命名

命名在所有计算机系统中都扮演着重要的角色。他们用于资源共享，唯一的标识实体，引用位置等。命名的一个重要课题是，名称可以解析为它所引用的实体（类似URL命名可以解析到一个具体的文件）。因此，名称解析允许进程访问指定的实体。要解析名称，需要实现一个命名系统。分布式系统和非分布式系统命名的区别在于命名系统的实现方式。

在分布式系统中，命名系统的实现本身常常分布在多台机器上。如何实现这种分布对命名系统的效率和可伸缩性起着关键作用。在本章中，我们将集中讨论在分布式系统中使用命名的三种不同的重要方式。

首先，我们考虑所谓的平面命名系统。在这样的系统中，实体由一个标识符引用，该标识符在原则上没有任何意义。此外，平面命名没有任何结构，这意味着我们需要特殊的机制来跟踪这些实体的位置。我们讨论了各种方法，从转发链接链到分布式哈希表，再到分层位置服务。

实际上，人们更喜欢使用可读的命名（个人理解，即结构化的命名，容易猜到意思）。这类名称通常是结构化的，从Web页面的引用方式就可以看出这一点。构化名称允许以高度系统的方式查找负责指定实体的服务器，域名系统就是一个例子。我们将讨论一般原则以及可伸缩性问题。

最后，人类通常喜欢用各种特征来描述实体，这导致我们需要用分配给实体的属性来解决描述。正如我们将看到的，这种类型的名称解析是出了名的困难，特别是与搜索相结合。

5.1名称、标识符和地址

让我们先仔细看看命名到底是什么。分布式系统中的名称是一串用于引用实体的位或字符。分布式系统中的实体实际上可以是任何东西。典型的例子包括主机、打印机、磁盘和文件等资源。通常显式命名的其他著名实体示例包括进程、用户、邮箱、新闻组、Web页面、图形化窗口、消息、网络连接等等。

实体可以操作。例如，打印机之类的资源提供一个接口，其中包含打印文档、请求打印作业状态等操作。此外，网络连接之类的实体可以提供发送和接收数据、设置服务质量参数、请求状态等操作。

要对实体进行操作，必须访问它，为此我们需要一个**访问点**。访问点是分布式系统中另一种特殊的实体。访问点的命名称为地址。实体的访问点的地址也简单地称为该实体的地址。

一个实体可以提供多个访问点。作为比较，电话可以看作是一个人的访问点，而电话号码对应一个地址。事实上，现在很多人都有几个电话号码，每个号码对应一个可以到达的地方。在分布式系统中，访问点的典型例子是运行特定服务器的主机，其地址由IP地址和端口号组合而成。

实体可以在一段时间内更改其访问点。例如，当移动计算机移动到另一个位置时，它通常被分配一个与以前不同的IP地址。同样，当一个人搬到另一个城市或国家时，通常也需要更改电话号码。同样，换工作或换互联网服务提供商，也意味着换电子邮件地址。

因此，地址只是一种特殊的命名：它引用实体的访问点。因为访问点与实体紧密关联，所以使用访问点的地址作为关联实体的常规名称似乎很方便。然而，这几乎从未做过，因为这样的命名通常是非常死板的，而且通常对人类不友好。

例如，经常重新组织分布式系统，使特定的服务器现在运行在与以前不同的主机上，这种情况并不少见。在过去旧机器上运行的服务，现在可能运行着一个完全不同的服务。换句话说，实体可以很容易地更改访问点，或者将访问点重新分配给不同的实体。如果一个地址用于引用一个实体，那么在访问点更改或重新分配给另一个实体时，我们将拥有一个无效的引用。因此，使用独立于关联服务器地址的独立名称来表示服务要好得多（就是说，不用地址+端口，因为这样客户端修改起来会很麻烦，如果使用域名，即使修改了，客户端也不用修改访）。

同样，如果一个实体提供多个访问点，则不清楚使用哪个地址作为引用。例如，许多组织将其Web服务分布在多个服务器上。如果我们将这些服务器的地址用作Web服务的参考，那么将不容易做出应该选择哪个地址作为最佳地址的选择。同样，更好的解决方案是为Web服务使用一个独立于不同Web服务器地址的单一名称。

这些示例说明，独立于其地址的实体的名称通常更容易使用，也更灵活。这样的名称称为**位置无关**。

除了地址之外，还有其他类型的名称需要特殊处理，比如用于惟一标识实体的名称。真正的标识符是一个具有以下属性的名称[Wieringa and de Jonge, 1995]：

1. 一个标识符对应最多一个实体
2. 每个实体最多对应一个标识符
3. 一个标识符总是对应着相同的实体

通过使用标识符，更容易明确地引用实体。例如，假设两个进程分别通过标识符引用一个实体。要检查进程是否引用相同的实体，只要测试两个标识符是否相等就足够了。如果两个进程使用常规的、非惟一的、非标识的名称，这样的测试是不够的。例如，“John Smith”这个名字不能仅作为一个人的唯一引用。

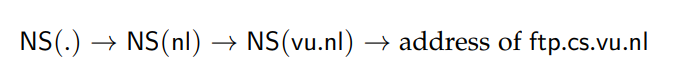
同样，如果一个地址可以重新分配给另一个实体，我们就不能使用地址作为标识符。考虑一下电话号码的使用，从某种意义上说，电话号码在一段时间内经常是指同一个人或组织，这是相当稳定的。然而，使用电话号码作为标识符将不起作用，因为它可以在一段时间内重新分配。因此，鲍勃的新面包店可能会在很长一段时间内接到爱丽丝旧古董店的电话。在这种情况下，使用Alice的真实标识符而不是她的电话号码会更好。

地址和标识符是两种重要的名称类型，它们用于非常不同的目的。在许多计算机系统中，地址和标识符仅以机器可读的形式表示，即以位串的形式表示。例如，以太网地址本质上是一个48位的随机字符串。同样，内存地址通常表示为32位或64位字符串。

另一种重要的名字类型是为人类量身定做的，也被称为**人类友好的命名**。与地址和标识符相反，对人友好的名称通常表示为字符串。这些名字以许多不同的形式出现。例如，Unix系统中的文件具有字符-字符串名称，通常可以长达255个字符，并且完全由用户定义。类似地，DNS名称表示为相对简单的不区分大小写的字符串。

名称、标识符和地址使我们进入本章的中心主题:如何将名称和标识符解析为地址？在讨论各种解决方案之前，务必认识到分布式系统中的名称解析与消息路由之间常常存在密切的关系。原则上，命名系统维护一个name-toaddress绑定，其最简单的形式就是一个(名称、地址)表。然而，在跨越大型网络的分布式系统中，需要为其命名许多资源，因此集中式表（名称，地址）无法工作。

相反，经常发生的情况是将名称分解为几个部分，比如ftp.cs.vu.nl。名称解析是通过对这些部分的递归查找来实现的。例如，客户端需要知道FTP .cs.vu.nl所命名的FTP服务器的地址。将首先解析nl以查找负责以nl结尾的名称的服务器NS(nl)，然后将其余名称传递给服务器NS(nl)（就是DNS查找方法，nl->uv->cs->ftp）。然后，此服务器可能将名称vu解析为服务器NS(vu.nl)，该服务器负责以vu.nl结尾的名称，即能进一步处理余下名称[ftp.cs](ftp://ftp.cs)的服务器。最终，这将导致将名称解析请求路由为：



其中NS(.)表示可以返回NS(nl)地址的服务器，也称为根服务器。NS(vu.nl)将返回FTP服务器的实际地址。有趣的是，名称解析和消息路由之间的界限开始变得模糊。

在接下来的章节中，我们将讨论几种不同类型的命名系统。首先，我们将研究如何将标识符解析为地址。在本例中，我们还将看到一个示例，其中名称解析实际上与消息路由难以区分。然后，我们考虑对人友好的名称和描述性名称。

5.2 Flat naming

上面，我们解释了标识符可以方便地惟一地表示实体。在许多情况下，标识符只是随机的位字符串，我们方便地将其称为非结构化或flatname。此类名称的一个重要属性是，它不包含关于如何定位关联实体的访问点的任何信息。在下面的文章中，我们将研究如何解析平面名称，或者，同样地，当只给出实体的标识符时，如何定位该实体。

**简单的解决方案**

我们首先考虑定位实体的两个简单解决方案:广播和转发指针。这两种解决方案主要适用于局域网。然而，在这样的环境中，它们常常能很好地完成工作，这使得它们的简单性特别具有吸引力。然而，使用广播和转发指针会带来可伸缩性问题。广播或多播在大规模网络中很难有效地实现，而长链转发指针会带来性能问题，容易导致链路中断。

**广播**

考虑一个建立在提供高效广播设施的计算机网络上的分布式系统。通常，这些设备是由局域网提供的，在局域网中，所有的机器都连接到一根电缆或其逻辑等效物上。此外，本地无线网络也属于这一类。

在这样的环境中定位实体很简单:将包含实体标识符的消息广播给每台机器，并请求每台机器检查它是否拥有该实体。只有能够为实体提供访问点的机器才会发送包含该访问点地址的回复消息。这一原理被用于互联网地址解析协议(ARP)中，当只给一台机器一个IP地址时，它可以用来查找该机器的数据链接地址[Plummer, 1982]。本质上，一台机器在本地网络上广播一个数据包，询问给定IP地址的所有者是谁。当消息到达一台机器时，接收者检查它是否应该侦听所请求的IP地址。如果是，它发送一个包含它的以太网地址的应答包。

一旦网络规模增大，这种广播的方式将不再起作用。请求消息不仅浪费了网络带宽，而且，更严重的是，太多的主机可能会被它们无法响应的请求中断。一种可能的解决方案是切换到多播，通过多播，只有一组受限的主机接收请求。例如，以太网支持直接在硬件上进行数据链路级多播。

多播还可以用来定位点对点网络中的实体。例如，Internet支持网络级的多播，允许主机加入特定的多播组。这些组由多播地址标识。当主机向多播地址发送消息时，网络层尽最大努力将消息传递给所有组成员。Deering和Cheriton[1990]以及Deering等人讨论了Internet中多播的高效实现。

多播地址可以用作多个实体的通用位置。例如，考虑一个组织，其中每个员工都拥有自己的移动计算机。当这样的计算机连接到本地可用的网络时，它被动态分配一个IP地址。此外，它还加入一个特定的多播组。当一个进程想要定位计算机a时，它发送一个“a在哪里?”向多播组请求。如果连接到A，它将使用当前IP地址进行响应。

使用多播地址的另一种方法是将其与复制的实体关联，并使用多播来定位最近的副本。当向多播地址发送请求时，每个副本使用其当前(正常)IP地址进行响应。选择最近的副本的一种简单方法是选择最先收到响应的副本，但事实证明，选择最近的副本通常并不那么容易。

**Forwardingpointers（没搞懂啥意思）**

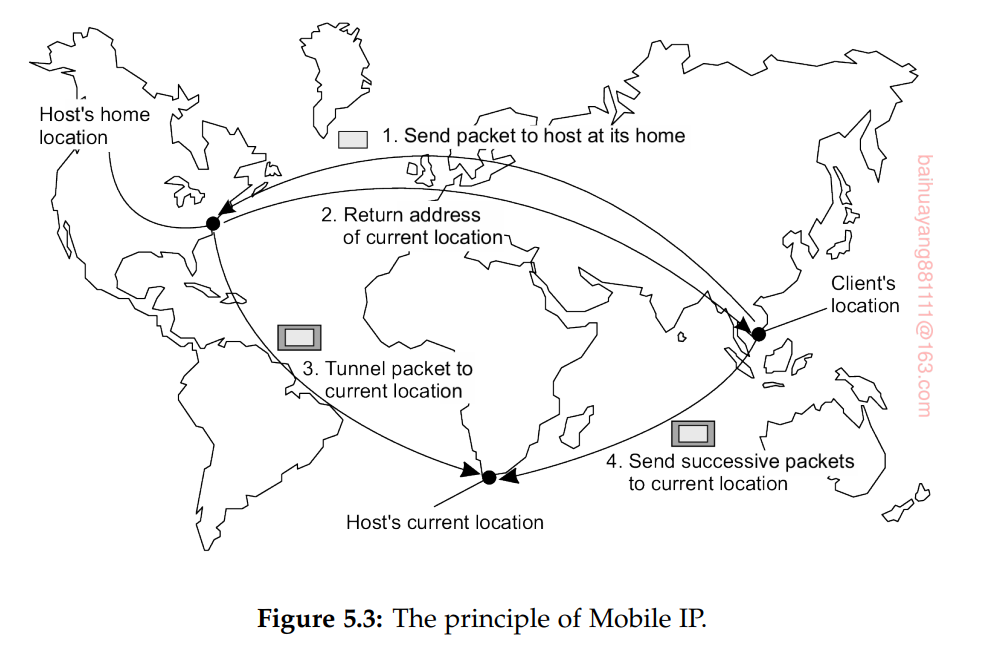
定位移动实体的另一种流行方法是使用forwarding pointers。原理很简单:当一个实体从A移动到B时，它会留下一个指向它在B的新位置的引用。这种方法的主要优点是它的简单性: ，例如通过使用传统的命名服务，只要找到了实体客户机就可以按照转发指针链查找当前地址。但它也有缺点。首先，如果不采取特别措施，一个高度移动的实体的链可能会变得非常长，以至于定位该实体的成本高得令人望而却步。其次，链中的所有中间位置都必须根据需要维护其转发指针链的一部分。第三个(也是相关的)缺点是容易受到断开链接的影响。一旦任何转发指针丢失，就无法再访问该实体。因此，一个重要的问题是保持链相对较短，并确保转发指针是健壮的。

