第五章 命名系统

Author： 中山大学 17数据科学与计算机学院 YSY

https://github.com/ysyisyourbrother

5.1 名称、标识符和地址

名称：在分布式系统中，名称是由位或字符组成的字符串，用来指向一个实体。

实体：实体可以是任何事物，比如主机、打印机、磁盘等。进程，邮箱，web页面等都是。

访问点：也称为地址

- 如果要对实体进行操作，就需要访问实体，因此需要一个访问点。**访问点是一个特殊类型的实体。它的名称就是地址。**
- 一个实体可以拥有多个访问点。
- 一个实体经过一定时间后可能改变其访问点。
- 一个实体很容易改变访问点，因此比较好的办法是让服务与名称相关，而名称与服务器地址是无关（**位置无关的**）

标识符：用来唯一标识实体的名称

1. 一个标识符最多引用一个实体
2. 每个实体最多由一个标识符引用
3. 一个标识符时钟引用同一个实体（标识符不会被重新使用）

用户友好名称：在很多操作系统中，地址和标识符仅使用计算机可读形式标识，另一种重要的名称类型是为用户使用而定制的，也称为**用户友好名称**

如何把名称和标识符解析为地址。原则上命名系统有一个名称到地址的绑定，其最简单形式是一个(name,address)对的表，但在分布式系统中一个集中式的表无法满足。

5.2 无层次命名

非结构化五层次名称的重要属性是：它不包含任何有关如何定位其相关实体的访问点的信息。（相当于要自己找目标的实体，而不是像5.3节中有路径可以找）

5.2.1.1 广播和多播

ARP协议来查找ip地址

5.2.1.2 转发指针

当实体A移动到B时，它将在后面留下一个指针，指向它在B中的新位置。因此一旦找到实体后，用户就能顺着转发指针形成的链来查找实体的当前地址。

缺点：

1. 如果不采取措施，链会非常的长，开销很大
2. 链中的所有位置都必须维护它们的转发指针
3. 链很脆弱，容易断开

因此需要让指针相对短一些，并且确保转发指针不容易被破坏

SSP链

每个转发指针以对的形式实现（（stub，scion）也就是（存根，后裔）对）。服务器存根要么包含实际对象的本地引用，要么包含该对象对远程客户存根的本地引用（要么指向对象，要么指向一个指向另外一个服务器存根的客户存根）

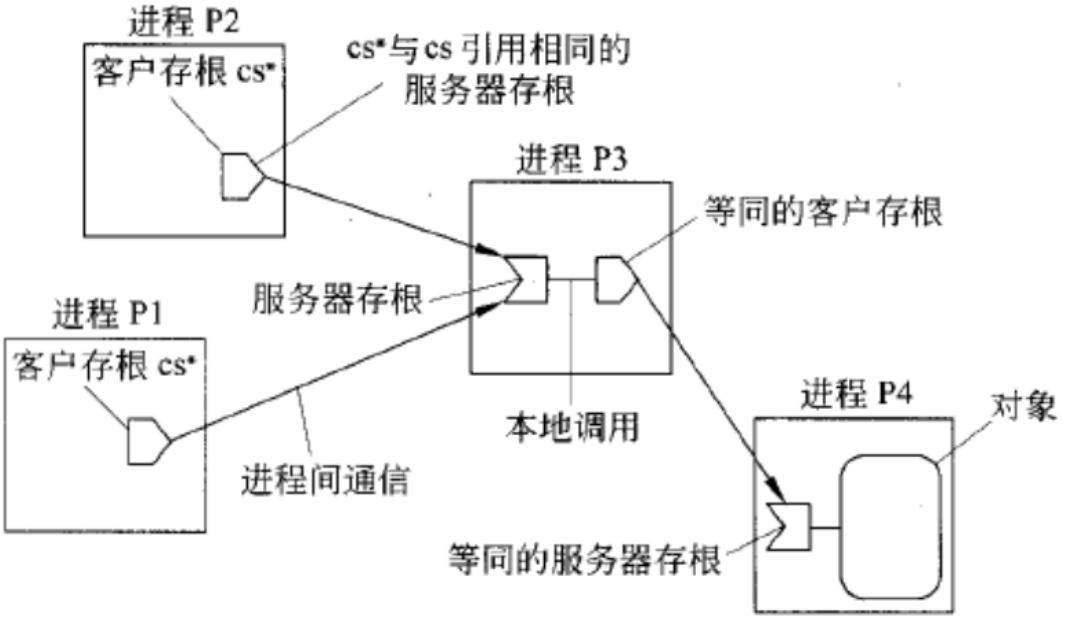


图 5.1　使用(客户存根,服务器存根)对的转发指针的原理

当对象从地址空间A移到地址空间B时，它会把一个客户的存根留在A中，并且在B中安装一个引用它的服务器存根。移动细节对客户来说是**完全透明**的，客户惟一能看到的对象就是一个客户存根。

为了简化（客户存根，服务器存根）对的链，对象调用会携带一个客户存根的标识，这是调用发起的地方。这个客户存根的标识由客户的传输层地址加上一个本地产生的、用于标识该存根的号码组成。当调用到达位于当前位置的对象以后，会向发起调用的客户存根发送回一个响应。**客户存根会把自己的对应服务器存根调整为对象当前位置中哪一个位置的服务器存根。**

