流量，从而改善了所有应用的性能。

为了深刻理解缓存器带来的好处，我们考虑在图 2-12 场景下的一个例子。该图显示了两个网络，即机构（内部）网络和公共因特网的一部分。机构网络是一个高速的局域网，它的一台路由器与因特网上的一台路由器通过一条 15Mbps 的链路连接。这些初始服务器与因特网相连但位于全世界各地。假设对象的平均长度为 1Mb，从机构内的浏览器对这些初始服务器的平均访问速率为每秒 15 个请求。假设 HTTP 请求报文小到可以忽略，因而不会在网络中以及接入链路（从机构内部路由器到因特网路由器）上产生什么通信量。我们还假设在图 2-12 中从因特网接入链路一侧的路由器转发 HTTP 请求报文（在一个 IP 数据报中）开始，到它收到其响应报文（通常在多个 IP 数据报中）为止的时间平均为 2 秒。我们非正式地将该持续时延称为“因特网时延”。

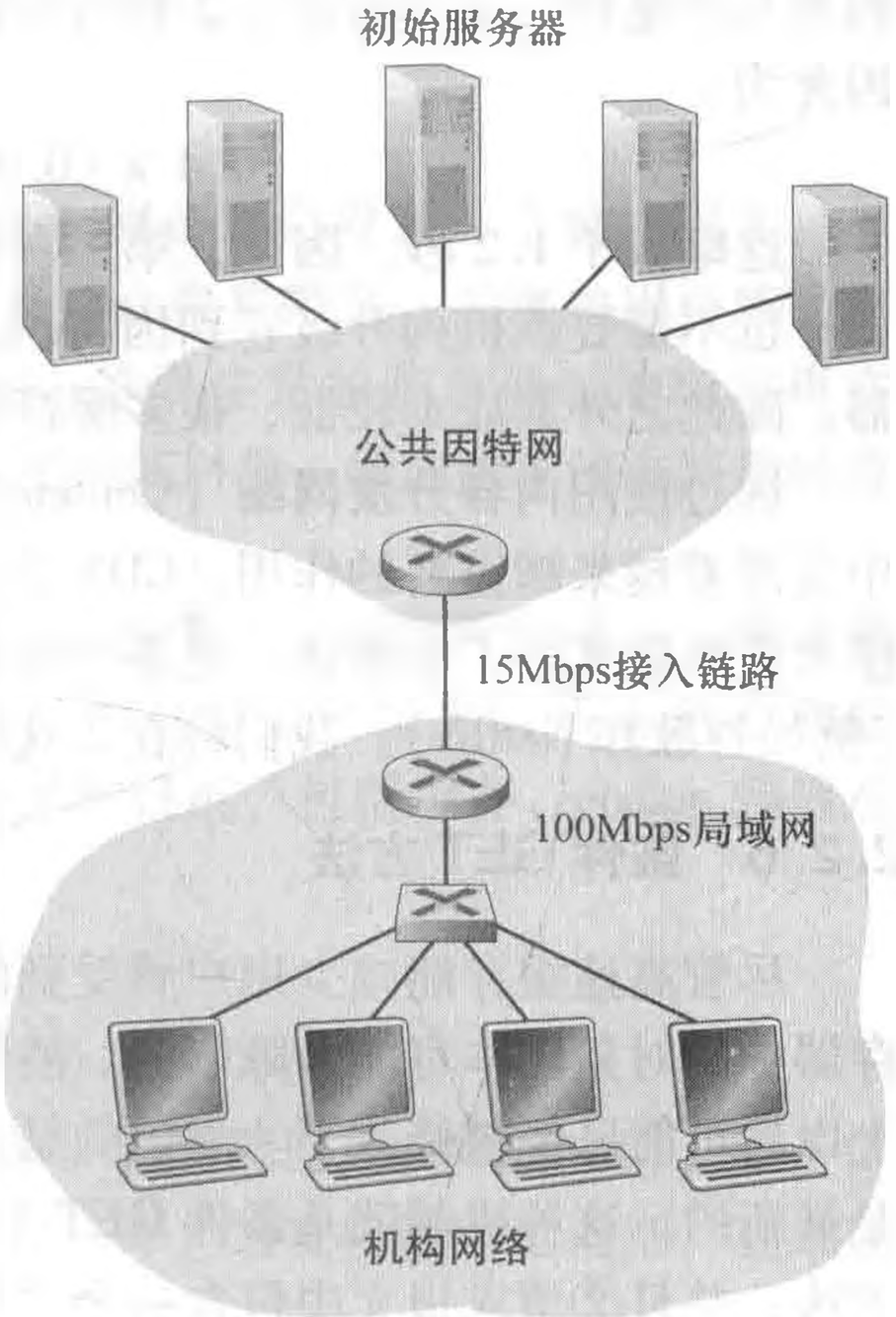

图 2-12 一个机构网络与因特网之间的瓶颈

总的响应时间，即从浏览器请求一个对象到接收到该对象为止的时间，是局域网时延、接入时延（即两台路由器之间的时延）和因特网时延之和。我们来粗略地估算一下这个时延。局域网上的流量强度（参见 1.4.2 节）为