**Home-based 方法 （这跟naming有啥关系）**

在大型网络中支持移动实体的一种流行方法是引入home location，它跟踪实体的当前位置。可以使用特殊技术来防止网络或进程故障。在实践中，通常选择home位置作为创建实体的位置。

基于home的方法用作基于转发指针的位置服务的回退机制。采用基于home的方法的另一个例子是移动IP [Johnson et al.， 2004]，我们在Note 3.9中简要解释。每个移动主机使用一个固定的IP地址。所有到该IP地址的通信最初都指向移动主机的home代理（很像基站那种，一个基站负责一片区域）。此home代理位于与移动主机的IP地址中包含的网络地址相对应的局域网中。在IPv6的情况下，它是作为一个网络层组件实现的。当移动主机移动到另一个网络时，它会请求一个临时地址，以便进行通信。这个地址将在home代理中进行注册。

当home代理接收到移动主机的数据包时，它会查找主机（此主机非彼主机）的当前位置。如果主机位于当前本地网络上，则只转发数据包。否则，它将通过隧道传输到主机的当前位置，也就是说，作为数据包返回给主机。同时，信息包的发送方被告知主机的当前位置。原理如图5.3，这里Ip地址被用作移动主机的标识（1.发送方发送信息，2.代理通知发送方真正目的地地址 3代理将其转发给真正的目的地，4.发送发发送后续的信息到真正的目的地）。



一个重要的方面是，整个机制在很大程度上对应用程序是透明的。换句话说，应用程序可以使用与移动主机关联的原始IP地址。客户端软件是独立于应用程序的通信层的一部分，它将处理到目标当前位置的重定向。同样，在目标的位置，已被隧道化的消息将被解压缩并传递到移动主机上的应用程序，就像它使用其原始地址一样（就是从隧道传来的消息看起来和从发送方直接发送过来是一样的）。的确，移动IP建立了高度的位置透明性（从何得出啊）。）

图5.3还说明了大型网络中基于home方法的一个缺点。要与移动实体通信，客户机首先必须与home联系，home可能位于与实体本身完全不同的位置。其结果是通信延迟的增加。

基于home的方法的另一个缺点是使用固定的home位置。首先，必须确保home位置始终存在。否则，连接将变得不可能。当移动实体迁移到另一个网络中，而且原来的home并不存在于这个网络中时，问题就更加严重了。在这种情况下，如果home能和主人一起搬家就更好了。

这个问题的解决方案是在传统的命名服务中注册home，并让客户机首先查找home的位置。因为可以假定主位置相对稳定，所以可以在查找后有效地缓存该位置。

**分布式哈希表（distributed hash table , DHT）**

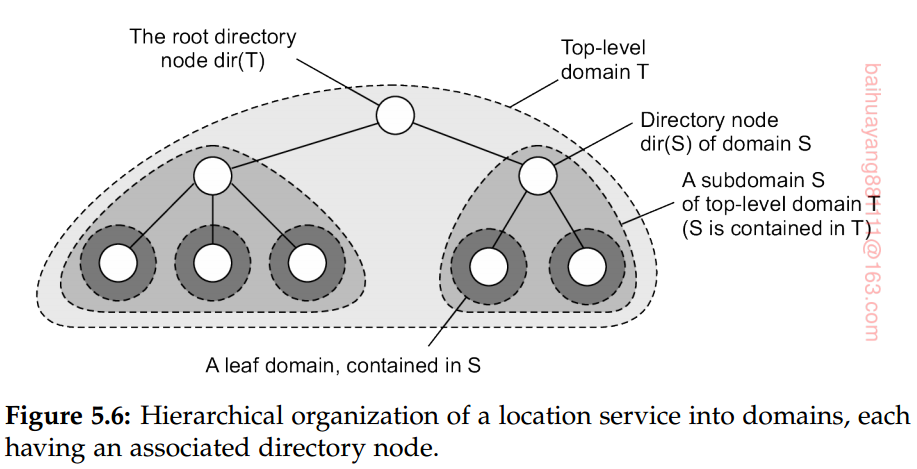
注：这一节的内容请自行查阅相关资料。

**分层的方法**

现在，我们讨论层次定位方案的一般方法，包括一些优化。我们提出的方法是基于全球定位服务的。这是一种通用的定位服务，它代表了为所谓的个人通信系统提出的许多层次定位服务。

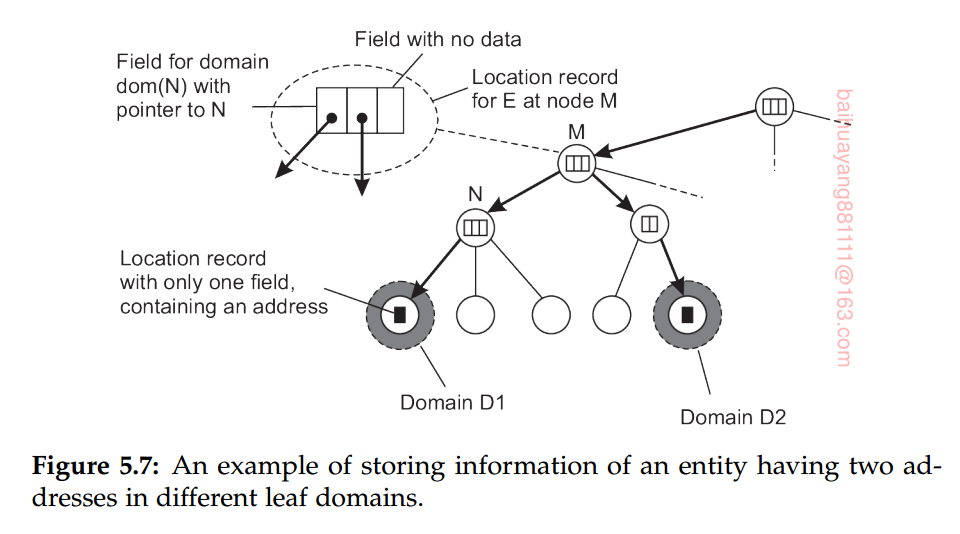
在层次结构中，网络被划分为一系列域。有一个跨整个网络的顶级域。其他的域都可以成为这个域的子域。最底层的域成为**叶子域**，代表了现实网络中一台实际的计算机或者手机等通信机器。一般的假设是，在较小的域中，将消息从一个节点传输到另一个节点所需的平均时间比在较大的域中要短。

每个域D都有一个关联的目录节点dir(D)，用于跟踪该域中的实体。这就是一个目录树节点。顶级域的目录节点称为根(目录)节点，它知道所有实体。将网络组织成域和目录节点的一般结构如图5.6所示。



为了跟踪实体的位置，当前位于域D中的每个实体都由目录节点dir(D)中的位置记录表示。叶子域D的目录节点N中的实体E的位置记录包含该域中实体E的当前地址。相反，在他们上一层的包含了D的D’域，会存在一个目录节点N’，这个N’有一个指向包含了E的N的指针。同样，N’的父节点也存在一个包含E的并对N的指针。因此，根节点将为每个实体拥有一个位置记录，其中每个位置记录存储指向下一个较低级别子域的目录节点的指针，该记录的关联实体当前位于该子域。

一个实体如果被复制，可能会有多个地址。如果一个实体在叶域D1和D2中分别有一个地址，那么包含D1和D2的最小域的目录节点将有两个指针，每个指针对应一个包含子域的地址。这将导致树的一般组织，如图5.7所示。



现在让我们考虑一下在这样一个分层的位置服务中查找操作是如何进行的。如图5.8所示，希望定位实体E的客户机向客户机所在的叶子域D的目录节点发出查找请求。如果目录节点不为实体存储位置记录，则该实体当前不在D中。因此，节点将请求转发给其父节点。注意，父节点表示比其子节点更大的域。如果父节点也没有E的位置记录，则查找请求将被转发到更高的级别，以此类推。一旦请求到达存储实体E位置记录的目录节点M，我们就知道E位于节点M表示的域dom(M)中的某个位置。在图5.8中，M表示存储一个位置记录，其中包含指向其子域之一的指针。然后，查找请求被转发到该子域的目录节点，该目录节点又将其转发到树的下一级，直到请求最终到达叶子节点。存储在叶子节点中的位置记录将包含叶子域中E的地址。然后可以将此地址返回给最初请求进行查找的客户机。

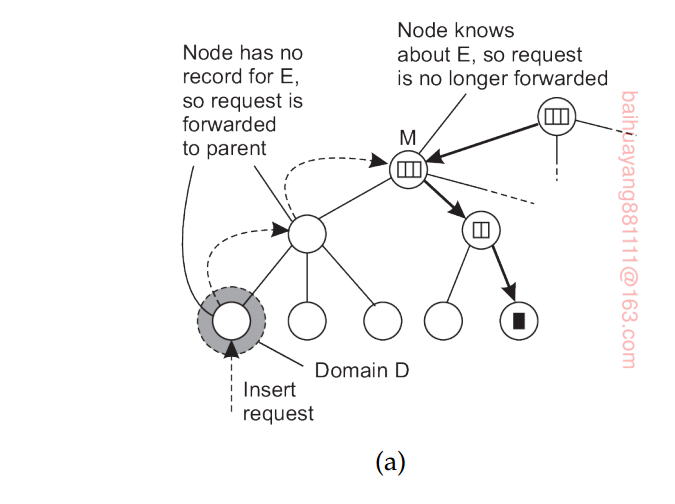
关于分层位置服务的一个重要发现是，查找操作利用了位置。原则上，实体是在一个以请求客户机为中心的逐渐增加的环中搜索的。每次将查找请求转发到下一个更高级别的目录节点时，都会扩展搜索区域（就是查询范围变大了呗）。最糟糕的情况时，一直向上查询到了根节点。因为根节点为每个实体都有一个位置记录，所以可以简单地沿着指向其中一个叶子节点的指针的下行路径转发请求。

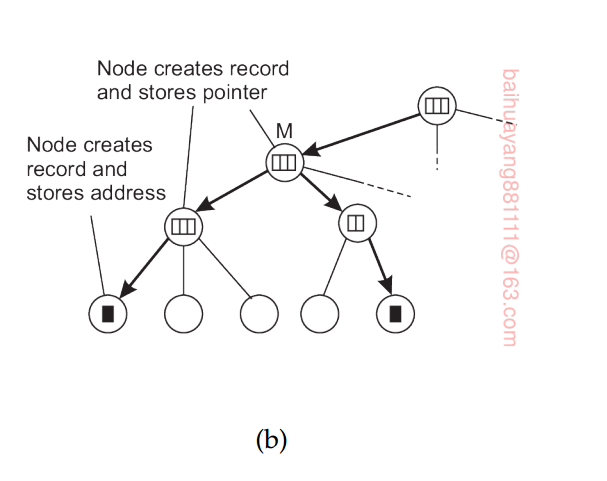
更新操作以类似的方式利用局部性，如图5.9所示。考虑这样一个实体E，它在叶子域D中创建了自己的一个副本，需要在D域为其插入地址。插入在D的叶子节点dir(D)处启动，该节点立即将插入请求转发给其父节点。父节点也将转发insert请求，直到它到达一个目录节点M，该节点已经为E存储了一个位置记录。