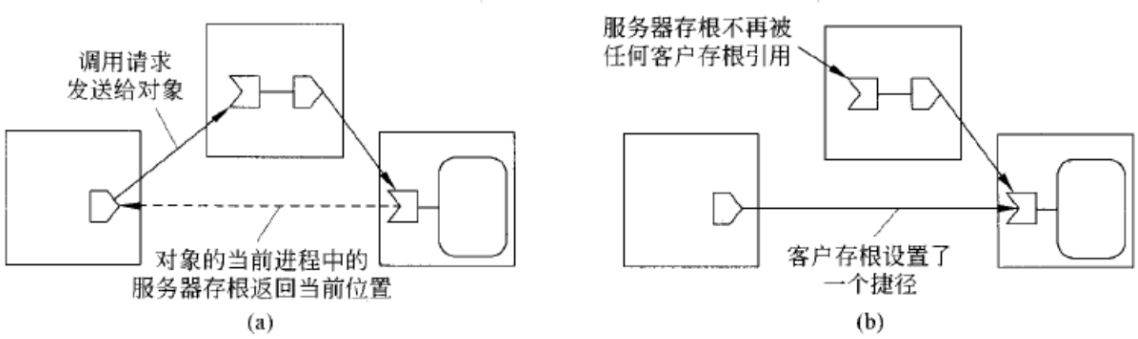


图 5.2　通过在客户存根中存储一个捷径来重定向转发指针

直接向起始客户发送响应或沿着转发指针的相反路线发送响应。前者更快。而后者可以调整中间存根和前者只能调整起始客户存根。

当服务器存根不再被引用就可以删除了。

5.2.2 基于宿主位置的方法：

在大型网络中支持移动实体定位是引入**宿主位置**的方法。

每个移动主机都是用一个固定的IP地址，所有与该IP地址的通信一开始都被转发到移动主机的**宿主代理**中。宿主代理位于局域网中，与包含在移动主机IP地址中的网络地址相对应。当一台移动主机转移到另外一个网络中的时候，它会请求一个用来的通信的临时地址，这种**转交地址**要在宿主代理中注册。

当宿主代理接受到发给主机的数据包后，会查找主机当前位置。如果主机在本地网络中，就转发数据包；如果不在，它就会把数据组成IP包，发送给转交地址。**同时告诉数据包发送者主机当前位置。**

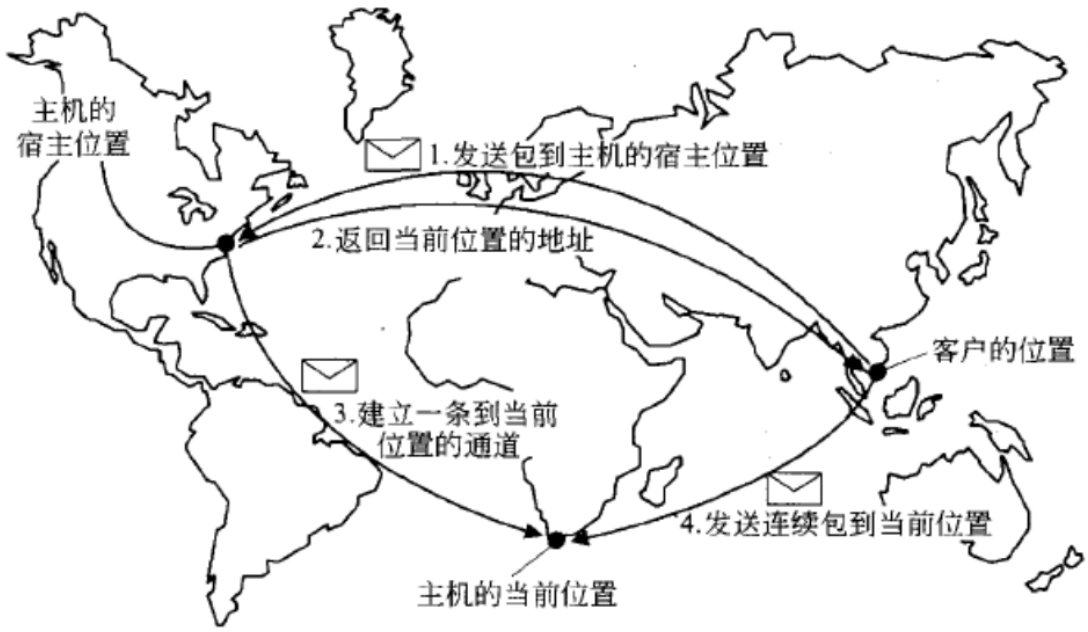


图 5.3　移动 IP 原理

缺点：

1. 宿主机需要伴随实体整个生命周期
2. 必须先和宿主通信，增加了通信延迟

3. 使用了固定的宿主位置，要保证宿主位置始终存在，否则无法与实体联系。

针对第二个问题的解决办法是，注册宿主的位置，让客户先查找宿主位置在通信，这样就允许了宿主进行位置移动。

5.2.3 分布式散列表DHT

含有键值k的实体位于含有最小标识符(键值)id>=k的节点内，该节点称为k的后继者，用succ(k)表示。**succ相当于解析结点真实的地址**

主要问题是如何高效的将键值k解析为succ(k)。一个方法是，让每个结点p都跟踪后继succ(p+1)和前继者pred(p)，**即收集前面和后面结点对应的键值。**只要结点p接受到一个将键值k解析的请求，除非pred(p)<k<=p（**代表这个k这个实体是在p结点内的**），否则将它转发到其中一个符合条件的相邻结点。如果是在结点p内，**那么p把自己的地址返回。**