（15 个请求 /s）×（1Mb/ 请求）/（100Mbps）= 0.15

然而接入链路上的流量强度（从因特网路由器到机构路由器）为

（15 个请求 /s）×（1Mb/ 请求）/（15Mbps）= 1

局域网上强度为 0.15 的通信量通常最多导致数十毫秒的时延，因此我们可以忽略局域网时延。然而，如在 1.4.2 节讨论的那样，如果流量强度接近 1（就像在图 2-12 中接入链路的情况那样），链路上的时延会变得非常大并且无限增长。因此，满足请求的平均响应时间将在分钟的量级上。显然，必须想办法来改进时间响应特性。

一个可能的解决办法就是增加接入链路的速率，如从 15Mbps 增加到 100Mbps。这可以将接入链路上的流量强度减少到 0.15，这样一来，两台路由器之间的链路时延也可以忽略了。这时，总的响应时间将大约为 2 秒，即为因特网时延。但这种解决方案也意味着该机构必须将它的接入链路由 15Mbps 升级为 100Mbps，这是一种代价很高的方案。

现在来考虑另一种解决方案，即不升级链路带宽而是在机构网络中安装一个 Web 缓存器。这种解决方案如图 2-13 所示。实践中的命中率（即由一个缓存器所满足的请求的比率）通常在 0.2 ~ 0.7 之间。为了便于阐述，我们假设该机构的缓存