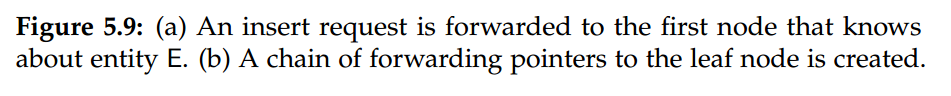
然后，节点M将在E的位置记录中存储一个指针，指向转发插入请求的子节点。此时，子节点为E创建一个位置记录，其中包含指向下一个较低级别节点的指针，该节点是请求发出的地方。这个过程将继续，直到到达启动插入的叶子节点为止。最后，叶子节点在关联的叶子域中创建一个带有实体地址的记录。

插入如上所述的地址将导致以自顶向下的方式安装指针链，从具有实体E位置记录的最低级别目录节点开始。另一种方法是在将插入请求传递给父节点之前创建位置记录。换句话说，指针链是自底向上构造的。后一种方法的优点是，地址可以尽快用于查找。因此，如果父节点暂时无法访问，仍然可以在当前节点表示的域中查找地址。

删除操作类似于插入操作。当需要删除叶子域D中的实体E的地址时，目录节点dir(D)被请求从E的位置记录中删除该地址。如果该位置记录变为空，则可以删除该记录。在这种情况下，dir(D)的父节点希望删除指向dir(D)的指针。如果E在父节点上的位置记录现在也变为空，那么也应该删除该记录，并通知下一个更高级别的目录节点。同样，这个过程将继续进行，直到从位置记录中删除一个指针，该位置记录在之后或到达根之前保持非空。







5.3 结构化命名

Flat name对机器来说很好，但对人类来说通常不太方便使用。作为一种替代方法，命名系统通常支持由简单的、人类可读的名称组成的结构化名称。不仅文件命名，Internet上的主机命名也遵循这种方法。在本节中，我们将重点讨论结构化名称以及将这些名称解析为地址的方式。

**命名空间**

这些名称通常是由**命名空间**来组织的。结构化名称的命名空间可以表示为具有标记和有向图的两种类型节点。叶节点表示一个已命名的实体，并且没有传出边。叶子节点通常存储它所代表的实体的信息(例如，它的地址)，以便客户机可以访问它。或者，它也可以存储该实体的状态，例如在文件系统中，叶子节点实际上包含它所表示的完整文件的所有信息。我们将在下面返回关于节点的内容。

与叶子节点相反，目录节点有许多输出边，每条边都标有名称，如图5.11所示。命名图中的每个节点，特别是具有关联的标识符，都被认为是分布式系统中的另一个实体。目录节点存储一个表，其中输出边缘表示为一对(节点标识符、边缘标签)。这样的表称为**目录表**。

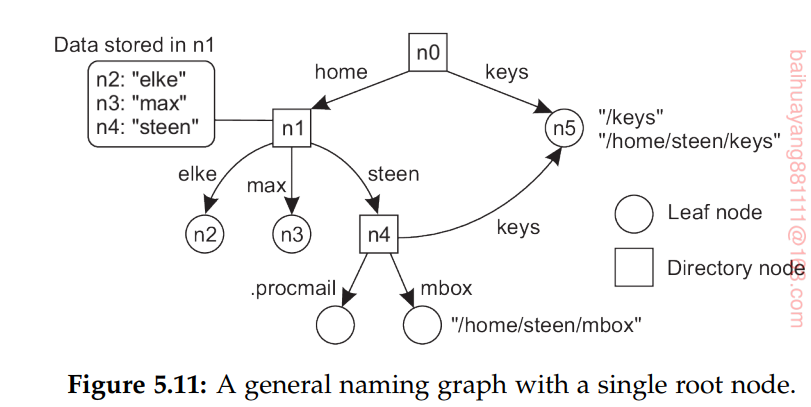


图5.11所示的命名图有一个节点，即n0，它只有传出的边，没有传入的边。这样的节点称为命名图的根(节点)。虽然命名图可以有多个根节点，但是为了简单起见，许多命名系统只有一个根节点。命名图中的每条路径都可以通过与该路径边对应的标签序列来引用，如N:[label1, label2，…，其中N为路径中的第一个节点。这样的序列叫做**路径名**。如果路径名的第一个节点是根节点，那么这个路径名就叫做**绝对路径名**。否则，就叫做**相对路径名**。

重要的是要认识到，名称总是组织在名称空间中。因此，名称总是相对于目录节点定义的。从这个意义上说，“绝对名称”这个词有点误导人。同样，全局名称和本地名称之间的区别也常常令人混淆。**全局名称**是表示相同实体的名称，无论该名称在系统中的何处使用。换句话说，全局名称总是相对于相同的目录节点进行解释。相反，本地名称的解释取决于该名称在何处使用。换句话说，本地名称本质上是一个相对名称，它所包含的目录(隐式)是已知的。

命名图的这种描述接近于许多文件系统中实现的内容。文件系统中的路径名通常表示为一个字符串，其中标签由一个特殊的分隔符分隔，如斜杠(" / ")。此字符还用于指示路径名是否绝对。例如，在图5.11中，没有使用n0:[home, steen, mbox]，即实际的路径名，而是使用它的字符串表示形式/home/steen/mbox。还要注意，当有多个路径指向同一个节点时，可以用不同的路径名表示该节点。还要注意，当有多个路径指向同一个节点时，可以用不同的路径名表示该节点。路径名称的字符串表示形式也可以很好地应用于命名图，而不是只用于文件系统。在Plan 9 [Pike et al.， 1995]中，所有资源(如进程、主机、I/O设备和网络接口)的命名方式与传统文件相同。这种方法类似于为分布式系统中的所有资源实现一个单一的命名图。

有许多不同的方法来组织名称空间。如前所述，大多数名称空间只有一个根节点。在许多情况下，命名空间也是严格分层的，因为命名图被组织为树。这意味着除了根节点外，每个节点都有一条传入的边;根没有传入的边。因此，每个节点也只有一个关联(绝对)路径名。

图5.11所示的命名图是**有向无环图**的一个例子。在这样的组织中，一个节点可以有多个传入边，但是不允许图形有一个环。还有一些名称空间没有这个限制。

**名称解析**

名称空间为通过名称存储和检索实体信息提供了一种方便的机制。更一般地说，给定一个路径名称，应该可以查找存储在该名称引用的节点中的任何信息。查找名称的过程称为**名称解析**。

为了解释名称解析的工作原理，让我们考虑一个路径名，比如N:[label1, label2，…labeln]。此名称的解析从命名图的节点N开始，其中在目录表中查找名称label1，并返回label1引用的节点的标识符。然后，通过在其目录表中查找名称label2继续解析所标识的节点，以此类推。假设指定的路径确实存在，那么通过返回该节点的内容，解析将在labeln引用的最后一个节点停止。

**闭包机制**

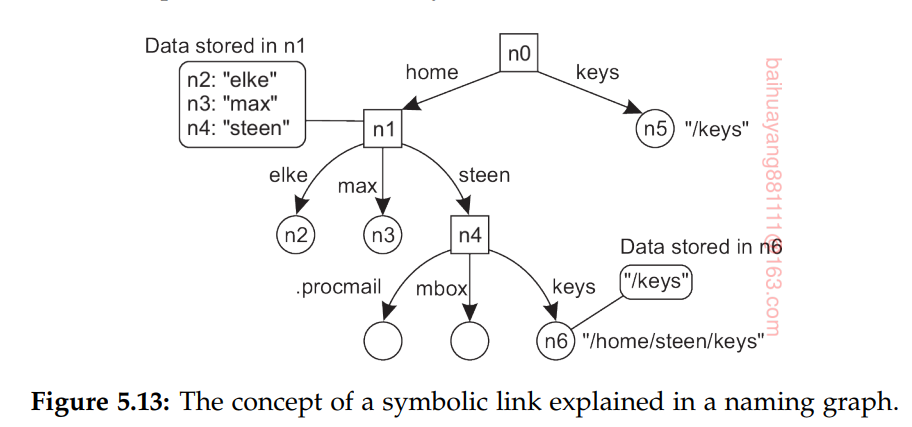
只有当我们知道如何以及从哪里开始时，名称解析才会发生。在我们的示例中，给定了起始节点，我们假设可以访问它的目录表。知道如何以及从何处开始名称解析通常被称为闭包机制。本质上，闭包机制处理的是在名称空间中选择要从其中启动名称解析的初始节点[Radia, 1989]。闭包机制有时之所以难以理解，是因为它们一定程度上是隐式的，而且在相互比较时可能非常不同。

例如，考虑字符串“00312059837784”。 许多人不知道如何处理这些数字，除非他们被告知序列是一个电话号码。这些信息足以启动解析过程，特别是通过拨打这个号码。电话系统随后完成其余的工作。

另一个例子是，考虑在分布式系统中使用全局和本地名称。本地名称的一个典型例子是环境变量。例如，在Unix系统中，名为HOME的变量用于引用用户的主目录。每个用户都有自己的这个变量的副本，它被初始化为与用户的主目录对应的全局、系统范围的名称。与环境变量关联的闭包机制确保通过在特定于用户的表中查找变量名来正确解析变量名。

**Linking and mounting**

与名称解析密切相关的是**别名**的使用。别名是同一实体的另一个名称。环境变量是别名的一个例子。在命名图方面，实现别名基本上有两种不同的方法。第一种方法是简单地允许多个绝对路径名称引用命名图中的同一个节点。这种方法如图5.13所示，其中节点n5可以由两个不同的路径名引用。在Unix术语中，图5.11中的路径名/key和/home/ stein /key都称为到节点n5的硬链接。



第二种方法是用叶子节点(比如N)表示一个实体，但是节点不存储该实体的地址或状态，而是存储一个绝对路径名。当第一次解析N节点时，名称解析将返回N存储的路径名称，之后继续解析新的路径名称。这个原则对应于Unix文件系统中符号链接的使用，如图5.13所示。在本例中，路径名/home/steen/keys是指向节点n5的符号链接，它引用一个包含绝对路径名/key的节点。

到目前为止所描述的名称解析完全在单个名称空间中进行。但是，名称解析也可以用于以透明的方式合并不同的名称空间。让我们首先考虑挂载的文件系统。在我们的命名模型中，挂载的文件系统对应于让目录节点存储来自不同名称空间的目录节点标识符，我们将其称为外部名称空间。存储节点标识符的目录节点称为挂载点。因此，外名称空间中的目录节点称为挂载点。通常，挂载点是名称空间的根。在名称解析期间，将查找挂载点并通过访问其目录表进行解析。

挂载的原理也可以推广到其他名称空间。别需要的是一个目录节点，它充当一个挂载点，并存储标识和访问外名称空间中的挂载点所需的所有信息。在许多分布式文件系统中都遵循这种方法。

考虑分布在不同机器上的名称空间集合。特别是，每个名称空间由不同的服务器实现，每个服务器可能运行在单独的机器上。因此，如果我们想将一个外部名称空间NS2挂载到一个名称空间NS1中，可能需要通过网络与NS2的服务器通信，因为该服务器可能运行在与NS1的服务器不同的机器上。要在分布式系统中挂载外部名称空间，至少需要以下信息：

1. 访问协议的名称。

2. 服务器的名称。

3.外名称空间中挂载点的名称。

注意，这些名称都需要解析。访问协议的名称需要解析为协议的实现，通过该协议可以与外部名称空间的服务器进行通信。需要将服务器的名称解析为可达的地址。作为名称解析的最后一部分，需要将挂载点的名称解析为外名称空间中的节点标识符。

在非分布式系统中，这三点实际上都不需要。例如，在Unix中没有访问协议和服务器。此外，挂载点的名称是不必要的，因为它只是外部名称空间的根目录。

挂载点的名称将由外部名称空间的服务器解析。但是，我们还需要访问协议和服务器名的名称空间和实现。一种可能性是将上面列出的三个名称表示为URL。

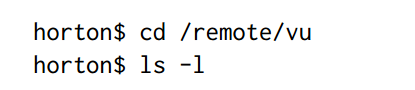
具体来说，考虑这样一种情况，即使用笔记本电脑的用户希望访问存储在远程文件服务器上的文件。客户机和文件服务器都配置了网络文件系统(NFS)。特别是，为了允许NFS在Internet上工作，客户机可以通过NFS URL确切地指定它想访问哪个文件，例如NFS: //flits.cs.vu.nl/home/steen。这个URL在NFS文件服务器flits.cs.vu上命名一个名为/home/steen的文件(恰好是一个目录)。nl，客户端可以通过NFS协议访问它[Shepler et al.， 2003]。

nfs是一个众所周知的名称，因为在如何解释这个名称方面存在着世界范围内的共识。假设我们正在处理一个URL，那么nfs名称将被解析为nfs协议的实现。使用DNS将服务器名称解析为它的地址，这将在后面的小节中讨论。如前所述，/home/steen由外部名称空间的服务器解析。