教程参考： http://www.yeolar.com/note/2010/04/06/p2p-chord/

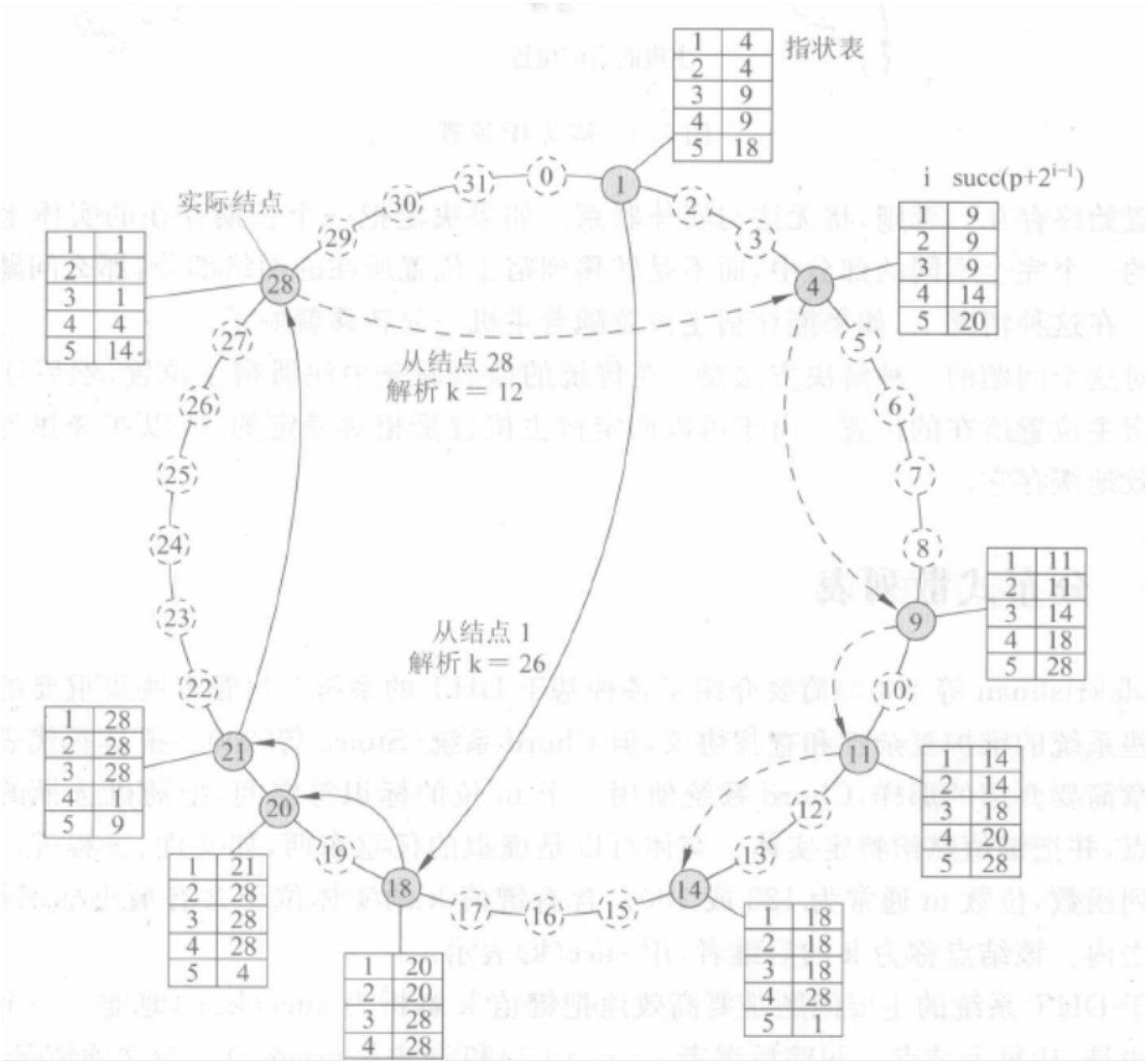
节点查找

在找键值k属于那个结点的时候，如果顺序查找每个结点会比较慢。因此，每个Chord结点维护一个最多有m个实体的**指状表**，如果用FT_p 表示：

也就是指状表中第i行保存了p后2^{i-1}结点的键值。

要查找键值k，结点p立即把该请求转发给在p的指状表中索引为j的结点q

为了解释这种查找，我们从结点 1 解析 k=26，如图 5.4 所示。首先，结点 1 在其指状表中查找 k=26，发现该值大于 $FT_1[5]$，表示该请求将转发给结点 18(=$FT_1[5]$)，而结点 18 将选择结点 20，因为 $FT_{18}[2]<k\leqslant FT_{18}[3]$。最后，该请求从结点 20 转发给结点 21，从结点 21 又转发给结点 28，该结点负责 k=26。此时，结点 28 的地址返回给结点 1，于是就完成了该键值的解析。同样，当请求结点 28 解析键值 k=12 时，该请求将按图 5.4 所示的虚线路由。可以证明，一个查找通常需要 O(log(N))步，其中 N 为系统中的结点数。



FT表举例说明：拿键值为28的结点来说，index=5时，(28+2^(5-1))%32=12，键值为12的地方的下一个结点是14号结点（虚线代表没有主机的结点），于是它可以通过index=5的对应的一个指针跳转，就不需要一个一个结点遍历。

指状表的意思是：比如拿key=20的节点为例，就是键值$20+2^{i-1}$的数据保存在哪个节点，儿22到27是不存在的节点，他们的数据都保存在key=28上。

succ(q+1)代表q的下一个结点，succ(q)=1，pred(q)等于q的上一个结点

当k=26到结点21的时候，查表发现下一个就是键值为28的，键值为28已经大于了26，因此，键值为28的结点存放数据k=26，因此跳到28，28将地址返回。

节点加入

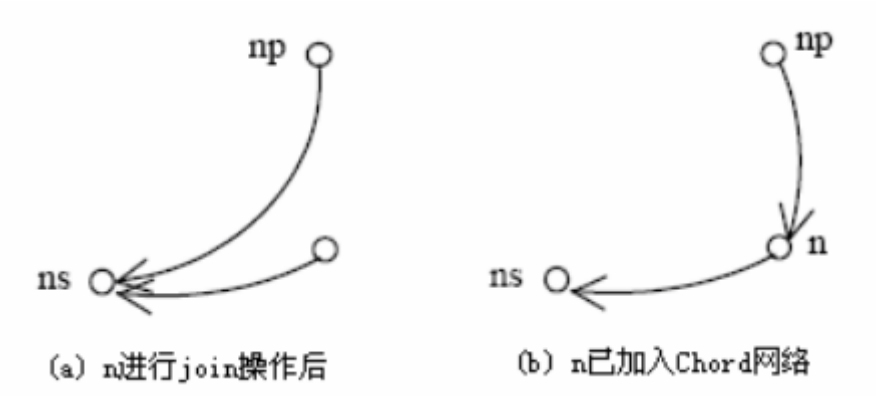
在大型分布式系统中，参与结点集可能总在变化。

1. 如果结点p要加入，则只需要与已有系统的任意结点联系，请求查找succ(p+1)，然后插入到环中。注意结点还要跟踪他的前任者。
2. 结点q必须保证它的下一个结点succ(q+1)是正确的。q可以定期请求返回pred(succ(q+1))，如果q=pred(succ(q+1))，说明一致。否则说明q的后继者更新了前继，即插入了结点p，且$q<p\leq succ(q+1)$，此时就需要将q的$FT_q[1]$调整称指向p，还有检查p是否将q作为其前继者，如果没有，继续调整$FT_q[1]$
3. 同时，每个结点q定期检查前继者是否为活动状态。如果前继者失效，q就将pred(q)记录为未知，