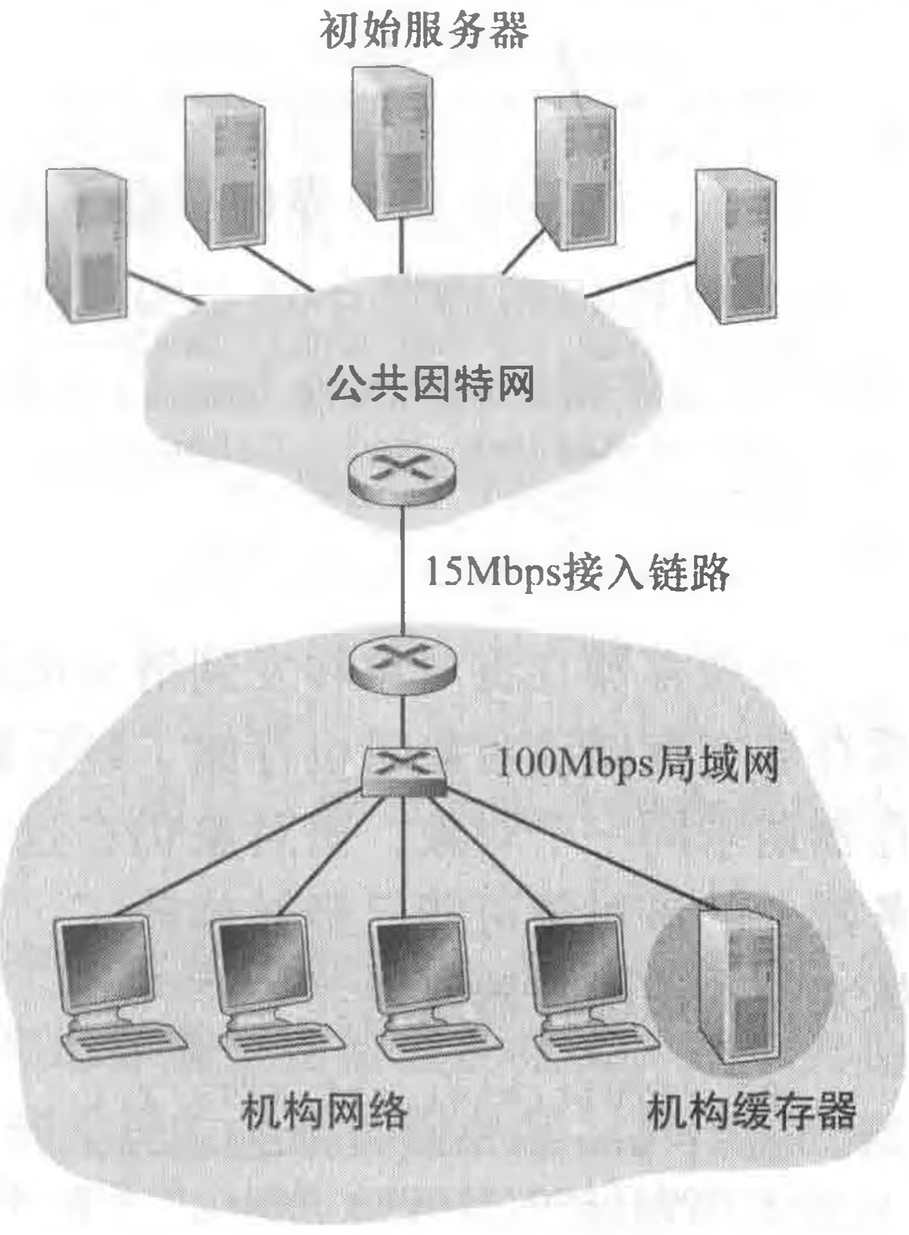

图 2-13 为机构网络添加一台缓存器

命中率为0.4。因为客户和缓存连接在一个相同的高速局域网上，这样40%的请求将几乎立即会由缓存器得到响应，时延约在10ms以内。然而，剩下的60%的请求仍然要由初始服务器来满足。但是只有60%的被请求对象通过接入链路，在接入链路上的流量强度从1.0减小到0.6。一般而言，在15Mbps链路上，当流量强度小于0.8时对应的时延较小，约为几十毫秒。这个时延与2秒因特网时延相比是微不足道的。考虑这些之后，平均时延因此为

$$0.4 \times (0.010 \text{ 秒}) + 0.6 \times (2.01 \text{ 秒})$$

这略大于1.2秒。因此，第二种解决方案提供的响应时延甚至比第一种解决方案更低，也不需要该机构升级它到因特网的链路。该机构理所当然地要购买和安装Web缓存器。除此之外其成本较低，很多缓存器使用了运行在廉价PC上的公共域软件。

通过使用**内容分发网络**（Content Distribution Network，CDN），Web缓存器正在因特网中发挥着越来越重要的作用。CDN公司在因特网上安装了许多地理上分散的缓存器，因而使大量流量实现了本地化。有多个共享的CDN（例如Akamai和Limelight）和专用的CDN（例如谷歌和Netflix）。我们将在2.6节中更为详细地讨论CDN。

2.2.6　条件GET方法

尽管高速缓存能减少用户感受到的响应时间，但也引入了一个新的问题，即存放在缓存器中的对象副本可能是陈旧的。换句话说，保存在服务器中的对象自该副本缓存在客户上以后可能已经被修改了。幸运的是，HTTP协议有一种机制，允许缓存器证实它的对象是最新的。这种机制就是**条件GET**（conditional GET）方法。如果：①请求报文使用GET方法；并且②请求报文中包含一个“If-Modified-Since:”首部行。那么，这个HTTP请求报文就是一个条件GET请求报文。