图5.14显示了客户机上文件系统的组织的一部分。根目录有许多用户定义的条目，包括一个名为/remote的子目录。这个子目录用于包含外部名称空间的挂载点，例如VU大学的用户主目录。为此，使用一个名为/remote/vu的目录节点来存储URL nfs: //flits.cs.vu.nl/home/steen。

现在考虑名称/remote/vu/mbox。通过从客户机机器上的根目录开始解析这个名称，并一直持续到到达节点/remote/vu。然后，通过返回URL nfs: // flights .cs.vu继续执行名称解析过程。然后引导客户机与文件服务器flits.cs.vu联系。nl通过NFS协议，并随后访问目录/home/steen。然后，可以通过读取该目录中名为mbox的文件继续执行名称解析，然后解析过程停止。

如前所述，允许挂载远程文件系统的分布式系统允许客户机执行以下命令。（假设客户机名为horton）：



哪个文件随后列出了远程文件服务器上目录/home/steen中的文件。这一切的美妙之处在于，用户不必考虑对远程服务器的实际访问的细节。理想情况下，与访问本地可用的文件相比，只会注意到性能上的一些损失。实际上，对客户机来说，根在本地机器上的名称空间和根在远程机器上的/home/steen上的名称空间形成了一个单独的名称空间。

**名称空间的实现**

名称空间构成命名服务的核心，即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。如果一个分布式系统被限制在一个局域网中，通常可以只通过一个名称服务器来实现一个命名服务（所有机器的名称空间都挂载到同一个服务器上？？？）。然而，在具有许多实体的大型分布式系统中(可能分布在一个较大的地理区域)，有必要将名称空间的实现分布在多个名称服务器上。

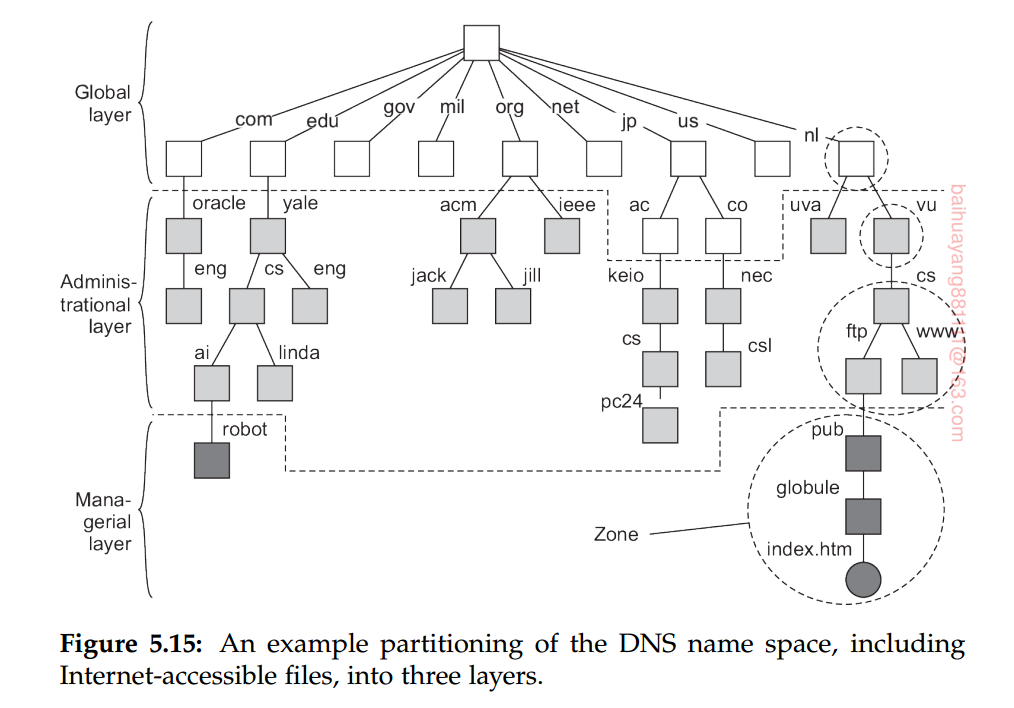
**名字空间分布**

大型(可能是全球范围的)分布式系统的名称空间通常是分层组织的。与前面一样，假设这样的名称空间只有一个根节点。为了有效地实现这样的名称空间，可以方便地将其划分为多个逻辑层。Cheriton和Mann[1989]区分了以下三个层次：

**全局层**由最高级别的节点组成，即根节点和逻辑上接近根的其他目录节点，即它的子节点。全局层中的节点通常以其稳定性为特征，即目录表很少更改。这些节点可以表示组织或组织组，他们的名称存储在命名空间中。

**权限层（或者叫功能层？）**由目录节点组成，这些节点一起在单个组织中进行管理。**权限**层中的目录节点的一个特征是，这些目录节点表示属于同一组织或管理单元的实体组。例如，一个组织中的每个部门可能都有一个目录节点，或者可以从中找到所有主机的目录节点。可以使用另一个目录节点作为命名所有用户的起点，等等（个人理解，每个节点或者目录，都表示一定的功能）。**权限**层中的节点相对稳定，尽管更改通常比全局层中的节点更频繁。

最后，**管理层**由节点组成，节点通常会定期更改。例如，表示本地网络中的主机的节点属于这一层。出于同样的原因，该层包含表示共享文件的节点，比如库或二进制文件的节点。另一类重要的节点包括那些表示用户定义的目录和文件的节点。与全局层和**权限**层不同，管理层中的节点不仅由系统管理员维护，而且由分布式系统的各个终端用户维护。



为了更加具体，图5.15显示了DNS名称空间部分分区的示例，包括可以通过Internet访问的组织内文件的名称，例如Web页面和可传输文件。名称空间被划分为不重叠的部分，在DNS中称为区域[Mockapetris, 1987]。区域是名称空间的一部分，由单独的名称服务器实现。其中一些区域如图5.15所示。

如果我们查看可用性和性能，每一层中的名称服务器都必须满足不同的需求。对于全局层中的名称服务器，高可用性尤其重要。如果命名服务器失败，将无法访问大部分命名空间，因为名称解析无法跨越故障服务器而进行。

性能方面则有点微妙。由于全局层中节点的变化率较低，所以查找操作的结果通常在很长一段时间内保持有效。因此，可以由客户端有效地缓存这些结果(即，存储在本地)。下一次执行相同的查找操作时，可以从客户机的缓存中检索结果，而不是让名称服务器返回结果。因此，全局层中的名称服务器不必快速响应单个查询请求。另一方面，吞吐量可能很重要，特别是在拥有数百万用户的大型系统中。

通过复制服务器和客户端缓存，可以满足全局层中名称服务器的可用性和性能需求。这一层的更新通常不需要立即生效，从而更容易保持副本的一致性。

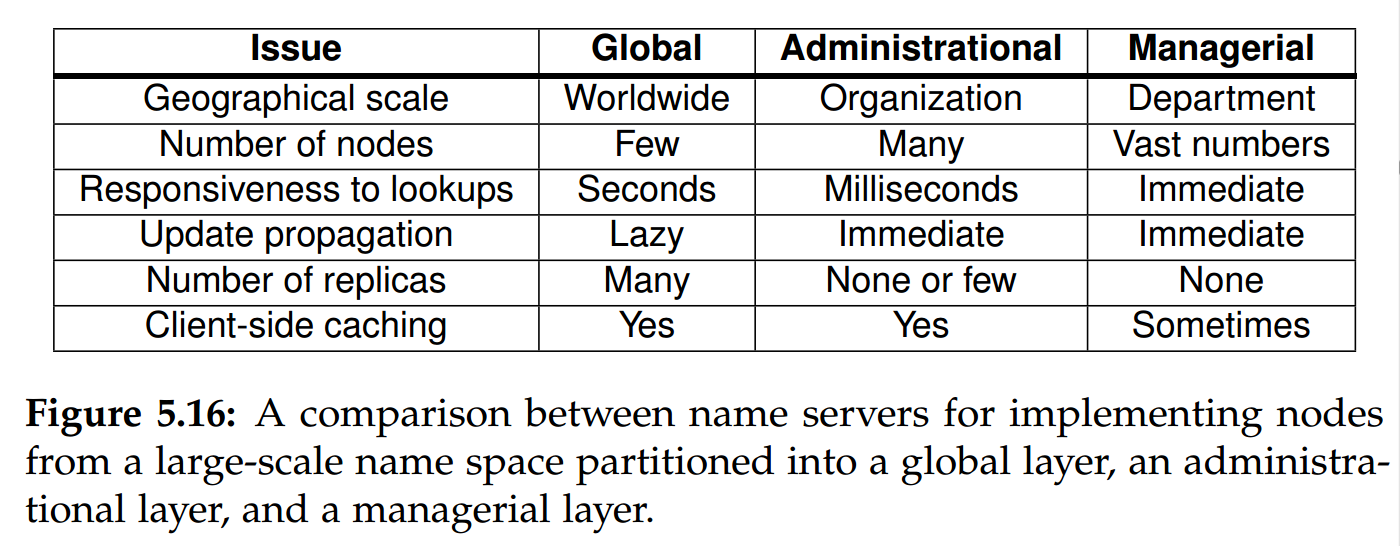
管理层中名称服务器的可用性对于与名称服务器位于同一组织中的客户端来说非常重要。如果名称服务器失败，组织中的许多资源将无法访问，因为它们无法被查找。另一方面，组织内的资源对组织外的用户暂时不可访问可能不那么重要。

在性能方面，权限层中的名称服务器与全局层中的名称服务器具有类似的特性。由于对节点的更改并不经常发生，缓存查找结果可能非常有效，从而降低了性能的重要性。然而，与全局层相反，权限层应该注意在几毫秒内返回查找结果，要么直接从服务器返回，要么从客户机的本地缓存返回。同样，更新通常应该比全局层的更新处理得更快。例如，新用户的帐户需要几个小时才能生效，这是不可接受的。

这些需求通常可以通过使用功能相对强大的机器来运行名称服务器来满足。此外，应该应用客户端缓存，并结合复制以提高整体可用性。

管理级别对名称服务器的可用性要求通常较低。特别是，通常使用一台机器来运行名称服务器就足够了，这可能会导致暂时不可用。然而，性能是至关重要的:操作必须立即进行。由于更新经常发生，客户端缓存的效率通常较低。

图5.16展示了这几层命名空间的差异。在分布式系统中，全局层和权限层中的名称服务器是最难实现的。困难是由复制和缓存造成的，这是可用性和性能所必需的，但也会带来一致性问题。由于缓存和副本分布在广域网上，这可能会在查找过程中引入长时间的通信延迟，这使得一些问题更加严重。