新节点的加入需要一个**称为向导的已知节点协助**，任何一个运行在 Chord 网络中的节点都可以充当这个角色。加入过程包括新节点本身的 Join 操作和被其他节点发现2个阶段 [5] ，如图 2 所示。假设 np 和 ns 是 Chord 网络中相邻两节点，n 为新节点，它加入网络后应该位于 np 和 ns 节点之间。

在第1阶段中，n 请求向导为它查找后继 (即 ns)，并初始化自身 Finger 表和后继表。按照 Finger 表定义，第1个需要查找后继的表项位置 join 操作完成后节点的情形如图 2(a) 所示。

此时只有 n 对自身属性进行了设置，其他节点并不知道新节点的加入，因此，第2阶段引入了 stabilize 操作，**所有节点定期检查其后继的前继，并向自己的直接后继发送 Notify 消息。**在图2的例子中：(1) 若 np 比 n 先向 ns 发送 Notify，此次 stabilize 并不改变网络状态；(2) 若 n 先向 ns 发送 Notify，ns 计算 n 的 ID 得知 n 比 np 更接近自己而认为有一个新节点加入， ns 由此把前继修改成 n，然后，np 在查看其后继(此时还是 ns)的前继(此时已改为 n)也发现了 n 的加入，np 把其后继修改成 n 并向 n 发送 Notify，把 n 的前继修改成 np，这样，np、n、ns 构成了完整的弦环，如图 4(b) 所示。当只有一个节点时，Chord 约定它的前继、后继都指向自身。



节点的失效

节点的失效是节点没有通知其他节点而突然离开网络，这通常由主机崩溃或 IP 网络断开等意外原因造成，此时失效节点的前继保存的后继信息变得不可用，从而造成 Chord 环的断裂

需要周期性对**节点的前继和后继**进行探测。如果节点 n 发现其后继已经失效，则从后继表中顺序查找第1个可用节点替换，并按照节点加入时的算法重建 Finger 表，然后通知前继 。 对前继节点失效的处理需要借助于 Notify 消息， 由于上述对后继失效的处理过程能够保证 Chord 环后继链的正确性，因此 np 通过在 stabilize 中向新后继 ns 发送 Notify，把 ns 的前继改成 np

得注意的是，其他节点也可能在 Finger 表项中保存有失效节点的记录，因此需要多次 stabilize，把失效信息扩散到 Chord 网络中

节点的退出

由于节点失效处理方法是稳定的，因此节点的退出可看作为失效而不采取其他附加措施。但基于效率的考虑，节点 n 退出时进行如下操作：

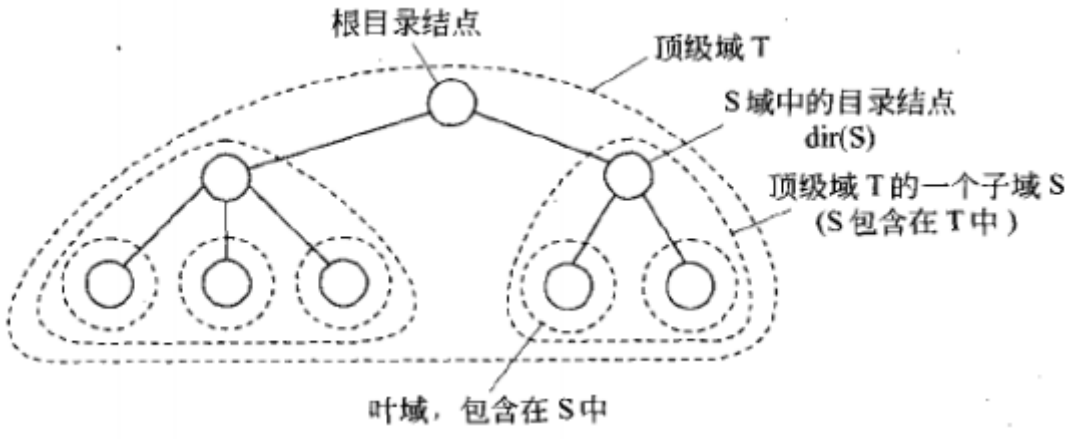
(1) 把 n 后继节点的前继改成 n 的前继；

(2) 把 n 前继节点的后继改成 n 的后继；

(3) 从 n 前继的后继表中删除 n。

5.2.4 分层方法

在分层方案中，网络被划分成了一组域。有一个覆盖整个网络的顶级域。每个域又可以进一步划分成多个更小的子域。最底层的域称为**叶域**，它通常与局域网对应或者与移动电话网络中的单元对应。



每个当前位于域D内的实体都由一条在**目录结点dir(D)**中的**位置记录**来表示。

1. 在叶域D的目录结点N中，实体E的位置记录包含该实体当前在域中的位置。
2. 含有D的更高一级的域D' 使用目录结点N'，它也拥有一个关于E的位置记录，这条位置记录只包含一个指向N' 的指针。
3. 中间结点包含了指向其孩子节点的指针，当且仅当根植于孩子节 点的子树存储有实体的地址
4. 根节点知道所有实体的地址

实体可以有多个地址，比如被复制了。如果**实体分别在叶域D1和D2拥有地址**，那么同时包含这两个域的域的目录结点将有两个指针，每个指针都指向一个包含地址的子域。

