威下创建新handle。接收CREATE\_HANDLE请求的服务器必须验证客户机是相应的命名权威handle的管理员，并确保管理员被授权在命名权威下创建handle。这与ADD\_VALUE请求不同，在ADD\_VALUE请求中，服务器将客户机身份验证为handle的管理员。身份验证成功后，服务器将继续创建新的handle，并将<ValueList>中的每个值添加到新的**<**handle**>**。如果成功，服务器将向客户机返回一条RC\_SUCCESS消息。

每个CREATE\_HANDLE请求必须作为一个事务执行。如果CREATE\_HANDLE进程的任何部分失败，则可以回滚整个操作。例如，如果服务器未能将<ValueList>中的值添加到新handle，则必须返回错误消息，而不创建新handle。任何要求创建已经存在的handle的CREATE\_HANDLE请求都将被视为错误。在这种情况下，服务器将返回一条错误消息，消息中将<ResponseCode>设置为RC\_HANDLE\_ALREADY\_EXIST。

CREATE\_HANDLE请求还可用于创建命名权威。命名权威是作为GHR的命名权威handle创建的。在创建新的命名权威handle之前，服务器必须对客户机进行身份验证，使其成为父命名权威的管理员。只有具有CREATE\_NA特权的管理员才允许创建任何子命名权威。根级命名权威可以由根handle“0.NA/0.NA”的管理员创建。

### 3.6.5删除handle

客户机通过向负责的Handle服务器发送DELETE\_HANDLE请求来删除现有handle。DELETE\_HANDLE请求的消息头必须将其<OpCode>设置为OC\_DELETE\_HANDLE。

任何DELETE\_HANDLE请求的消息体定义如下:

<删除handle请求的消息体>::= <Handle>

这里

<Handle>

指定handle的UTF8字符串。

接收DELETE\_HANDLE请求的服务器必须首先使用DELETE\_HANDLE特权对客户机进行身份验证。成功验证之后，服务器将删除handle和分配给handle的任意handle值。如果成功，服务器将向客户机返回一条RC\_SUCCESS消息。

每个DELETE\_HANDLE请求必须作为事务执行。如果DELETE\_HANDLE进程的任何部分失败，则必须回滚整个操作。例如，如果服务器未能删除分配给handle的任何handle值(在删除handle之前)，则必须返回错误消息，但不能删除handle。如果handle包含一个既没有PUB\_WRITE权限也没有ADMIN\_WRITE权限的值，就可能发生这种情况。在这种情况下，服务器将返回一条错误消息，并将<ResponseCode>设置为RC\_PERMISSION\_DENIED。要求删除不存在handle的DELETE\_HANDLE请求也将被视为错误。服务器将返回一条错误消息，<ResponseCode>设置为RC\_HANDLE\_NOT\_EXIST。

DELETE\_HANDLE请求还可用于删除命名权威。这是通过删除GHR上相应的命名权威handle来实现的。在删除命名权威handle之前，服务器必须对客户机进行身份验证，使其成为命名权威handle的管理员。只有具有DELETE\_NA特权的管理员才允许删除命名权威。根级命名权威可以由根handle“0.NA/0.NA”的管理员删除。

## 3.7命名权威(NA)管理

Handle系统通过命名权威handle来管理命名权威。命名权威handle由GHR管理。客户机可以通过更改分配给相应命名权威handle的HS\_SITE值来更改任何命名权威的服务信息。创建或删除命名权威是通过创建或删除相应的命名权威handle来完成的。根级命名权威可以由根handle“0.NA/0.NA”的管理员创建或删除。非根级命名权威可以由其父命名权威的管理员创建。

例如，命名权威handle“0.NA/10”的管理员，通过向GHR发送一个CREATE\_HANDLE请求创建命名权威handle“0.NA/10.1000”，可以创建命名权威“10.1000”。在完成请求之前，GHR上的服务器必须对客户机进行身份验证，使其成为父命名权威的管理员，即命名权威handle“0.NA/10”的管理员。服务器还必须确保管理员具有NA\_CREATE特权。

Handle协议还允许客户机在命名权威下列出handle或子命名权威。这些操作的细节将在下面的部分中描述。

### 3.7.1列出命名权威下的handle

客户机向Handle服务器发送LIST\_HANDLE请求，以获得命名权威下的handle列表。LIST\_HANDLE请求的消息头必须将其<OpCode>设置为OC\_LIST\_HANDLE。