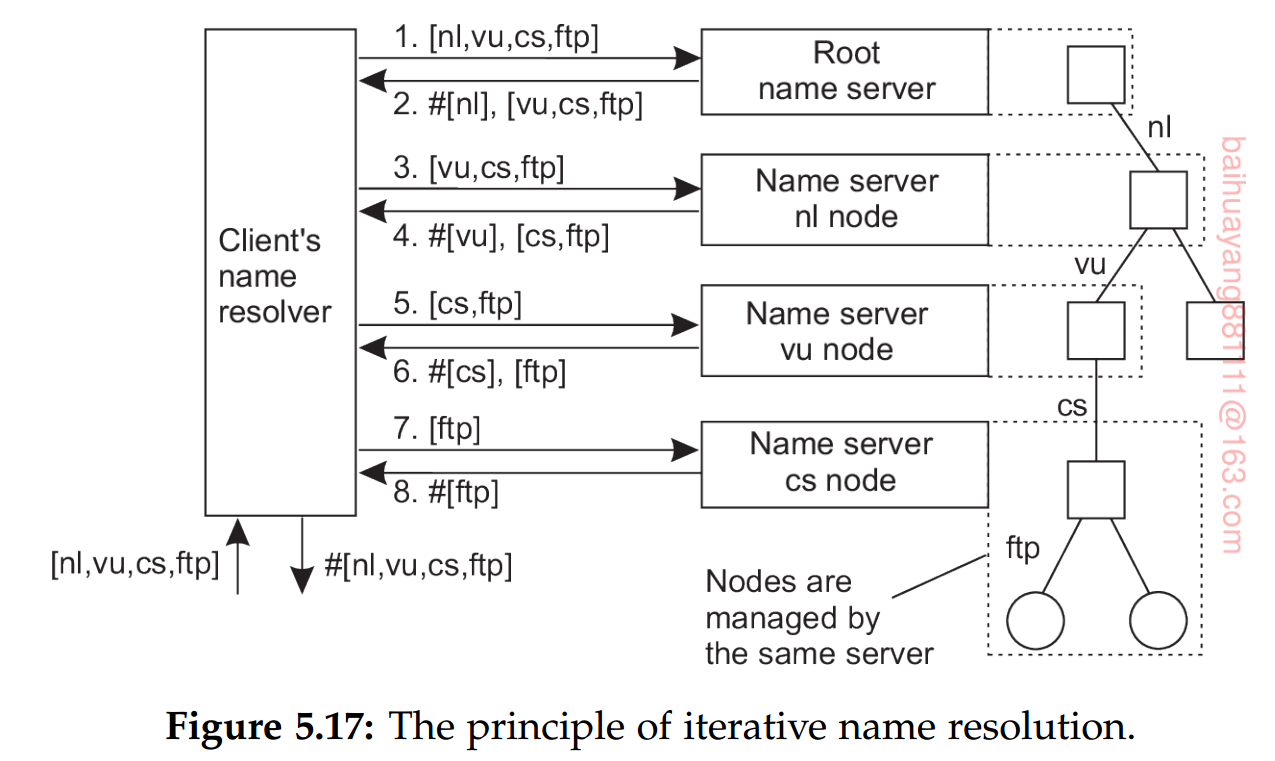


**名称解析的实现**

跨多个名称服务器的名称空间分布会影响名称解析的实现。为了解释大型名称服务中名称解析的实现，我们暂时假设没有复制名称服务器，也没有使用客户端缓存。每个客户机都可以访问本地名称解析器，该解析器负责确保执行名称解析过程。参照图5.15，假设(绝对)路径名root:[nl,vu,cs,ftp,pub,globe,index]将被解析。使用URL符号，这个路径名将对应于ftp: //ftp.cs.vu.nl/pub/globe/index.html。现在有两种方法可以实现名称解析。

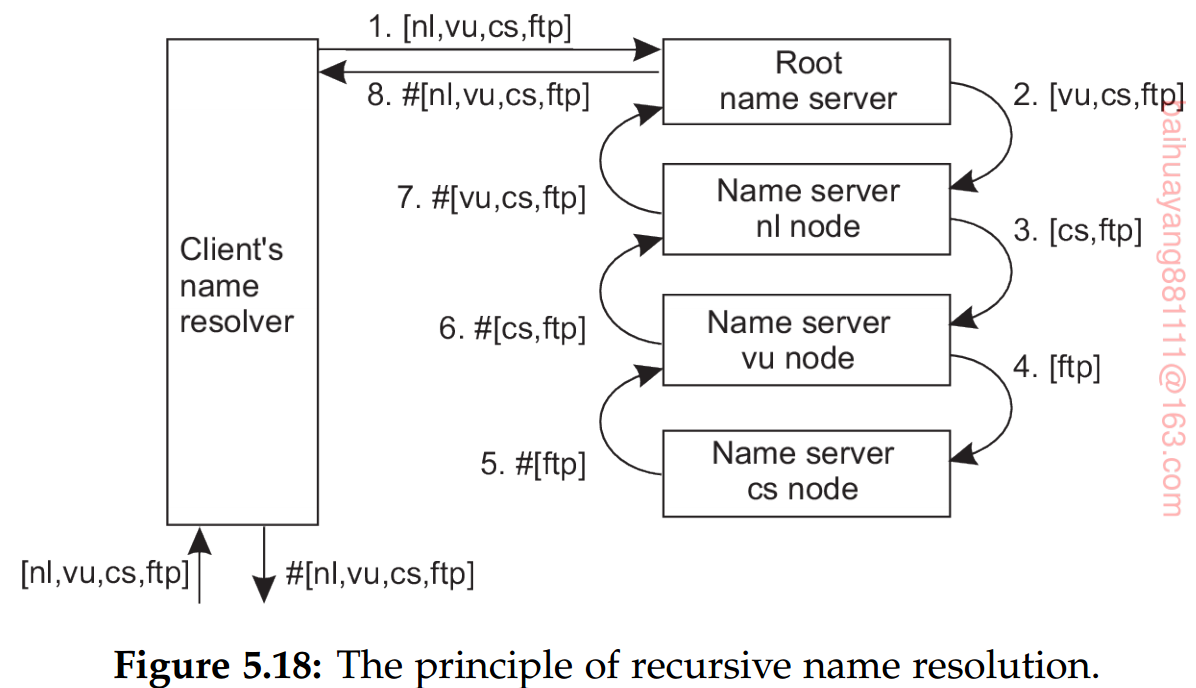
在迭代名称解析中，名称解析器将完整的名称传递给根名称服务器。假定可以联系根服务器的地址是已知的。根服务器将尽可能解析路径名，并将结果返回给客户机。在我们的示例中，根服务器只能解析标签nl，接着它将返回相关名称服务器的地址。

此时，客户机传递剩余的路径名(即。， nl:[vu,cs,ftp, pub,globe,index.html])到该名称服务器。此服务器只能解析标签vu，并返回关联名称服务器的地址，以及剩余的路径名vu:[cs,ftp,pub,globe,index.html]。然后，客户机的名称解析器将与下一个名称服务器响应是解析标签cs，然后解析ftp，返回ftp服务器的地址和路径名ftp:[pub, globe, index.html]。然后客户机与FTP服务器联系，请求它解析原始路径名称的最后一部分。FTP服务器随后将解析标签pub、globe和index。html，并传输请求的文件(在本例中使用FTP)。这个迭代名称解析过程如图5.17所示。



实际上，最后一步是联系FTP服务器，并请求它传输路径名为FTP:[pub, globe, index]的文件。，由客户端进程单独执行。换句话说，客户端通常只将路径名root:[nl, vu, cs, ftp]传递给名称解析器，它期望从该解析器获得可以联系ftp服务器的地址，如图5.17所示。

迭代名称解析的另一种方法是在名称解析期间使用递归。名称服务器不是使用递归名称解析器将每个中间结果返回给客户机的名称解析器，而是将结果传递给它找到的下一个名称服务器（就是不返回给客户端了？）。例如，当根名称服务器找到实现节点nl的名称服务器的地址时，它请求该名称服务器解析路径名称nl:[vu, cs, ftp, pub, globe, index.html]。下一个服务器还将使用递归名称解析器解析完整的路径，并最终将文件index.html返回给根服务器，根服务器又将该文件传递给客户机的名称解析器。递归名称解析如图5.18所示。与迭代名称解析一样，最后一步(联系FTP服务器并要求它传输指定的文件)通常由客户机作为单独的进程执行。



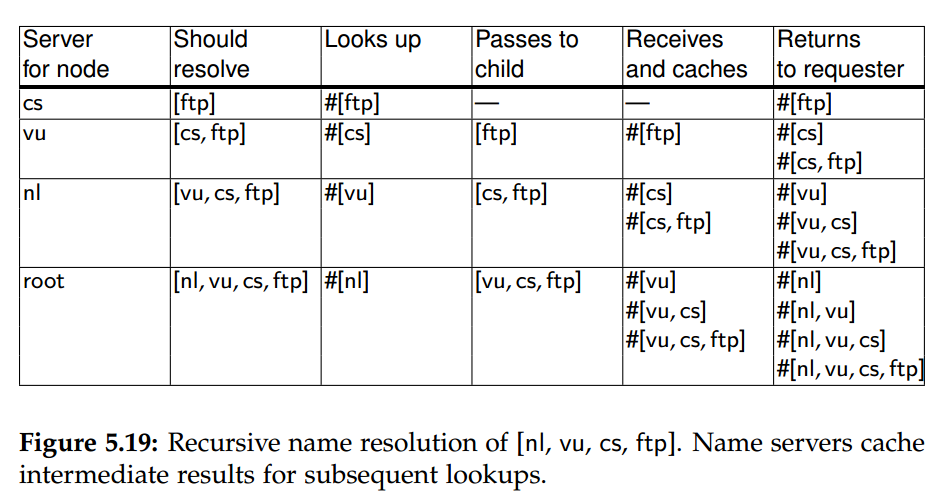
递归名称解析的主要缺点是对每个名称服务器的性能要求更高。基本上，虽然可以与其他的服务器协作，然而一个命名服务器还是需要自己完整的解析路径(第一个方法不是嘛？)。这种额外的负担通常非常高，以至于名称空间的全局层中的名称服务器只支持迭代名称解析。

递归名称解析有两个重要的优点。第一个优点是，与迭代名称解析相比，缓存结果更有效。第二个好处是通信成本可能会降低。为了解释这些优点，假设客户机的名称解析器只接受只引用名称空间的全局层或权限层中的节点的路径名称。要解析与权限层中的节点对应的路径名称的那一部分，客户机将分别与由名称解析器返回的名称服务器联系，如上所述。

递归名称解析允许每个名称服务器逐步了解负责实现低级节点的每个名称服务器的地址。因此，缓存可以有效地用于提高性能。例如，当根服务器被请求解析路径名root:[nl,vu,cs,ftp]时，它最终将获得实现该路径名引用的节点的名称服务器的地址。为此，nl节点的名称服务器必须查找vu节点的名称服务器地址，而vu节点必须查找处理cs节点的名称服务器地址。

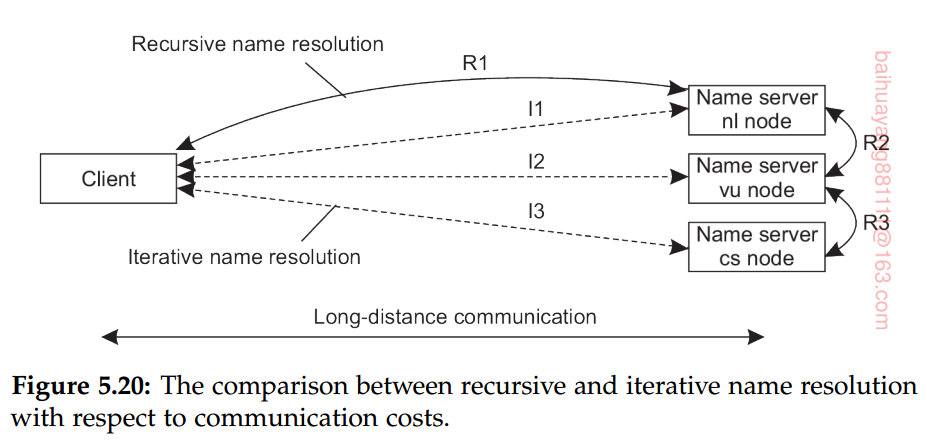
由于全局和管理层中节点的更改并不经常发生，所以根名称服务器可以有效地缓存返回的地址。此外，由于地址也通过递归返回给负责实现vu节点的名称服务器和实现nl节点的名称服务器，所以它也可以缓存在这些服务器上。（就是各个节点都会有缓存，而且根节点也会通过这种递归调用，获得缓存，而且是完整路径的缓存，这在迭代是无法实现的）

同样，中间名称查找的结果也可以返回并缓存。例如，nl节点的服务器必须查找vu节点服务器的地址。当nl服务器返回原始名称查找的结果时，可以将该地址返回给根服务器。解析过程的完整概述，以及每个名称服务器可以缓存的结果，如图5.19所示。



这种方法的主要好处是，最终可以非常有效地处理查找操作。例如，假设另一个客户机稍后请求解析路径名根:[nl, vu, cs, flits]。这个名称被传递给根，根可以立即将它转发到cs节点的名称服务器，并请求它解析剩余的路径名称cs:[flits]。对于迭代名称解析，缓存必须限于客户机的名称解析器。因此，如果客户机a请求解析名称，而另一个客户机B稍后请求解析相同的名称，则名称解析必须通过与客户机a相同的名称服务器。作为一种折衷方案，许多组织使用由所有客户机共享的本地中间名称服务器。此本地名称服务器处理所有命名请求并缓存结果（必须这样，因为所有的解析结果都只能缓存在一台客户机上）。从管理的角度来看，这样的中间服务器也很方便。例如，只有该服务器需要知道根名称服务器位于何处;其他机器不需要此信息。