为了说明GET方法的操作方式，我们看一个例子。首先，一个代理缓存器（proxy cache）代表一个请求浏览器，向某Web服务器发送一个请求报文：

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
```

其次，该Web服务器向缓存器发送具有被请求的对象的响应报文：

```
HTTP/1.1 200 OK
Date: Sat, 3 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)
Last-Modified: Wed, 9 Sep 2015 09:23:24
Content-Type: image/gif
(data data data data data ...)
```

该缓存器在将对象转发到请求的浏览器的同时，也在本地缓存了该对象。重要的是，缓存器在存储该对象时也存储了最后修改日期。最后，一个星期后，另一个用户经过该缓存器请求同一个对象，该对象仍在这个缓存器中。由于在过去的一个星期中位于Web服务器上的该对象可能已经被修改了，该缓存器通过发送一个条件GET执行最新检查。具体来说，该缓存器发送：

```
GET /fruit/kiwi.gif HTTP/1.1
Host: www.exotiquecuisine.com
If-modified-since: Wed, 9 Sep 2015 09:23:24
```

值得注意的是If-Modified-Since:首部行的值正好等于一星期前服务器发送的响应报文

中的 Last-Modified：首部行的值。该条件 GET 报文告诉服务器，仅当自指定日期之后该对象被修改过，才发送该对象。假设该对象自 2015 年 9 月 9 日 09：23：24 后没有被修改。接下来的第四步，Web 服务器向该缓存器发送一个响应报文：

```
HTTP/1.1 304 Not Modified
Date: Sat, 10 Oct 2015 15:39:29
Server: Apache/1.3.0 (Unix)