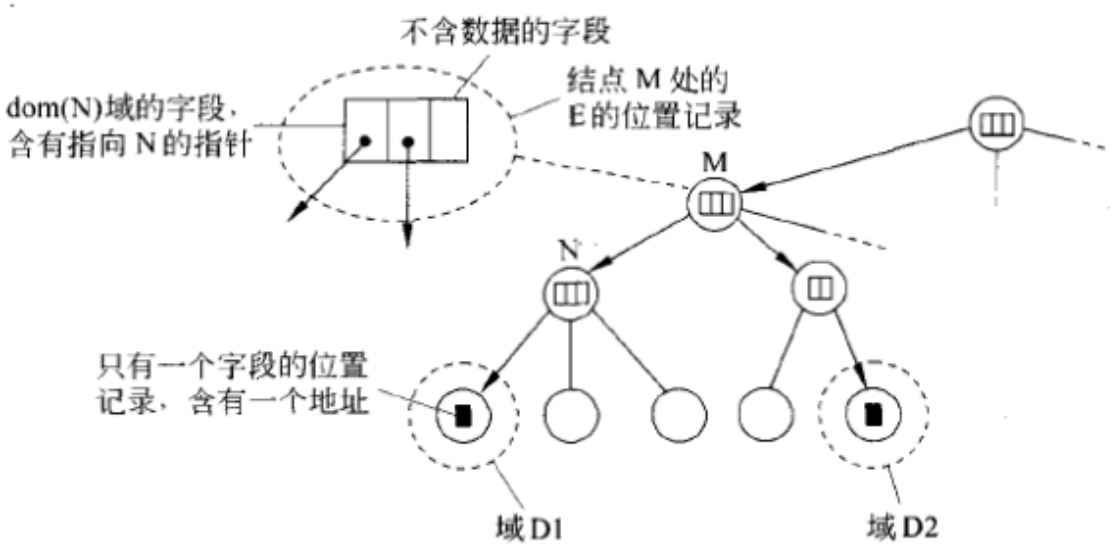


图 5.6　实体信息存储示例，这里的实体在不同的叶域中拥有两个地址

分层定位服务中的查询操作实现：

希望查询实体E的客户，向叶域D的目录结点发送查询请求。如果没有，就往父节点转发；如果有，就把请求转发给那个子域的目录结点，子域会依次进一步向树的下方转发请求。储存在**叶结点的位置记录会包含E在叶域的地址。**将这个地址返回给客户。

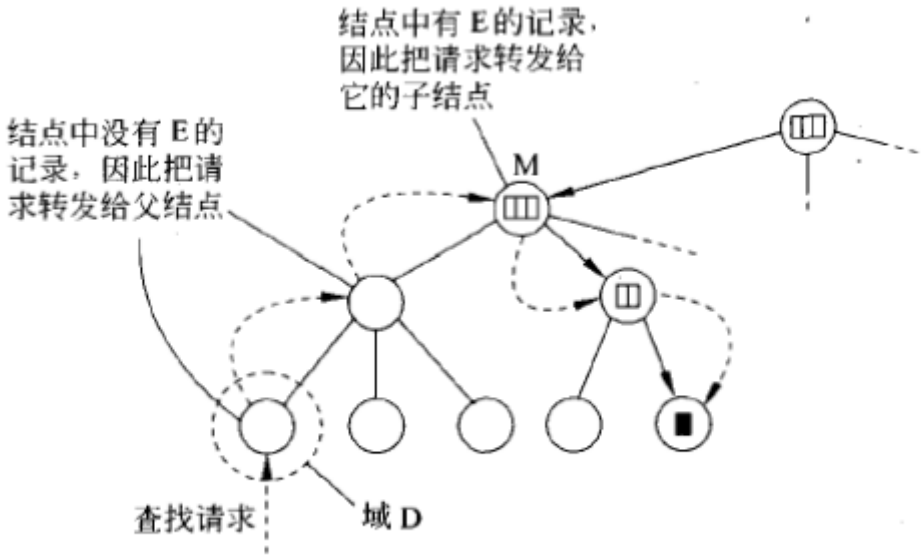
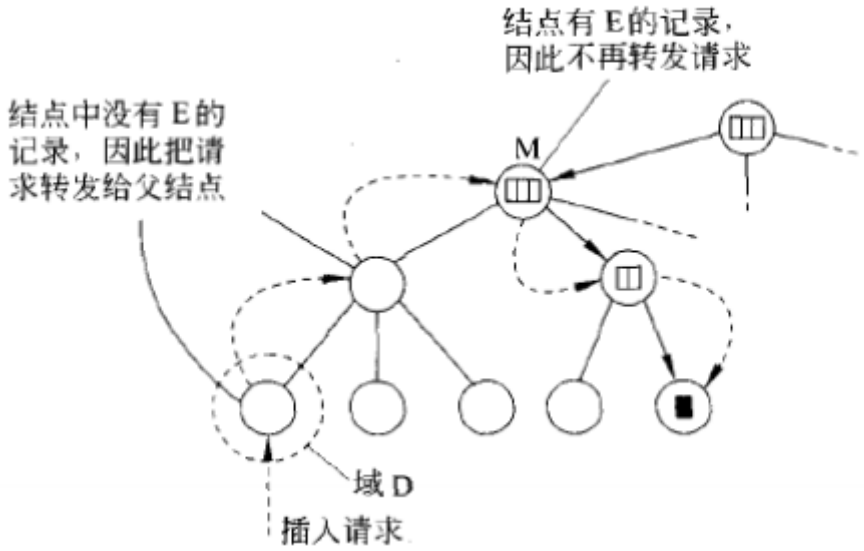
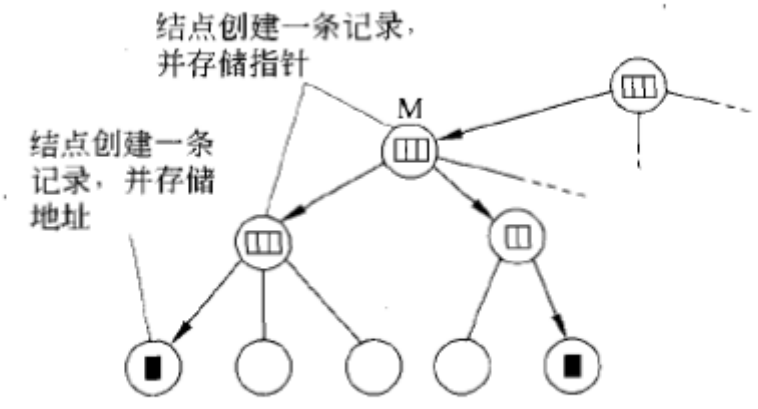


图 5.7　在分层组织的定位服务中的位置查找

更新插入操作也类似，假设实体E在叶域D中创建了一个副本，就需要插入这个副本的地址。插入操作从某一个D的叶结点开始，然后将请求转发给父结点，以此类推**直到到达储存了位置记录的目录节点M（第一个知道实体E的节点M）。**



(a)



(b)

图 5.8　更新操作

然后M会建立一个**指针指向转发插入请求的子结点**，也就是M新增一条记录：如果要访问实体E也可以访问左下的节点。然后左下的节点也这样做，最后在域D的目录节点dir(D)（对于只有一个节点的叶域，目录节点就是自身）中，新增一条记录记录实体E的真实地址。也可以从下向上建立指针链。