递归名称解析的第二个优点是，相对于通信，它通常更便宜（减少了很多和客户端的通信）。同样，考虑路径名root:[nl, vu, cs, ftp]的解析，并假设客户机位于旧金山。假设客户机知道nl节点的服务器地址，并使用递归名称解析，通信遵循从客户机位于旧金山的主机到位于荷兰的nl服务器的路由，称此次通信为R1，如图5.20所示。从那时起，nl服务器和位于荷兰阿姆斯特丹校园内的VU大学的名称服务器之间就需要通信了，称此次通信为R2。最后，需要vu服务器与计算机科学系的名称服务器通信，如R3所示。回复的路径是相同的，但是方向相反。显然，通信成本由客户机主机和nl服务器之间的消息交换决定。相反，对于迭代名称解析，客户机的主机必须分别与nl服务器、vu服务器和cs服务器通信，其中总成本大约是递归名称解析的三倍。图5.20中标记为I1、I2和I3的箭头显示了迭代名称解析的通信路径。



**例子 域名系统**

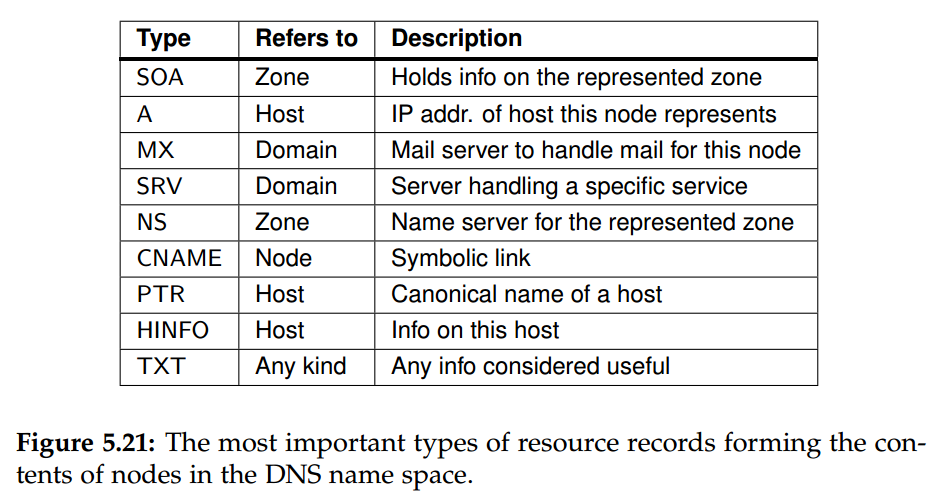
目前使用的最大的分布式命名服务之一是Internet域名系统(DNS)。DNS主要用于查找主机和邮件服务器的IP地址。在接下来的几页中，我们将重点介绍DNS名称空间的组织及其节点中存储的信息。此外，我们还将进一步研究DNS的实际实现。更多信息可以在[Mockapetris, 1987]和[Liu and Albitz, 2006]中找到。DNS的评估，特别是关于它是否仍然适合当前互联网的需要，可以在[Levien, 2005]中找到。从这份报告中，我们可以得出一个有些令人惊讶的结论，即即使在30多年后，DNS也没有显示出需要替换它的迹象。我们认为，主要原因在于设计师对如何让事情变得简单有着深刻的理解。在分布式系统的其他领域的实践表明，没有多少人具有这样的理解天赋。

**DNS名称空间**

DNS名称空间按层次结构组织为一棵树。标签是由字母数字字符组成的不区分大小写的字符串。标签的最大长度为63个字符;完整路径名的长度限制为255个字符。路径名的字符串表示形式包括列出它的标签，从最右边的标签开始，并用点(.)分隔标签。根就用点表示。例如，路径名root:[nl,vu,cs,flits]由字符串“flits.cs.vu.nl”表示。，其中包括表示根节点的最右边的点。为了可读性，我们通常省略这个点。

因为DNS名称空间中的每个节点只有一个传入边缘(根节点除外，根节点没有传入边缘)，所以附加到节点传入边缘的标签也用作该节点的名称。对于根节点来说，子树被称作**域**路径，也叫做**域名**。注意，就像路径名一样，域名可以是绝对的，也可以是相对的。

节点的内容由一组资源记录组成。这里有不同类型的资源记录。主要的如图5.21所示：



DNS名称空间中的节点通常同时表示多个实体。例如，一个域名，如vu.nl用于表示domain（域）和zone。在这种情况下，域是通过几个(不重叠的)区域实现的。SOA(起始权限)资源记录包含一些信息，比如负责所表示区域的系统管理员的电子邮件地址、可以获取该区域数据的主机的名称，等等。

A(地址)记录，表示Internet中的特定主机。A记录包含该主机用来通信的IP地址。如果主机有多个IP地址，就像多主机的情况一样，节点将包含每个地址的a记录。

另一种类型的记录是MX(邮件交换)记录，它类似于到表示邮件服务器的节点的符号链接。例如表示域cs.vu.nl的节点，有一个包含MX记录的地址为zephyr.cs.vu.nl的邮件服务器。这个服务器将处理发送给cs.vu.nl域的用户的邮件。一个节点中可能存储有多个MX记录。

与MX记录相关的是SRV记录，其中包含特定服务的服务器名。服务本身通过名称和协议名称来标识。例如，cs.vu.nl中的Web服务器可以通过SRV记录来命名，比如\_http\_tcp.cs.vu.nl。然后，该记录将引用服务器的实际名称。SRV记录的一个重要优点是，客户机不再需要知道提供特定服务的主机的DNS名称。相反，只需要标准化服务名称，然后就可以查找提供的主机。

表示区域的节点，包含一个或多个NS(名称服务器)记录。与MX记录一样，NS记录包含实现由节点表示的区域的名称服务器的名称。原则上，名称空间中的每个节点都可以存储一个引用实现它的名称服务器的NS记录。但是，正如我们下面讨论的，DNS名称空间的实现是这样的，只有表示区域的节点才需要存储NS记录。

DNS将别名与所谓的**规范名称**区分开来。假定每个主机都有一个规范名称或主要名称。别名是通过节点存储包含主机规范名称的CNAME记录来实现的。因此，存储这样一条记录的节点的名称与符号链接相同，如图5.13所示。

DNS通过PTR(指针)记录维护IP地址到主机名的反向映射。为了适应只提供IP地址时主机名的查找，DNS维护一个名为in-addr的域，arpa，它包含表示Internet主机的节点，这些节点由所表示主机的IP地址命名。例如，主机www.cs.vu.nl的IP地址为130.37.20.20。DNS创建一个名为20.20.37.130.in-addr的节点。arpa，用于存储该主机的规范名称(sol .cs.vu.nl在PTR记录中)。

最后，HINFO (host info)记录用于在主机上存储附加信息，比如主机的机器类型和操作系统。以类似的方式，TXT记录用于用户认为对存储由节点表示的实体有用的任何其他类型的数据。

**DNS实现**

实质上，DNS名称空间可以分为全局层和权限层，如图5.15所示。管理层通常由本地文件系统组成，形式上不属于DNS的一部分，因此也不受DNS管理。每个区域都由一个名称服务器实现，为了可用性，它总是被复制为多个。区域的更新通常由主名称服务器处理。更新是通过修改主服务器本地的DNS数据库进行的。辅助名称服务器不直接访问数据库，而是请求主服务器传输其内容。后者在DNS术语中称为**区域传输**。

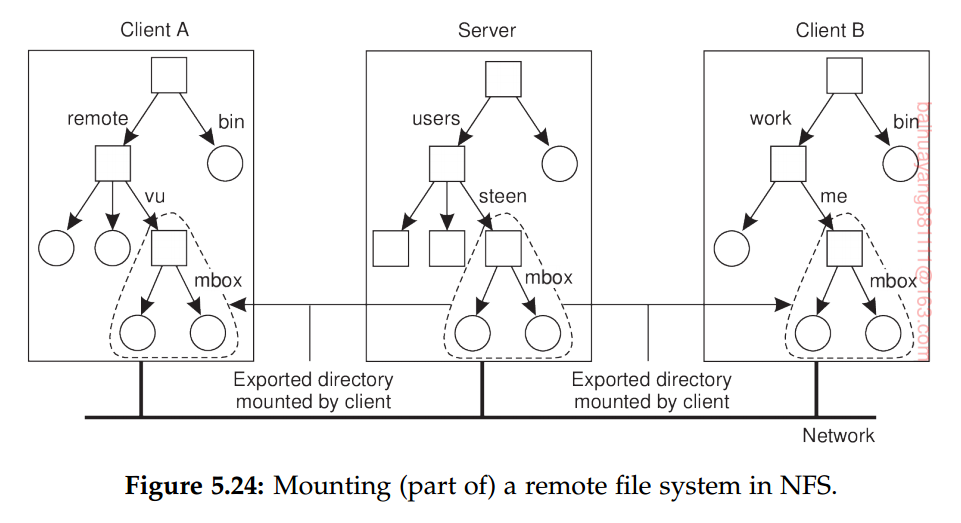
DNS数据库实现为(小)文件集合，其中最重要的一个包含特定区域内所有节点的所有资源记录。这种方法允许仅通过节点的域名来标识节点，通过域名，节点标识符的概念简化为(隐式的)文件索引。

（从DNS名称空间 到 DNS实现，有点懵比）

（看note，总结起来，就是每个服务节点，都会包含若干个信息，SOA ，A地址，MX等信息。这些信息表示着这个节点应该提供的功能。）

**例子：网络文件系统(NFS)**

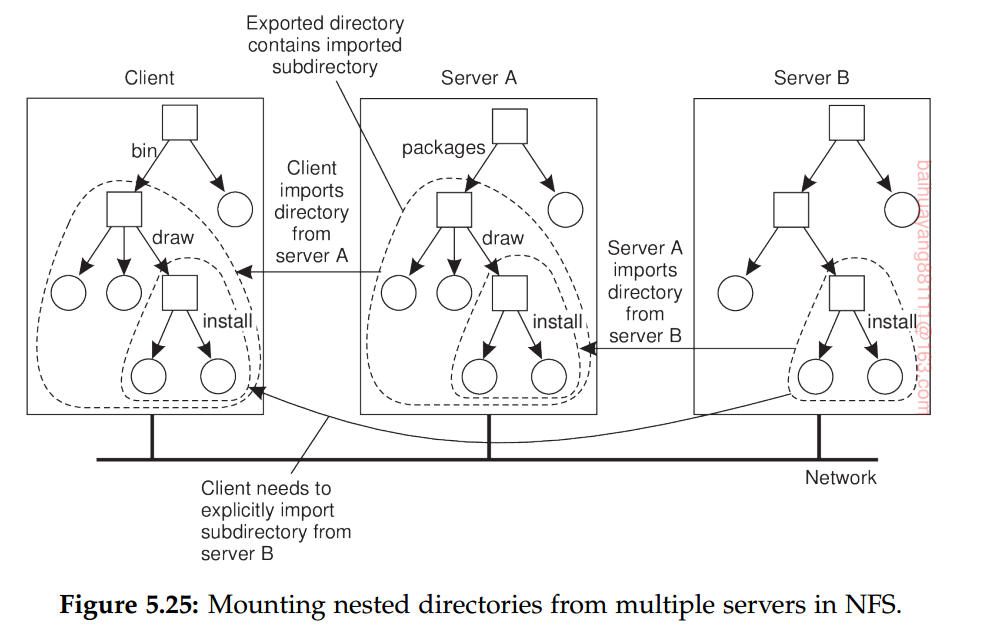
作为另一个非常不同的示例，考虑使用NFS进行命名。NFS命名模型的基本思想是为客户机提供对由服务器维护的远程文件系统的完全透明访问。



NFS不允许挂载整个文件系统，而只允许客户端挂载文件系统的一部分，如图5.24所示。导出的目录可以挂载到客户机的本地名称空间中。这种设计方法有一个严重的含义:原则上，用户不共享名称空间。如图5.24所示，客户机A上名为/remote/vu/mbox的文件在客户机B上名为/work/me/mbox。因此，文件的名称取决于客户机如何组织自己的本地名称空间，以及导出的目录挂载在何处。这种方法在分布式文件系统中的缺点是共享文件变得更加困难。例如，Alice不能使用分配给该文件的名称告诉Bob关于该文件的信息，因为该名称在Bob的文件名称空间中可能具有完全不同的含义。

有几种方法可以解决这个问题，但最常见的一种方法是为每个客户机提供一个部分标准化的名称空间。例如，每个客户机可能使用本地目录/usr/bin来挂载一个文件系统，其中包含对每个人都可用的程序的标准集合。同样，目录/local可以用作一个标准来挂载位于客户机主机上的本地文件系统。