任何LIST\_HANDLE请求的消息体定义如下:

**<**LIST\_HANDLE请求的消息体> ::= <NA\_Handle>

这里

**<**NA\_Handle**>**

指定命名权威handle的UTF8字符串。

要获得完整的handle列表，必须将请求发送到负责Handle服务的服务站点中列出的每个Handle服务器。服务站点中的每个服务器将在命名权威下返回自己的handle列表。一个成功的LIST\_HANDLE响应(来自每个Handle服务器)的消息体定义如下:

<LIST\_HANDLE响应的消息体> ::= <Num\_Handles>

<HandleList>

这里

**<**Num\_Handles**>**

在命名权威下的handle数(由Handle服务器管理)。

**<**HandleList**>**

UTF8字符串的列表，每个字符串在命名权威下标识一个handle。

LIST\_HANDLE请求可能会降低整个系统的性能。Handle服务(或其服务站点)可以选择是否支持此类请求。如果不支持LIST\_HANDLE，服务器将返回RC\_OPERATION\_DENIED消息。在完成请求之前，接收LIST\_HANDLE请求的服务器应该使用LIST\_HANDLE特权对客户机进行身份验证，使其作为命名权威的管理员。

### 3.7.2 在命名权威下列出子命名权威

客户机发送LIST\_NA请求到Handle服务器，以获得命名权威下的子命名权威列表。LIST\_NA请求的消息头必须将其<OpCode>设置为OC\_LIST\_NA。

任何LIST\_NA请求的消息体定义如下:

**<**LIST\_HANDLE请求的消息体> ::= <NA\_Handle>

这里

**<**NA\_Handle**>**

指定命名权威handle的UTF8字符串。

要获得完整的子命名权威列表，必须将请求发送到GHR任何一个服务站点中列出的每个Handle服务器。服务站点中的每个服务器将在给定的命名权威下返回自己的子命名权威handle列表。一个成功的LIST\_NA响应(来自每个Handle服务器)的消息体定义如下:

**<**LIST\_HANDLE响应的消息体> ::=**<**Num\_Handles**>**

<HandleList>

这里

**<**Num\_Handles**>**

在命名权威下的handle数(由Handle服务器管理)。

**<**HandleList**>**

UTF8字符串的列表，每个字符串标识用户指定的子命名权威。

LIST\_NA请求必须发送到管理全部命名权威handle的GHR下的服务器。LIST\_NA请求可能会降低整个系统的性能，特别是GHS。GHR下的服务器(或服务站点)可以选择是否支持此类请求。如果不支持LIST\_NA，服务器将返回RC\_OPERATION\_DENIED消息。在完成请求之前，接收LIST\_HANDLE请求的服务器应该使用LIST\_NA特权对客户机进行身份验证。

## 3.8 会话和会话管理

会话用于允许在多个协议操作之间共享身份验证信息或网络资源。例如，命名权威管理员可以通过会话设置对自己进行一次身份验证，然后在会话下注册多个handle。

客户机可以要求服务器建立一个会话密钥，并将其用于后续请求。会话密钥是客户机和服务器共享的密钥。它可用于对会话下交换的任何消息进行身份验证或加密。如果会话内交换的每个消息都使用会话密钥加密，则会话将被加密。

可以根据来自客户机的显式OC\_SESSION\_SETUP请求建立会话。当期望多个消息交换来满足请求时，服务器也可以自动设置会话。例如，如果服务器接收到需要客户机身份验证的CREATE\_HANDLE请求，它将自动建立一个会话。

每个会话都由出现在消息头中的非零会话ID标识。服务器负责为每个未完成的会话生成惟一的会话ID。每个会话可能有一组与之关联的状态信息。状态信息可以包括会话密钥和从客户机身份验证获得的信息，以及任何通信选项。服务器和客户机负责保持状态信息的同步，直到会话终止。