(empty entity body)
```

我们看到，作为对该条件 GET 方法的响应，该 Web 服务器仍发送一个响应报文，但并没有在该响应报文中包含所请求的对象。包含该对象只会浪费带宽，并增加用户感受到的响应时间，特别是如果该对象很大的时候更是如此。值得注意的是在最后的响应报文中，状态行中为 304 Not Modified，它告诉缓存器可以使用该对象，能向请求的浏览器转发它（该代理缓存器）缓存的该对象副本。

我们现在完成了对 HTTP 的讨论，这是我们详细学习的第一个因特网协议（应用层协议）。我们已经学习了 HTTP 报文的格式，学习了当发送和接收这些报文时 Web 客户和服务器所采取的动作。我们还学习了一点 Web 应用程序基础设施，包括缓存、cookie 和后端数据库，所有这些都以某种方式与 HTTP 协议有关。

2.3 因特网中的电子邮件

自从有了因特网，电子邮件就在因特网上流行起来。当因特网还在襁褓中时，电子邮件已经成为最为流行的应用程序［Segaller 1998］，年复一年，它变得越来越精细，越来越强大。它仍然是当今因特网上最重要和实用的应用程序之一。

与普通邮件一样，电子邮件是一种异步通信媒介，即当人们方便时就可以收发邮件，不必与他人的计划进行协调。与普通邮件相比，电子邮件更为快速并且易于分发，而且价格便宜。现代电子邮件具有许多强大的特性，包括具有附件、超链接、HTML 格式文本和图片的报文。

在本节中，我们将讨论位于因特网电子邮件的核心地位的应用层协议。在深入讨论这些应用层协议之前，我们先总体上看看因特网电子邮件系统和它的关键组件。

图 2-14 给出了因特网电子邮件系统的总体情况。从该图中我们可以看到它有 3 个主要组成部分：**用户代理**（user agent）、**邮件服务器**（mail server）和**简单邮件传输协议**（Simple Mail Transfer Protocol，SMTP）。下面我们结合发送方 Alice 发电子邮件给接收方 Bob 的场景，对每个组成部分进行描述。用户代理允许用户阅读、回复、转发、保存和撰写报文。微软的 Outlook 和 Apple Mail 是电子邮件用户代理的例子。当 Alice 完成邮件撰写时，她的邮件代理向其邮件服务器发送邮件，此时邮件放在邮件服务器的外出报文队列中。当 Bob 要阅读报文时，他的用户代理在其邮件服务器的邮箱中取得该报文。

邮件服务器形成了电子邮件体系结构的核心。每个接收方（如 Bob）在其中的某个邮件服务器上有一个**邮箱**（mailbox）。Bob 的邮箱管理和维护着发送给他的报文。一个典型的邮件发送过程是：从发送方的用户代理开始，传输到发送方的邮件服务器，再传输到接收方的邮件服务器，然后在这里被分发到接收方的邮箱中。当 Bob 要在他的邮箱中读取该报文时，包含他邮箱的邮件服务器（使用用户名和口令）来鉴别 Bob。Alice 的邮箱也必