NFS服务器本身可以挂载其他服务器导出的目录。但是，它不允许将这些目录导出到自己的客户机。相反，客户机必须显式地从维护目录的服务器挂载这样一个目录，如图5.25所示。这种限制部分来自于简单性。如果服务器可以导出它从另一台服务器上挂载的目录，那么它必须返回包含服务器标识符的特殊文件句柄。NFS不支持这样的文件句柄。



为了更详细地解释这一点，假设服务器A承载一个文件系统FSA，并从中导出目录/包。这个目录包含一个子目录/draw，它充当由服务器B导出并由a挂载的文件系统FSB的挂载点。让A也导出/packages/draw到它自己的客户机，并假设客户机已经将/packages挂载到它的本地目录/bin中，如图5.25所示。

如果名称解析是迭代的，那么要解析名称/bin/draw/install，客户机将在本地解析/bin并请求A返回目录/draw的文件句柄时与服务器A联系。在这种情况下，服务器A应该返回一个包含服务器B标识符的文件句柄，因为在本例/install中，只有B可以解析路径名的其余部分。正如我们所说，NFS不支持这种名称解析。（是说NFS不允许这种多重挂载，还是说第二层挂载必须在B中的目录进行）

在NFS的早期版本中，名称解析是严格迭代的，因为一次只能查找一个文件名。换句话说，解析/bin/draw/install这样的名称需要对NFS服务器进行三个单独的调用。此外，客户机完全负责实现路径名称的解析。NFSv4还支持递归名称查找。在这种情况下，客户机可以将完整的路径名传递给服务器，并请求该服务器解析它。

NFS名称查找的另一个特性在最新版本(NFSv4)中得到了解决。考虑一个文件服务器承载多个文件系统。使用严格的迭代名称解析，每当对挂载另一个文件系统的目录执行查找时，查找将返回该目录的文件句柄。随后读取该目录将返回其原始内容，而不是挂载文件系统的根目录。（看下面的例子，返回原始就是返回A的目录，这里面将没有install，install是在B中的）

为了解释，假设在前面的示例中，文件系统FSA和FSB都由一台服务器托管。如果客户机已将/包挂载到其本地目录/bin中，那么在服务器上查找文件名draw将返回draw的文件句柄。随后对服务器的调用将通过readdir列出draw的目录条目，然后返回最初存储在FSA子目录/packages/draw中的目录条目列表。只有当客户机也挂载了文件系统FSB，才有可能正确地解析相对于/bin的路径名draw/install。

NFSv4通过允许查找跨服务器上的挂载点解决了这个问题。特别是，lookup返回**挂载目录**的文件句柄，而不是**原始目录**的文件句柄。客户端可以通过检查所查找文件的文件系统标识符来检测查找是否越过了挂载点。如果需要，客户机也可以在本地挂载该文件系统。

文件句柄是对文件系统中的文件的引用。它独立于它所引用的文件的名称。文件句柄由承载文件系统的服务器创建，并且对于服务器导出的所有文件系统是惟一的。它是在创建文件时创建的。客户端不知道文件句柄的实际内容;完全不透明。文件句柄在NFS版本2中是32字节，但是在版本3中是64字节，在版本4中是128字节。当然，文件句柄的长度不是不透明的。

理想情况下，文件句柄被实现为相对于文件系统的文件的真实标识符。首先，这意味着只要文件存在，它就应该有一个相同的文件句柄。这种持久性要求允许客户端在通过其名称查找相关文件之后本地存储文件句柄。一个好处是性能:由于大多数文件操作需要文件句柄而不是名称，客户端可以避免在每次文件操作之前重复查找名称。这种方法的另一个好处是，客户端现在可以访问文件，而不管它的(当前)名称是什么。

因为文件句柄可以由客户机本地存储，所以服务器在删除文件后不重用文件句柄也很重要。否则，客户端在使用本地存储的文件句柄时，可能会错误地访问错误的文件。请注意，迭代名称查找和不允许查找跨越挂载点的操作的组合会带来获取初始文件句柄的问题。为了访问远程文件系统中的文件，客户机将需要向服务器提供应该执行查找的目录的文件句柄，以及要解析的文件或目录的名称。NFSv3通过一个单独的挂载协议解决了这个问题，通过该协议，客户机实际上挂载一个远程文件系统。在挂载之后，客户机被传回挂载的文件系统的根文件句柄，它随后可以将根文件句柄用作查找名称的起点。

在NFSv4中，通过提供一个单独的putrootfh操作来解决这个问题，该操作告诉服务器解决与它管理的文件系统的根文件句柄相关的所有文件名。根文件句柄可用于查找服务器文件系统中的任何其他文件句柄。这种方法还有一个额外的好处，即不需要单独的挂载协议。相反，可以将挂载集成到用于查找文件的常规协议中。客户端只需使用putrootfh请求服务器解析与文件系统根文件句柄相关的名称，就可以挂载远程文件系统。

**5.4属性命名**

平面和结构化名称通常提供一种独特的、与位置无关的引用实体的方法。此外，结构化名称的部分设计目的是提供一种对人友好的方式来命名实体，以便方便地访问它们。在大多数情况下，假定名称只引用一个实体。然而，位置独立性和人性化并不是命名实体的唯一标准。

特别是，随着越来越多的信息可用，有效地搜索实体变得非常重要。这种方法要求用户仅仅提供他正在寻找的内容的描述即可。

提供描述的方法有很多，但是在分布式系统中，一种比较流行的方法是使用(属性、值)对来描述实体，通常称为基于属性的命名。在这种方法中，假定一个实体具有相关的属性集合。每个属性都对这个实体进行了描述。通过指定特定属性应该具有哪些值，用户实际上限制了他感兴趣的实体集。由命名系统返回一个或多个符合用户描述的实体。在本节中，我们将进一步研究基于属性的命名系统。

**目录服务**

基于属性的命名系统也称为目录服务，而支持结构化命名的系统通常称为命名系统。对于目录服务，实体具有一组可用于搜索的关联属性。在某些情况下，属性的选择可能相对简单。例如，在电子邮件系统中，可以使用发件人、收件人、主题等属性标记消息。然而，即使是在电子邮件的情况下，当需要其他类型的描述符时，事情就变得困难了，正如开发只允许某些消息(基于它们的描述符)通过的过滤器的实现。

归根结底，设计一组合适的属性并不简单。在大多数情况下，属性设计必须手工完成。即使对要使用的属性集有共识，实践表明，由不同的人群一致地设置值本身就是一个问题，就像许多人在访问Internet上的音乐和视频数据库时所经历的那样。

为了缓解其中的一些问题，对统一描述资源的方法进行了研究。在分布式系统上下文中，一个特别相关的开发是资源描述框架(RDF)。RDF模型的基础是将资源描述为由一个主题、一个谓词和一个对象组成的三元组。例如，(Person, name, Alice)描述了一个名为Person的资源，该资源的名称是Alice。在RDF中，每个主题、谓词或对象都可以是资源本身。这意味着可以将Alice实现为对一个文件的引用，该文件随后可以检索。对于谓词，这样的资源可以包含该谓词的文本描述。与主题和对象关联的资源可以是任何东西。RDF中的引用本质上是url。

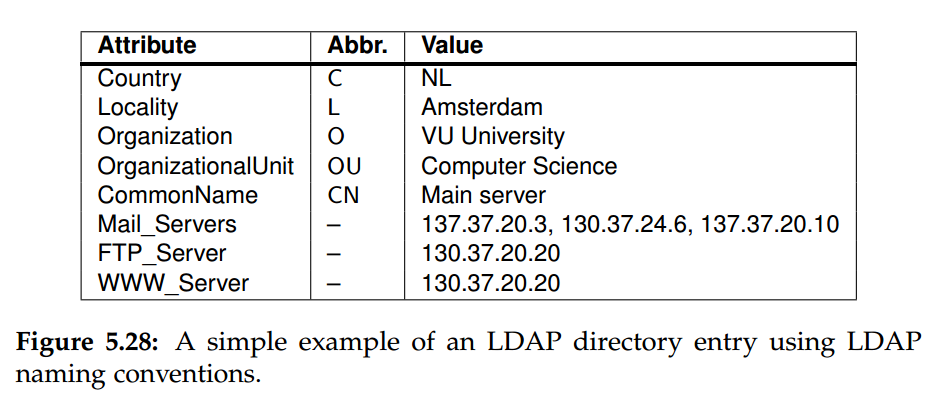
如果存储了资源描述，就有可能以许多基于属性的命名系统的方式查询该存储。例如，应用程序可以询问与名为Alice的人相关的信息。这样的查询将返回对与Alice关联的person资源的引用。然后，应用程序可以获取此资源。

在本例中，资源描述存储在一个中心位置。没有理由让资源也应该驻留在相同的位置。（描述资源单独放在一个服务器，与客户和服务端都不在一个地方）。然而，没有将描述放在相同的位置可能会导致严重的性能问题。与结构化命名系统不同，在基于属性的命名系统中查找值本质上需要对所有描述符进行彻底的搜索。(可以使用各种技术来避免这种彻底的搜索，其中一个明显的方法就是索引。) 在考虑性能时，在单个非分布式数据存储中进行彻底的搜索可能没有那么大的问题，但是仅仅向联合实现分布式数据存储的数百台服务器发送搜索查询通常不是一个好主意。在下面的文章中，我们将研究在分布式系统中解决这个问题的不同方法。

**分层实现:LDAP（这么眼熟呢！）**

处理分布式目录服务的一种常见方法是将结构化命名与基于属性的命名相结合。这种方法已经被广泛采用，例如，在微软的Active Directory服务和其他系统中。其中许多系统使用或依赖轻量级目录访问协议(通常简称LDAP)。LDAP目录服务是从OSI的X.500目录服务派生出来的。与许多OSI服务一样，它们相关实现的质量阻碍了其广泛使用，需要进行简化才能使其有用。

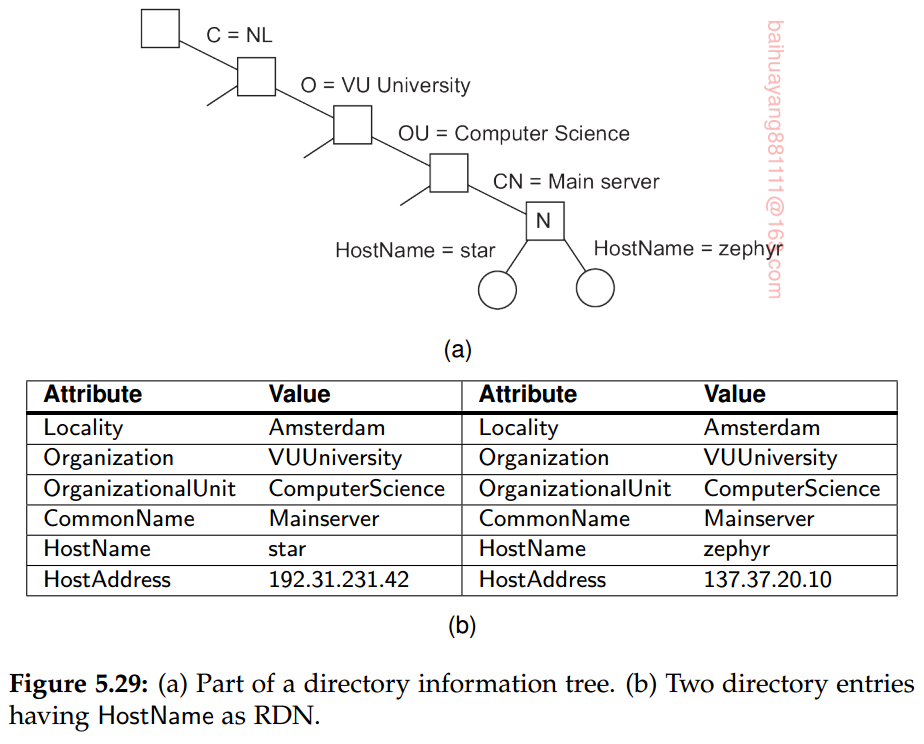
从概念上讲，LDAP目录服务由许多记录组成，通常称为目录条目。目录项相当于DNS中的资源记录。每个记录由一组(属性、值)对组成，其中每个属性都有一个关联的类型。单值属性和多值属性也是区分开来的。多值属性通常表示数组和列表。例如，图5.23中的一个简单目录条目标识了一些通用服务器的网络地址，如图5.28所示。