删除操作与插入操作也类似，如果需要删除叶域D中的实体E的地址，那么需要请求目录节点dir(D)从它为E储存的位置记录中删除地址，并通知更高一层结点。

5.3 结构化命名

5.3.1 名称空间

结构化名称空间可以表示为带有标记的具有**两种类型的结点**的有向图

- 叶结点：表示一个具名实体，没有分支边

  叶节点还可以存储实体的属性、状态等信息
- 目录节点：具有一定数量的分支边。**每条边用一个名称来标记**

命名图的每一个结点都被认为是分布式系统的一个**实体**，且都有一个标识符

目录节点存储一张**目录表**，每条分支边用一个（**边标签，结点标识符**）对来表示

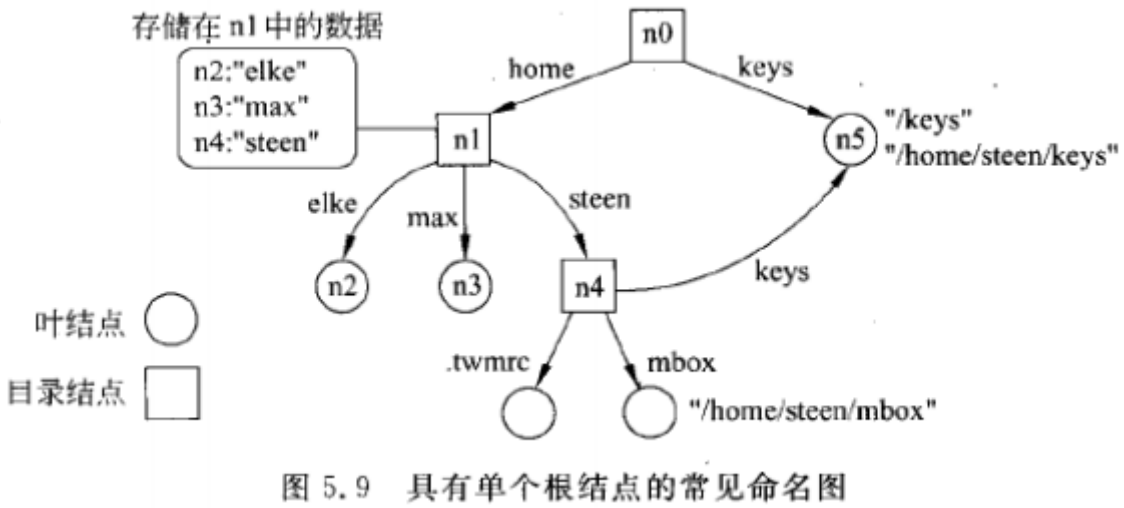


图 5.9　具有单个根结点的常见命名图

命名图中的每个路径可以通过路径中的**边的标签序列**指向

$$N:\langle label\text{-}1, label\text{-}2, \cdots, label\text{-}n \rangle$$

其中N为路径中第一个结点。这样的序列称为路径名。如果路径名中第一个节点是命名图的根，那么该路径名称为**绝对路径名**。否则称为**相对路径名**

5.3.2 名称解析

给定一个路径名，查找出存储在由该名称所指向的节点中的任何 信息，查询名称的过程称为**名称解析**

闭包机制：知道如何启动以及在何处启动名称解析通常称为 闭包机制

www.distributed-systems.net, 从DNS服务器开始；

/home/maarten/mbox: 从本地的NFS服务器开始；

13587569903：拨打电话号码；

222.200.145.180: 把消息路由到特定的IP地址；

5.3.2.2 链接

与名称解析紧密联系的是**别名的使用**，别名是同一实体的另外一个名称。

硬链接

我们所描述的路径名即用于在命名图中按照特定路径搜索节点的 名字就是“硬链接”

比如下面，/keys 和 /home/steen/keys都是对n5的硬链接

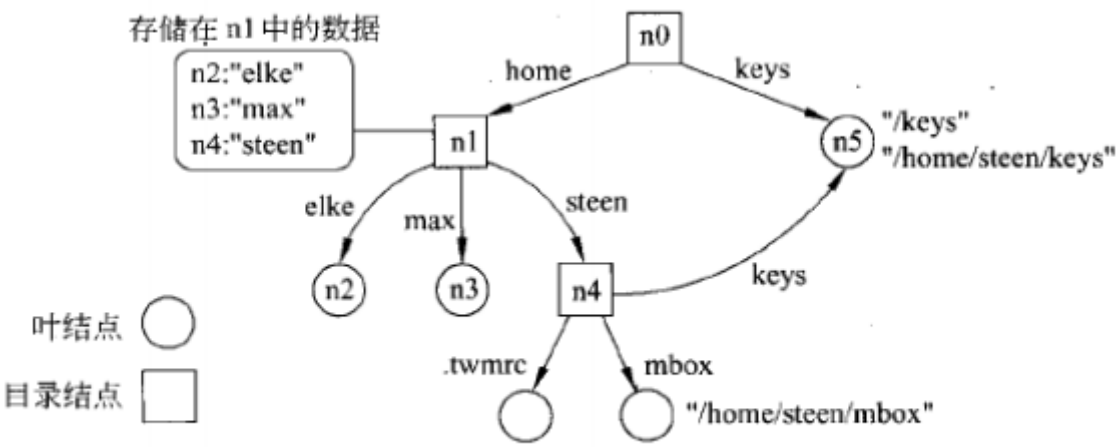
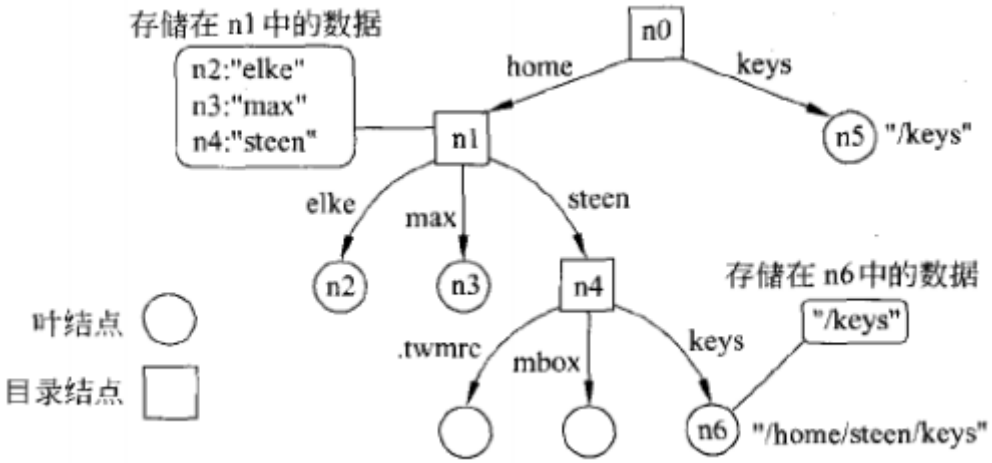