可以使用来自客户机的OC\_SESSION\_TERMINATE请求终止会话。服务器还可以终止已闲置了大量时间的会话。

### 3.8.1会话设置请求

客户机使用带有SESSION\_SETUP的请求与一个Handle服务器建立会话。SESSION\_SETUP请求还可用于更新与现有会话关联的任何状态信息。SESSION\_SETUP请求的消息头必须将其<OpCode>设置为OC\_SESSION\_SETUP，并将<ResponseCode>设置为0。

任何SESSION\_SETUP请求的消息体定义如下:

<SESSION\_SETUP请求消息体>::= <SessionAttributes>

这里

**<**SessionAttributes**>**

一个4字节无符号整数，后跟一个会话属性列表。整数表示列表中会话属性的数量。可能的会话属性包括<HS\_SESSION\_IDENTITY>、<HS\_SESSION\_TIMEOUT>和<HS\_SESSION\_KEY\_EXCHANGE>。每个属性定义如下

**<**HS\_SESSION\_IDENTITY**>**:: = <Key>

<Handle>

**<**ValueIndex**>**

这里

<Key>

一个UTF-8字符串常量“HS\_SESSION\_IDENTITY”。

<Handle>

**<**ValueIndex**>**

一个UTF-8字符串，后跟一个4字节无符号整数，指定用于客户机身份验证的handle和handle值。它必须引用包含客户机公钥的handle值。服务器使用公钥对客户机进行身份验证。

<HS\_SESSION\_KEY\_EXCHANGE>::= <Key>

<KeyExchangeData>

这里

<Key>

UTF-8字符串常量“HS\_SESSION\_KEY\_EXCHANGE”。

**<**KeyExchangeData**>**

这些元组之一:<ClientCipher KeyExchange>、<ClientCipher KeyExchange>、<HdlCipher KeyExchange>或<ServerCipher KeyExchange**>**。

每个元组的定义如下:

<ClientCipher KeyExchange>::= <Key>

**<**PubKey**>**

这里

<Key>

一个UTF-8字符串常量“CLIENT\_CIPHER”。

**<**PubKey**>**

由客户机提供并由服务器用于加密会话密钥的公钥。

<HdlCipher KeyExchange>::= <Key>

**<**ExchangeKeyHdl**>**

**<**ExchangeKeyIndex**>**

这里

<Key>

一个UTF-8字符串常量“HDL\_CIPHER”。

**<**ExchangeKeyHdl**>**

**<**ExchangeKeyIndex**>**

一个UTF-8字符串，后跟一个4字节无符号整数。<ExchangeKeyHdl>和<ExchangeKeyIndex>是用于会话密钥交换的handle值。handle值必须包含客户机的公钥。在将会话密钥发送给客户机之前，服务器将使用公钥对会话密钥进行加密。

<ServerCipher KeyExchange>::= <Key>

这里

<Key>

一个UTF-8字符串常量“SERVER\_CIPHER”。这告诉服务器客户机将负责生成会话密钥。服务器必须在响应消息中提供其公钥，并将<ResponseCode>设置为RC\_SESSION\_EXCHANGEKEY。客户机可以使用服务器的公钥加密会话密钥，并通过后续的SESSION\_EXCHANGEKEY请求将其发送给服务器。

<DiffieHellman KeyExchange>::= <Key> <DHParams>

这里

<Key>

一个UTF-8字符串常量"DIFFIE\_HELLMAN"

**<**DHParams**>**

在Diffie-Hellman算法中用作输入的值。它由三个变长整数组成。每个大整数都是以一个4字节无符号整数后跟一个字节字符串的形式编码的。字节字符串本身包含一个大整数。4字节无符号整数指定字节字符串的字节数。

**<**HS\_SESSION\_TIMEOUT**>**:: = <Key>

<TimeOut>

这里

<Key>

UTF-8字符串常量“HS\_SESSION\_TIMEOUT”。

<TimeOut>

一个4字节无符号整数，指定会话所需的持续时间(以秒为单位)。

注意，如果在<SessionAttribute>字段中多次列出相同的会话属性，则应该将其视为错误。发生这种情况时，服务器应该返回一条错误消息，将<ResponseCode>设置为RC\_PROTOCOL\_ERROR。