在我们的示例中，我们使用了LDAP标准中描述的命名约定，它适用于前五个属性。属性Organization和OrganizationUnit分别描述与存储与记录中的数据相关联的组织和部门。同样，属性Locality和Country提供关于条目存储位置的附加信息。CommonName属性通常用作(模糊的)名称，用于标识目录中有限部分的条目。例如，如果给定其他四个属性的特定值(Country、Locality、Organization和OrganizationalUnit)，名称“Main server”可能足以找到我们的示例条目。在我们的示例中，只有属性Mail\_Servers具有多个与之关联的值。所有其他属性只有一个值。

LDAP目录服务中所有目录条目的集合称为**目录信息库(DIB)**。DIB的一个重要方面是，每条记录都有唯一的名称，这样就可以查找它。这样一个全局惟一的名称作为命名属性序列出现在每个记录中。每个命名属性都称为**相对专有名称**，简称**RDN**。在图5.28中的示例中，前五个属性都是命名属性。使用表示LDAP中命名属性的传统缩写，如图5.28所示，属性Country、Organization和OrganizationalUnit可用于形成全局惟一名称/C = NL/O = VU University/OU = Computer Science。类似于DNS名称nl.vu.cs。

在DNS中，通过按顺序列出RDNs来使用全局惟一名称，构成了目录条目集合的层次结构，它被称为**目录信息树(DIT)**。DIT实际上形成LDAP目录服务的命名图，其中每个节点表示一个目录条目。此外，节点还可以充当传统意义上的目录，因为节点可以作为多个子节点的父节点。要解释这一点，请考虑图5.29中部分显示的命名图。(请记住标签与边关联)。



节点N对应于图5.28中所示的目录条目。同时，这个节点充当许多其他目录条目的父节点，这些目录条目具有作为RDN使用的附加命名属性HostName。例如，这些条目可以用来表示主机，如图5.29所示。（说白了，就是属性组成一个目录信息树，就跟mysql的联合索引一样）。

因此，LDAP命名图中的节点可以同时表示传统意义上的目录(如前所述)和LDAP记录。这一区别由两个不同的查找操作支持。read操作用于读取一条给定路径名的记录。相反，list操作用于列出DIT中给定节点的所有输出边缘的名称。每个名称对应于给定节点的子节点。注意，列表操作不返回任何记录;它只返回名称。换句话说，如图5.29这样的路径，最后将会返回 start 和 zephyr。

实现LDAP目录服务的过程与实现命名服务(如DNS)的过程非常相似，只是LDAP支持更多的查找操作，我们稍后将对此进行讨论。在处理大型目录时，DIT通常被分区并分布在多个服务器上，称为**目录服务代理(DSA)**。因此，分区DIT的每个部分都对应DNS中的一个区域。同样，每个DSA的行为与普通名称服务器非常相似，只是它实现了一些典型的目录服务，比如高级搜索操作。

客户端由目录**用户代理**(简称**DUA**)表示。DUA类似于结构化命名服务中的名称解析器。DUA根据标准化的访问协议与DSA交换信息。

LDAP实现与DNS实现的不同之处在于通过DIB进行搜索的功能。特别是，提供了搜索目录条目的工具，这些工具提供了一组搜索条目的属性应该满足的条件。例如，假设我们想要一个VU大学所有主服务器的列表。使用Howes[1997]中定义的符号，可以使用类似于这样的搜索操作返回这样的列表。



在本例中，我们指定了查找主服务器的位置是国家NL中名为VU\_University的组织，但是我们对特定的OrganizationUnit不感兴趣。但是，每个返回的结果的CN属性应该等于main\_service。

正如我们已经提到的，在目录服务中搜索通常是一个昂贵的操作。例如，要找到VU University的所有主服务器，需要搜索每个系的所有条目，并将结果合并到一个答案中。换句话说，为了得到一个答案，我们通常需要访问DIT的几个叶子节点。实际上，这也意味着需要访问几个dsl。相反，命名服务通常可以这样实现:查找操作只需要访问一个叶子节点。

LDAP的整个设置可以更进一步，允许多个树共存，同时还相互链接。这种方法在微软的Active Directory中得到了应用，导致了大量LDAP域[Allen和Lowe-Norris, 2003]。显然，在这样的组织中进行搜索非常复杂。为了避免一些可伸缩性问题，Active Directory通常假设有一个可以首先搜索的全局索引服务器(称为全局目录)。索引将指示需要进一步搜索哪些LDAP域。虽然LDAP本身已经利用层次结构来实现可伸缩性，但是将LDAP和DNS结合起来是很常见的。例如，LDAP中的每棵树都需要在根目录中访问(在Active Directory中称为域控制器)。根通常是在DNS名称下知道的，而DNS名称又可以通过适当的SRV记录找到，如上所述。

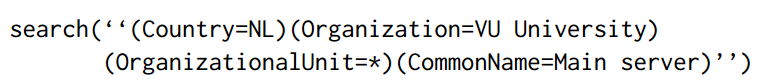
**去中心化的实现**

值得注意的是，随着peer-to-peer系统的出现，研究人员也一直在寻找基于分散属性命名系统的解决方案。特别是，点对点系统通常用于存储文件。最初，文件无法被搜索，只能通过密钥来查找。然而，能够基于描述符搜索文件是非常方便的，因为每个描述符只是一个(属性，值)对。显然，查询peer-to-peer系统中的每个节点以查看它是否包含匹配一个或多个此类对的文件是不可行的。我们需要的是(属性、值)对到索引服务器的映射，而索引服务器又指向匹配这些对的文件。

**使用分布式索引**

让我们首先看看构建(分布式)索引的情况。基本思想是将搜索查询表示为(属性、值)对的列表，就像在LDAP示例中一样。结果应该是匹配所有对的实体的(引用)列表。在点对点系统存储文件的情况下，可能会返回相关文件的密钥列表，然后客户端可以使用返回的密钥查找每个文件。（类似倒排索引？）

下面是一种直接的分布式索引方法。假设有d个不同的属性。在这种情况下，我们可以为每个d属性使用一个服务器，其中属性a的服务器为属性a具有值val的每个实体E维护一组(E,val)对。搜索查询，如：



将分别发送到属性为country、organization和CommonName的服务器，之后客户端需要查看服务器返回的所有三个集合中出现了哪些实体。为了防止服务器需要维护非常大的实体集，可以将每个服务器的实体集进一步分区，并分布在多个子服务器上，每个子服务器都与相同的属性相关联。更精确地说，如果我们有一组属性{a1，…，aN}然后对于每个属性ak，我们关联一个set Sk = {Sk 1，…， Sk nk}的nk个服务器。假设属性ak从集合Rk中获取值，我们构造一个全局映射F。



在本例中，服务器Sk j将跟踪与具有ak = v的文件相关联的每个键。这个方案的优点是简单。如果L(ak, v)是服务器F(ak, v)返回的键的集合，那么查询可以被表示为逻辑表达式，例如：



或者在客户端是这样表达的



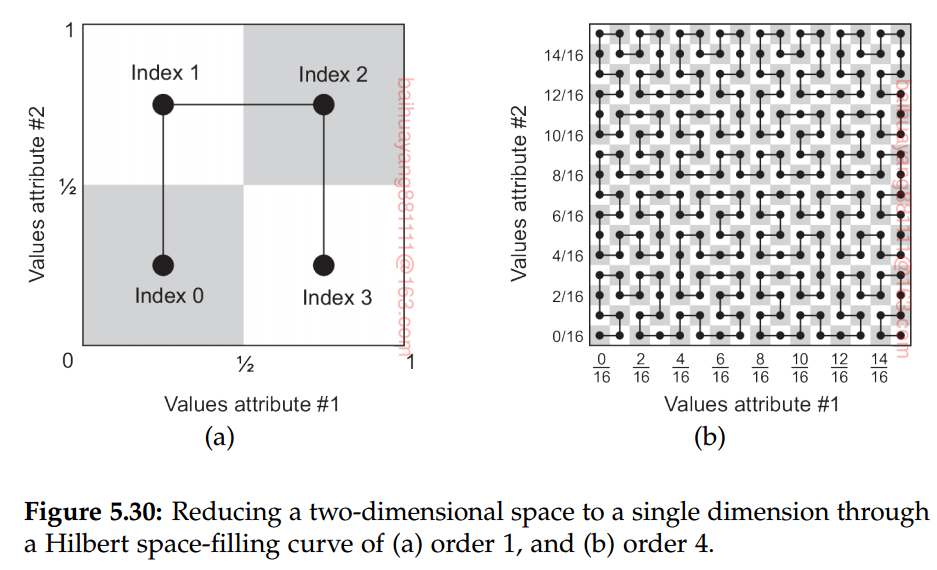
（完全搞不懂啊 上面这些）

不幸的是，这个方案（这方案是啥？？）有一些重要的缺点。首先，任何涉及k属性的查询都需要与k索引服务器联系，这可能会导致很大的通信开销。此外，与此相关的是，客户端需要处理索引服务器返回的集合。假设每个文件都有两个属性firstName和lastName，并且客户机正在寻找Pheriby Smith拥有的文件。虽然菲利比的名字很独特，但史密斯绝对不是。第三，尽管该方案允许保留某些未指定的属性(通过在查询中简单地不提及它们)，但它不容易支持范围查询，比如price = [1000 - 2500]。

**Space-filling curves**

实现基于属性的分散命名系统的一种常见方法是使用所谓的**Space-filling curves**。基本思想是通过简单的hash技术，将N维度属性{a1,a2,…aN}映射到一个单一维度。其中一个关键问题是，相同的索引服务器必须处理彼此“接近”的(属性、值)对。

让我们把事情具体化，看看一个流行的例子，即希尔伯特空间填充曲线（**Space-filling curves**）。通过只考虑两个维度，即只考虑两个不同的属性，可以很容易地解释这些问题。每个属性可能具有的值对应于二维空间中的一个轴。在不失一般性的情况下，我们假设每个属性在区间[0,1]中取值。作为正方形的一个近似，我们把它分成四个象限，如图5.30(a)所示。0≤x, y < 0.5的所有数据值(x, y)与索引0关联。0.5≤x, y < 1.0的值(x, y)与索引2相关联。



我们可以递归地对每个子正方形重复这个过程:将它分成四个较小的正方形，并通过一条线将较小的正方形连接起来。使用旋转和反射，我们确保这条线可以很好地连接到之前相邻的较大的子正方形(也被分成较小的正方形)。为了说明这一点，图5.30(a)为一阶希尔伯特曲线，图5.30(b)为一阶256个指标的曲线。一般情况下，k阶希尔伯特曲线连接2^2k个子平方，因此也有2^2k个指标。我们有很多方法可以系统地在二维空间中画出一条曲线，这个空间被分割成大小相等的正方形。此外，正如Sagan[1994]和Bader[2013]所解释的，该过程可以很容易地扩展到更高的维度。

空间填充曲线的一个重要特性是它们保持局部性：曲线上两个相互接近的指标对应多维空间中两个相互接近的点。(注意，反过来并不总是正确的:在多维空间中相邻的两点不一定在曲线上相邻。)

要完成这个故事，需要做几件事。首先，需要索引属性值。假设我们正在处理总共N个可能的属性{a1，…，aN} 每个实体为这N个属性中的每一个赋值。为了简单起见，我们假设每个属性值都规范化为区间[0,1]的值。然后很明显，一个实体E有值的元组(v1，…， vN)与n维空间中的实值坐标相关联，反过来又与我们讨论的二维情况下的n维子平方相关联。这种子正方形的中心对应于关联的Hilbert空间填充曲线上的索引，现在是与实体E关联的索引。当然，如果多个实体的关联坐标落在相同的子正方形中，那么它们的索引都是相同的。为了尽可能避免这种碰撞，我们需要使用高阶空间填充曲线。

其次，我们还需要能够搜索实体。根据属性值搜索实体的原则现在应该很清楚了。

假设我们正在寻找两个属性值a1和a2分别位于[vl1, vu 1]和[vl2, vu 2]区间内的文件。很明显，这描绘了一个矩形区域，曲线通过该区域，所有由与该区域相交的曲线段索引的文件都符合搜索条件。因此，我们需要一个操作，返回一系列与曲线相关的索引给定一个区域(以子平方表示)在相关的n维空间。这种操作显然依赖于所使用的空间填充曲线，但有趣的是，不一定依赖于实际实体。

最后，我们需要维护(对)与索引关联的实体的引用。在Squid系统中使用的一种方法是使用弦环。在Squid中，索引空间被选择为与弦环相同，即都使用m位标识符。显然，负责索引i的Chord节点将存储(对)i索引的实体的引用。

注：5.4节属性命名 大部分没看懂

**总结 略**