图 5.9 具有单个根结点的常见命名图

软链接

软链接（符号链接）：允许一个节点 N 包含另外一个节点名字

在叶节点n6中保存指向n5的的路径名。当第一次解析到n6的时候，名称解析会返回n6中存储的路径名，然后进一步解析新的路径名找到n5



5.3.2.2 挂载

命名解析也可以应用于**合并不同的命名空间**，通过挂载的方法透 明地实现；将另异空间的节点标识符与当前命名空间的节点相关联；

外部命名空间： 需要访问的命名空间

挂接点： 在**当前命名空间**中用于**存储节点标识符的目录节点** 成为挂接点

挂载点： 外部名称空间中的目录节点称为挂载点；挂载点是 命名空间的“根”

在名称解析的过程中，挂载点被查询，并通过访问目录表来完成解析

5.3.3 名称空间的实现

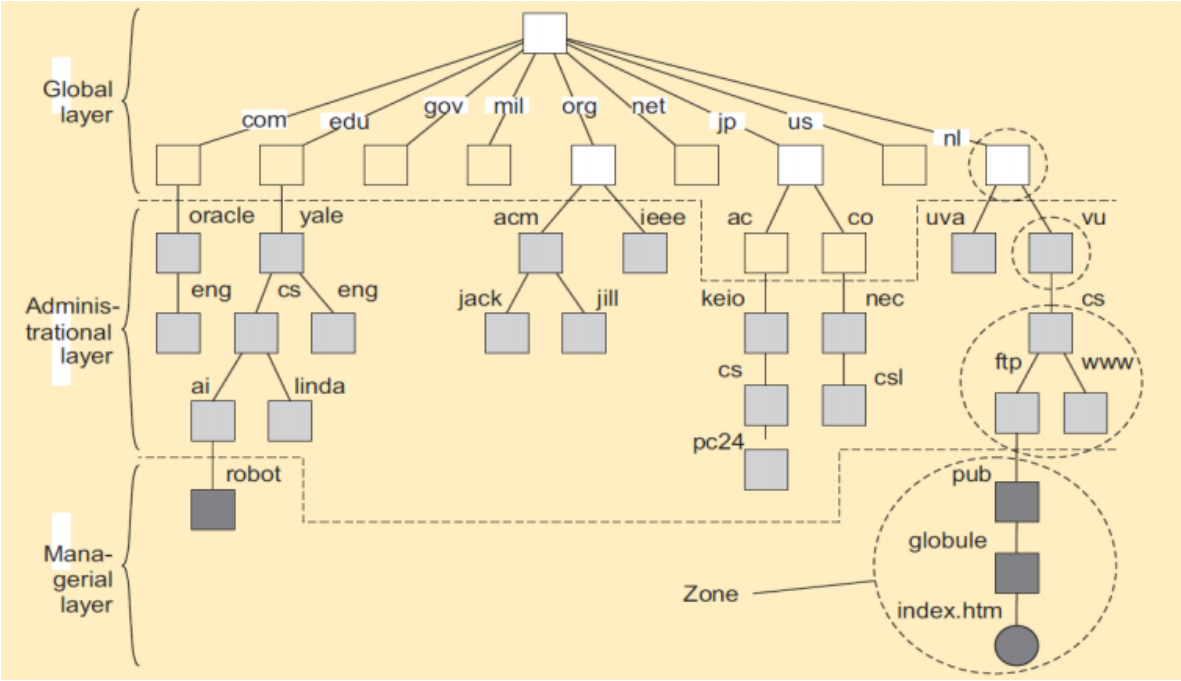
名称空间形成了命名服务的核心，准确的说，命名服务就是一种允许用户和进程添加、删除和查找名称的服务。命名服务由**名称服务器实现**，如果分布式系统居于在一个局域网内，那么可以只用一台名称服务器来实现命名服务。

跨越多个机器的分布式命名解析过程与命名空间管理是通过将 命名图分布在多个节点上实现的

5.3.3.1 名称空间的分布性

名称空间通常分层组织

- 全局层： 由最高级别的节点构成，即由根节点以及其他逻辑上 靠近根节点的目录节点组成。特点是：稳定，目录表很少改变， 可以代表组织或者组织群
- 行政层： 由那些在单个组织内一起被管理的目录节点组成。行 政层中的目录节点所具有的特点是代表属于同一组织或行政单 位的实体组；相对稳定，但是比全局层的目录变化频繁；
- 管理层： 由经常改变的节点组成。如代表本地主机的节点及本 地文件系统等，由终端用户维护；



5.3.3.2 名称解析的实现

迭代命名解析

加入要解析的绝对路径名为：

root: ⟨nl, vu, cs, ftp, pub, globe, index.txt⟩

将被解析。如果使用 URL 表示法，那么这个路径名将对应于 ftp：//ftp. cs. vu. nl/pub/globe/index. txt。现在可以用两种方法实现名称解析。

名称解析程序把完整的名称转发给根名称服务器，根服务器会尽可能解析路径名（比如这里就解析除了nl）然后把与**nl相关的名称服务器的地址**返回给客户。

接着客户把剩下的路径名

nl: ⟨vu, cs, ftp, pub, globe, index. txt⟩)