SESSION\_SETUP\_REQUEST可用于更改任何已建立的会话的会话属性。如果<SessionId>非零并且匹配一个已建立的会话，就会发生这种情况。服务器必须注意防止任何未经授权的请求更改会话属性。例如，一个加密的会话只能由一个SESSION\_SETUP\_REQUEST将其更改为一个未加密的会话，该SESSION\_SETUP\_REQUEST的消息凭据中包含一个适当的MAC。

### 3.8.2会话设置响应

SESSION\_SETUP响应的消息头必须将其<OpCode>设置为OC\_SESSION\_SETUP。SESSION\_SETUP响应的<ResponseCode>根据SESSION\_SETUP请求而变化。

如果SESSION\_SETUP请求成功，并且服务器不希望客户机返回会话密钥，则必须将其设置为RC\_SUCCESS。

除非请求<HS\_SESSION\_KEY\_EXCHANGE>，否则SESSION\_SETUP响应的消息体是空的。在这种情况下，SESSION\_SETUP响应的消息体可能包含来自服务器的加密会话密钥，或者服务器的公钥，用于会话密钥交换。确切的格式取决于SESSION\_SETUP请求中<HS\_SESSION\_KEY\_EXCHANGE>的内容。如果在SESSION\_SETUP请求中给出了<ClientCipher KeyExchange>或<HdlCipher KeyExchange>，则SESSION\_SETUP响应的消息体将包含来自服务器的加密会话密钥，定义如下:

**<**SESSION\_SETUP响应的消息体>::= <RequestDigest>

<EncryptedSessionKey>

[ <EncryptionAlgorithm> ]

这里

**<**RequestDigest**>**

SESSION\_SETUP请求的消息摘要在2.2.3节中指定。

**<**EncryptedSessionKey**>**

会话密钥使用SESSION\_SETUP请求中提供的公钥进行加密。会话密钥是从服务器随机生成的字节字符串。如果SESSION\_SETUP请求中的<KeyExchangeData>从客户机提供了公钥，服务器将只返回<EncryptedSessionKey>。

**<**EncryptionAlgorithm**>**

(可选)标识会话密钥使用的加密算法的UTF-8字符串。

如果在SESSION\_SETUP请求中给出了<ServerCipher KeyExchange>，则服务器必须在SESSION\_SETUP响应中提供其公钥。客户机可以在后续的SESSION\_EXCHANGEKEY请求(定义如下)中使用公钥进行会话密钥交换。在这种情况下，SESSION\_SETUP响应的消息头必须将其<ResponseCode>设置为RC\_SESSION\_EXCHANGEKEY。SESSION\_SETUP响应的消息体必须包含服务器的公钥，定义如下:

**<**SESSION\_SETUP响应的消息体>::= <RequestDigest>

<会话密钥交换的公钥>

这里

**<**RequestDigest**>**

在2.2.3节中指定的SESSION\_SETUP请求的消息摘要。

<会话密钥交换的公钥>

来自服务器的公钥，用于会话密钥交换。它以与HS\_SITE值中的<PublicKey>记录相同的格式编码(参见[2]中的3.2.2节)。

### 3.8.3会话密钥交换

如果SESSION\_SETUP响应的<ResponseCode>是RC\_SESSION\_EXCHANGEKEY，客户机负责生成会话密钥并将其发送到服务器。在这种情况下，客户机可以生成一个会话密钥，使用服务器在SESSION\_SETUP响应中提供的公钥加密它，然后在SESSION\_EXCHANGEKEY请求中将加密的会话密钥发送到服务器。

SESSION\_EXCHANGEKEY请求的消息头必须将其<OpCode>设置为OC\_SESSION\_EXCHANGEKEY，将其<ResponseCode>设置为0。SESSION\_EXCHANGEKEY请求的消息体定义如下:

**<**OC\_SESSION\_EXCHANGEKEY的消息体> ::= <EncryptedSessionKey>

[<EncryptionAlgorithm>]

这里

**<**EncryptedSessionKey**>**

使用SESSION\_SETUP响应中提供的公钥加密会话密钥。会话密钥是客户机随机生成的字节字符串。

**<**EncryptionAlgorithm**>**

(可选)标识会话密钥使用的加密算法的UTF-8字符串。

在会话密钥交换期间，接收交换密钥或会话密钥的服务器负责确保密钥满足其本地策略中定义的安全需求。如果服务器认为密钥是脆弱的，那么它必须向客户机返回一条错误消息，并将<ResponseCode>设置为RC\_SESSION\_KEY\_INVALID。

