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。

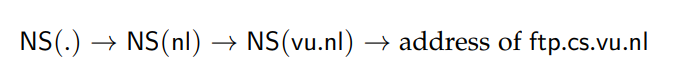
同样，如果一个地址可以重新分配给另一个实体，我们就不能使用地址作为标识符。考虑一下电话号码的使用，从某种意义上说，电话号码在一段时间内经常是指同一个人或组织，这是相当稳定的。然而，使用电话号码作为标识符将不起作用，因为它可以在一段时间内重新分配。因此，鲍勃的新面包店可能会在很长一段时间内接到爱丽丝旧古董店的电话。在这种情况下，使用Alice的真实标识符而不是她的电话号码会更好。

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）**

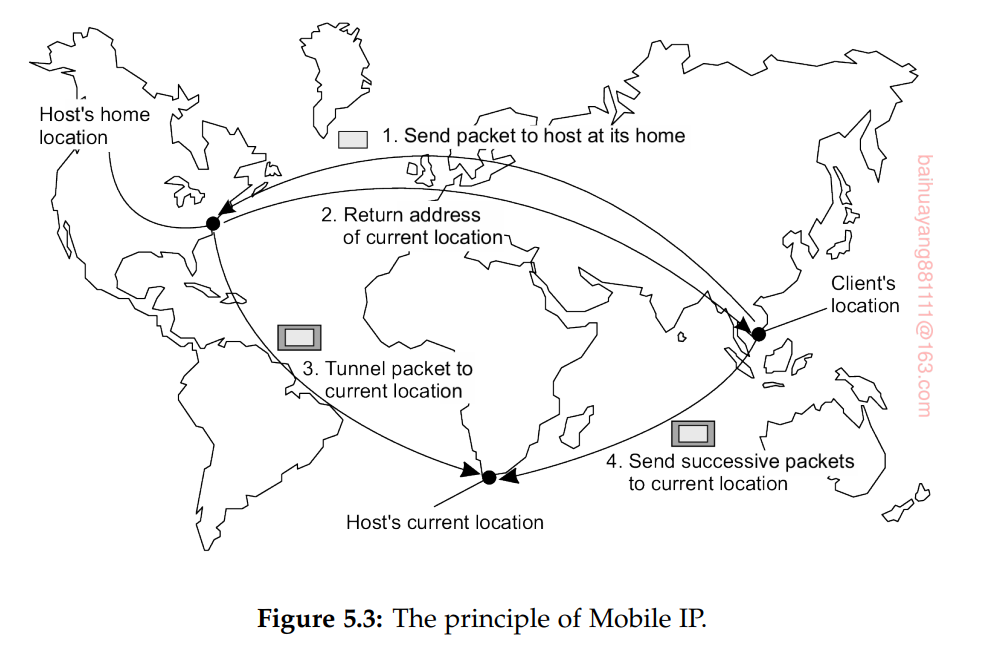
定位移动实体的另一种流行方法是使用forwarding pointers。原理很简单:当一个实体从A移动到B时，它会留下一个指向它在B的新位置的引用。这种方法的主要优点是它的简单性: ，例如通过使用传统的命名服务，只要找到了实体客户机就可以按照转发指针链查找当前地址。但它也有缺点。首先，如果不采取特别措施，一个高度移动的实体的链可能会变得非常长，以至于定位该实体的成本高得令人望而却步。其次，链中的所有中间位置都必须根据需要维护其转发指针链的一部分。第三个(也是相关的)缺点是容易受到断开链接的影响。一旦任何转发指针丢失，就无法再访问该实体。因此，一个重要的问题是保持链相对较短，并确保转发指针是健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

**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）**

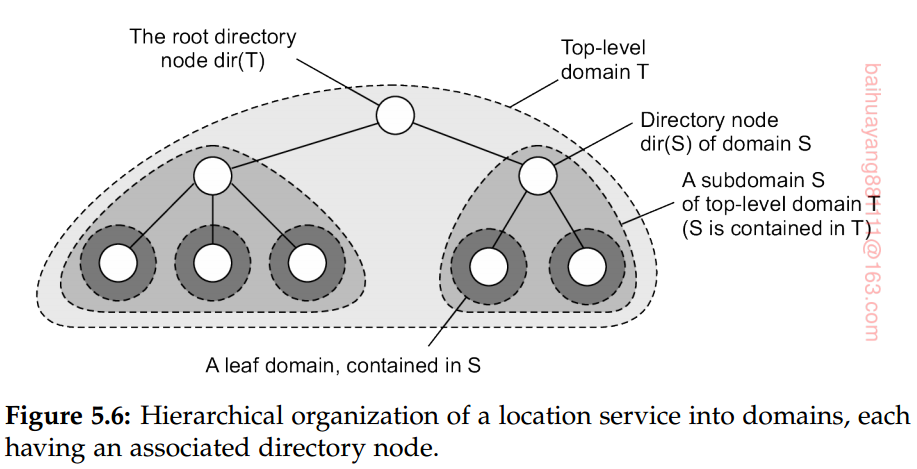
注：这一节的内容请自行查阅相关资料。

**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。

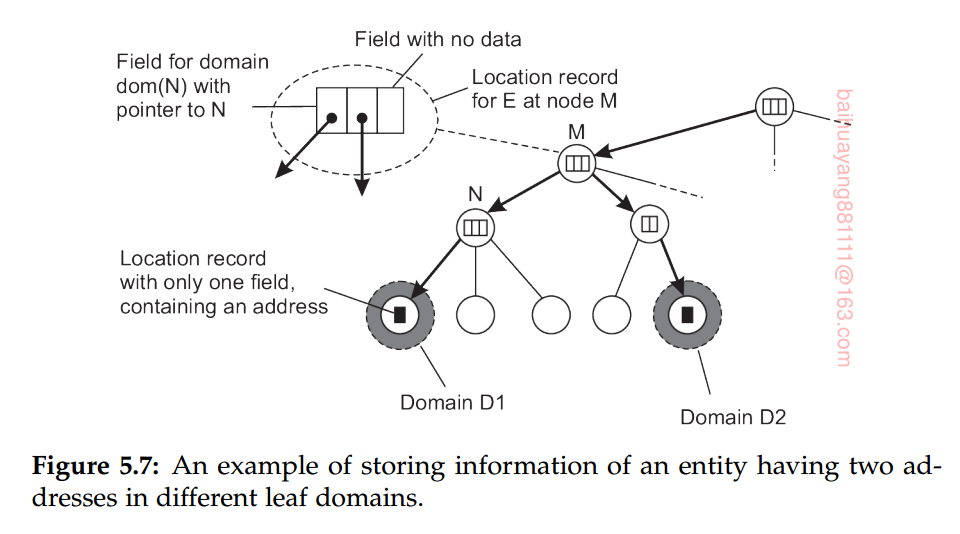
在层次结构中，网络被划分为一系列域。有一个跨整个网络的顶级域。其他的域都可以成为这个域的子域。最底层的域成为**叶子域**，代表了现实网络中一台实际的计算机或者手机等通信机器。一般的假设是，在较小的域中，将消息从一个节点传输到另一个节点所需的平均时间比在较大的域中要短。

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。