转发给下一台名称服务器。这台服务器只能解析vu，然后返回**相关名称服务器地址**

接下来客户的名称解析程序将同下一台名称服务器联系，该服务器响应的结果是解析标识符 cs，随后还有 ftp，最后会连同路径名 ftp：⟨pub, globe, index. txt⟩一起返回该 FTP 服务器的地址。客户端接着再与该 FTP 服务器联系，请求它解析原始路径名的最后一部分。随后 FTP 服务器会解析标识符 pub、globe 和 index. txt，并发送被请求的文件（这里是使用 FTP）。图 5. 15 显示了这个迭代名称解析过程。符号 #⟨cs⟩用来指明一台服务器的地址，该服务器负责处理⟨cs⟩涉及到的结点。

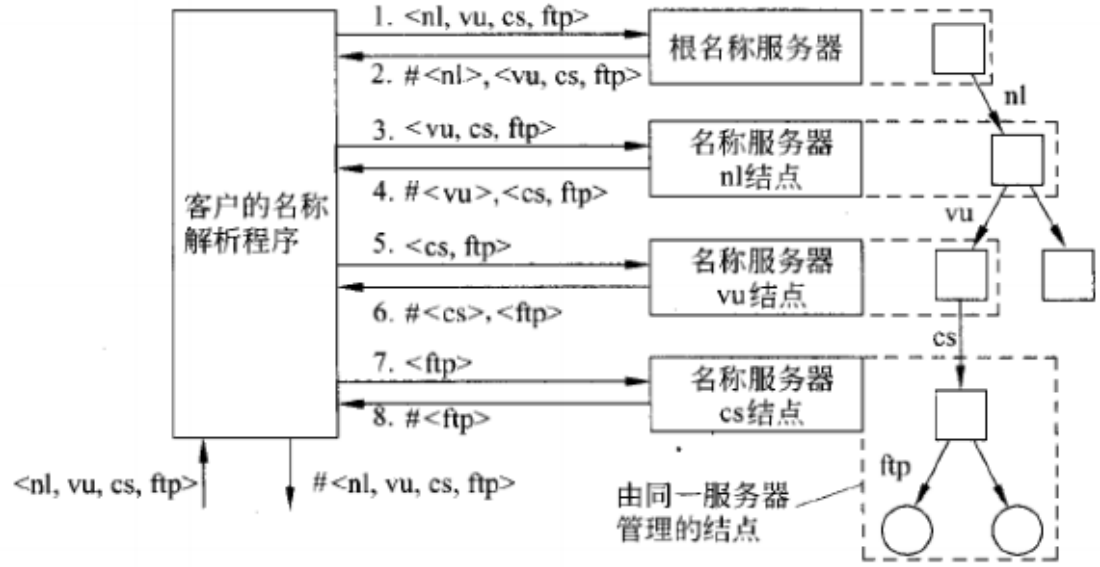


图 5.15　迭代名称解析原理

实际上，最后一步，即联系 FTP 服务器并请求它发送路径名 ftp:〈pub,globe,index.txt〉所指文件的那一步，是由客户进程独立完成的。换句话说，客户通常只把路径名 root:〈nl,vu,cs,ftp〉转交给名称解析程序，希望从名称解析程序那里得到可以联系的那一台 FTP 服务器的地址，图 5.15 也说明了这一点。

最后一步联系FTP服务器并请求它传送指定文件那一步是由客户独立完成的

递归命名解析

迭代名称解析的一种替代方法是在解析过程中使用递归。

其实也就是相当于当前名称服务器直接去找下一个名称服务器。不用将地址传回客户端让客户端去请求。

迭代名称解析的一种替代方法是在名称解析过程中使用递归。与返回客户解析程序每个中间结果不同，在使用**递归名称解析**(recursive name resolution)时，名称服务器会把结果传递给它找到的下一台服务器。举例来说，当根名称服务器找到实现 nl 结点的名称服务器所用的地址后，它会请求该名称服务器解析路径名 nl:〈vu,cs,ftp,pub,globe,index.txt〉。在同样使用递归名称解析的情况下，下一台服务器将解析完整的路径名，最终会把 index.txt 文件返回根服务器，根服务器再依次把该文件传递给客户的名称解析程序。

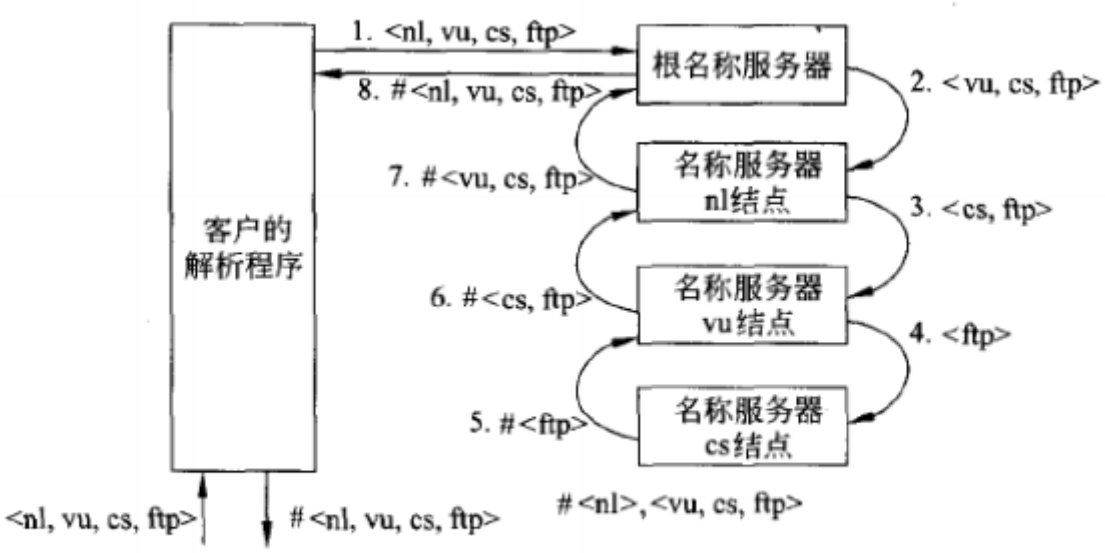


图 5.16　递归名称解析原理

缺点：

它要求每台名称服务器都具有较高的性能，相当于要求名称服务器对路径名进行完整解析

优点：

1. 递归名称解析的缓存结果更有效一些
2. 减少通信开销