### 3.8.4会话终止

客户机可以使用SESSION\_TERMINATE请求终止会话。SESSION\_TERMINATE请求的消息头必须将其<OpCode>设置为OC\_SESSION\_TERMINATE，并将其<ResponseCode>设置为0。任何SESSION\_TERMINATE请求的消息主体必须为空。

服务器必须在会话终止后向客户机发送SESSION\_TERMINATE响应。服务器应该只在处理完在会话终止请求之前提交的所有请求(会话下的请求)之后才终止会话。

SESSION\_TERMINATE响应的消息头必须将其<OpCode>设置为OC\_SESSION\_TERMINATE。成功的SESSION\_TERMINATE响应必须将其<ResponseCode>设置为RC\_SUCCESS，并设置一个空的消息体。

# 实施细则

## 服务器实现

Handle服务器的最佳结构取决于主机操作系统。本节只讨论大多数Handle服务器常见的实现注意事项。

一个好的服务器实现应该允许简单的配置或调优。建议的配置项列表包括服务器的网络接口(s)(如IP地址,端口号,等等),允许的并发进程/线程的数量,任何TCP连接和/或验证过程的超时时间间隔,UDP连接的重试策略,是否支持递归服务策略,ASCII字符的大小写敏感性,和事务日志不同级别的等。

所有Handle服务器实现必须支持“Handle系统名称空间和服务定义”[2]中定义的所有handle数据类型。它们还应该能够存储任何应用程序定义的数据类型的handle值。

Handle服务器必须支持多个并发活动，不管它们是作为主机操作系统中的独立进程或线程实现的，还是在单个名称服务器程序中实现的。Handle服务器在等待TCP数据或其他查询活动时不应该阻塞UDP请求的服务。类似地，如果不并行处理此类请求，Handle服务器不应尝试提供递归服务，尽管它可以选择串行化来自单个客户机的请求，或者将来自相同客户机的相同请求视为重复请求。

## 4.2客户机实现

客户机应该准备接收任何数据类型的handle值。客户机可以选择实现回调接口，以允许添加新的模块或插件来支持任何应用程序定义的数据类型。

遵循服务引用或处理别名的客户机必须避免陷入无限循环。对于相同的请求，他们不应该使用相同的目标条目重复联系相同的服务器。客户机可以选择使用一个计数器，该计数器在每次跟随服务引用或处理别名时递增。应该为计数器设置一个可配置的上限，以控制客户机跟随的服务引用或handle别名的级别。

提供一些缓存的客户机可以比不提供缓存的客户机获得更好的性能。客户机实现应该始终考虑缓存与命名权威关联的服务信息。这将减少相同命名权威下后续handle请求的往返次数。

# 安全注意事项

在“Handle系统概述”[1]中讨论了整个Handle系统安全注意事项;这种讨论同样适用于本文件。关于Handle系统数据模型和服务模型的安全性考虑在“Handle系统名称空间和服务定义”[2]中进行了讨论。

为了提高效率，Handle协议包含一个简单的挑战-响应身份验证协议，用于基本的客户机身份验证。Handle服务器可以根据需要自由地提供额外的身份验证机制(如SASL)。这方面的细节将在另一份文件中讨论。

Handle协议下的数据完整性是通过服务器的数字签名实现的。必须注意保护服务器的私钥不受任何模拟攻击。对服务器的公钥对的任何更改都必须向GHR注册(就服务信息而言)。

# 感谢

这项工作源自Handle系统实现的早期版本。1995年，Robert Kahn和Robert Wilensky[22]在一篇论文中描述了整个数字对象体系结构，包括Handle系统。发展在CNRI继续作为计算机科学技术报告(CSTR)项目的一部分，由美国国防部高级项目局(DARPA)资助，资助号为MDA-972-92-J-1029和MDA-972-99-1-0018。设计思想基于Handle系统开发团队中的讨论，包括David Ely、Charles Orth、Allison Yu、Sean Reilly、Jane Euler、Catherine Rey、Stephanie Nguyen、Jason Petrone和Helen She。我们感谢他们对这项工作的贡献。

作者还感谢Russ Housley (housley@vigilsec.com)、Ted Hardie (hardie@qualcomm.com)和Mark Baugher (mbaugher@cisco.com)的广泛评论，以及来自IETF/IRTF社区其他成员的建议。

# 参考文献

[1] Sun, S. and L. Lannom, "Handle System Overview", RFC 3650,

November 2003.

[2] Sun, S., Reilly, S. and L. Lannom, "Handle System Namespace and

Service Definition", RFC 3651, November 2003.

[3] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC

2279, January 1998.

[4] A. Freier, P. Karlton, P. Kocher "The SSL Protocol Version 3.0"

[5] RSA Laboratories, "Public-Key Cryptography Standard PKCS#7"

http://www.rsasecurity.com/rsalabs/pkcs/

[6] U.S. Federal Information Processing Standard: Digital Signature

Standard.

[7] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms",

RFC 3370, August 2002.

[8] Braden, R., "FTP Data Compression", RFC 468, March 1973.

[9] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April

1992.

[10] NIST, FIPS PUB 180-1: Secure Hash Standard, April 1995.

[11] D. Cohen, "On Holy Wars and a Plea for Peace", Internet

Experiment, Note IEN 137, 1 April 1980.

[12] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC

3124, June 2001.

[13] R. Kahn, R. Wilensky, "A Framework for Distributed Digital

Object Services, May 1995, http://www.cnri.reston.va.us/k-w.html

[14] Polk, W., Housley, R. and L. Bassham, "Algorithms and

Identifiers for the Internet X.509 Public Key Infrastructure

Certificate and Certificate Revocation List (CRL) Profile", RFC

3279, April 2002.

[15] Housley, R., Polk, W., Ford, W. and D. Solo, "Internet X.509

Public Key Infrastructure Certificate and Certificate Revocation

List (CRL) Profile", RFC 3280, April 2002.

[16] M. Bellare and P. Rogaway. The Exact Security of Digital

Signatures - How to Sign with RSA and Rabin. In Advances in

Cryptology-Eurocrypt ’96, pp.399-416, Springer-Verlag, 1996.

[17] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing

for Message Authentication", RFC 2104, February 1997.

[18] R. Kahn, R. Wilensky, "A Framework for Distributed Digital

Object Services, May 1995, http://www.cnri.reston.va.us/k-w.html

# 作者的地址

Sam X. Sun

Corporation for National Research Initiatives (CNRI)

1895 Preston White Dr., Suite 100

Reston, VA 20191

Phone: 703-262-5316

EMail: ssun@cnri.reston.va.us

Sean Reilly

Corporation for National Research Initiatives (CNRI)

1895 Preston White Dr., Suite 100

Reston, VA 20191

Phone: 703-620-8990

EMail: sreilly@cnri.reston.va.us

Larry Lannom

Corporation for National Research Initiatives (CNRI)

1895 Preston White Dr., Suite 100

Reston, VA 20191

Phone: 703-262-5307

EMail: llannom@cnri.reston.va.us

Jason Petrone

Corporation for National Research Initiatives (CNRI)

1895 Preston White Dr., Suite 100

Reston, VA 20191

Phone: 703-262-5340

EMail: jpetrone@cnri.reston.va.us

# 完整的版权声明

版权所有(C)互联网协会(2003)。 保留所有权利。

这个文档和翻译的可能被复制和家具,和衍生著作,评论或者解释或者协助其实现可能准备,复制,发布和分发,在全部或部分,没有任何形式的限制,前提是上面的版权声明和本段包括在所有此类副本和衍生著作。然而,这个文档本身可能不会以任何方式修改,如通过删除版权通知或引用互联网协会或其他网络组织,除所需的目的为版权开发互联网标准在这种情况下,程序中定义的过程必须遵循网络标准,或根据需要翻译成除英语之外的其他语言。

上述所授予的有限权限是永久的，互联网协会及其继任者或受让人不会撤销上述权限。

这个文档,此处包含的信息是“是”的基础上提供的互联网协会和互联网工程任务组并不保证,明示或暗示,包括但不限于任何保证,使用文中的信息不会侵犯任何权利或任何隐含保证适销性或健身为特定目的。

感谢

RFC编辑器功能的资金目前由互联网协会提供。

译者：王伟兵