须能处理 Bob 的邮件服务器的故障。如果 Alice 的服务器不能将邮件交付给 Bob 的服务器，Alice 的邮件服务器在一个**报文队列**（message queue）中保持该报文并在以后尝试再次发送。通常每 30 分钟左右进行一次尝试；如果几天后仍不能成功，服务器就删除该报文并以电子邮件的形式通知发送方（Alice）。

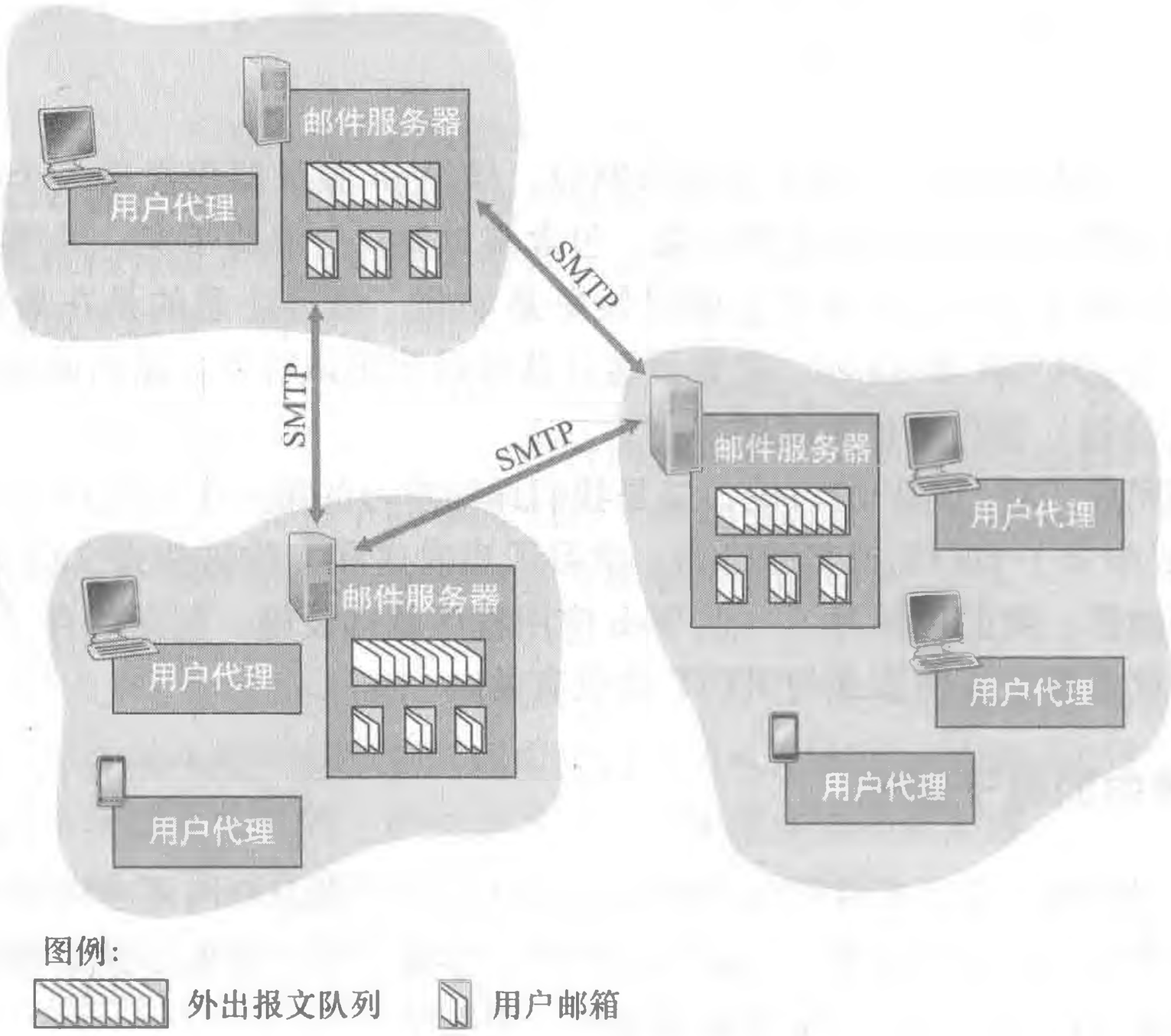

图 2-14　因特网电子邮件系统的总体描述

SMTP 是因特网电子邮件中主要的应用层协议。它使用 TCP 可靠数据传输服务，从发送方的邮件服务器向接收方的邮件服务器发送邮件。像大多数应用层协议一样，SMTP 也有两个部分：运行在发送方邮件服务器的客户端和运行在接收方邮件服务器的服务器端。每台邮件服务器上既运行 SMTP 的客户端也运行 SMTP 的服务器端。当一个邮件服务器向其他邮件服务器发送邮件时，它就表现为 SMTP 的客户；当邮件服务器从其他邮件服务器上接收邮件时，它就表现为一个 SMTP 的服务器。

2.3.1　SMTP

RFC 5321 给出了 SMTP 的定义。SMTP 是因特网电子邮件的核心。如前所述，SMTP 用于从发送方的邮件服务器发送报文到接收方的邮件服务器。SMTP 问世的时间比 HTTP 要长得多（初始的 SMTP 协议的 RFC 可追溯到 1982 年，而 SMTP 在此之前很长一段时间就已经出现了）。尽管电子邮件应用在因特网上的独特地位可以证明 SMTP 有着众多非常出色的性质，但它所具有的某种陈旧特征表明它仍然是一种继承的技术。例如，它限制所有邮件报文的体部分（不只是其首部）只能采用简单的 7 比特 ASCII 表示。在 20 世纪 80 年代早期，这种限制是明智的，因为当时传输能力不足，没有人会通过电子邮件发送大的附件或是大的图片、声音或者视频文件。然而，在今天的多媒体时代，7 位 ASCII 的限制的确有点痛苦，即在用 SMTP 传送邮件之前，需要将二进制多媒体数据编码为 ASCII 码，并且在使用 SMTP 传输后要求将相应的

ASCII 码邮件解码还原为多媒体数据。2.2 节讲过，使用 HTTP 传送前不需要将多媒体数据编码为 ASCII 码。

为了描述 SMTP 的基本操作，我们观察一种常见的情景。假设 Alice 想给 Bob 发送一封简单的 ASCII 报文。

1）Alice 调用她的邮件代理程序并提供 Bob 的邮件地址（例如 bob@ someschool. edu），撰写报文，然后指示用户代理发送该报文。

2）Alice 的用户代理把报文发给她的邮件服务器，在那里该报文被放在报文队列中。

3）运行在 Alice 的邮件服务器上的 SMTP 客户端发现了报文队列中的这个报文，它就创建一个到运行在 Bob 的邮件服务器上的 SMTP 服务器的 TCP 连接。

4）在经过一些初始 SMTP 握手后，SMTP 客户通过该 TCP 连接发送 Alice 的报文。

5）在 Bob 的邮件服务器上，SMTP 的服务器端接收该报文。Bob 的邮件服务器然后将该报文放入 Bob 的邮箱中。

6）在 Bob 方便的时候，他调用用户代理阅读该报文。

图 2-15 总结了上述这个情况。

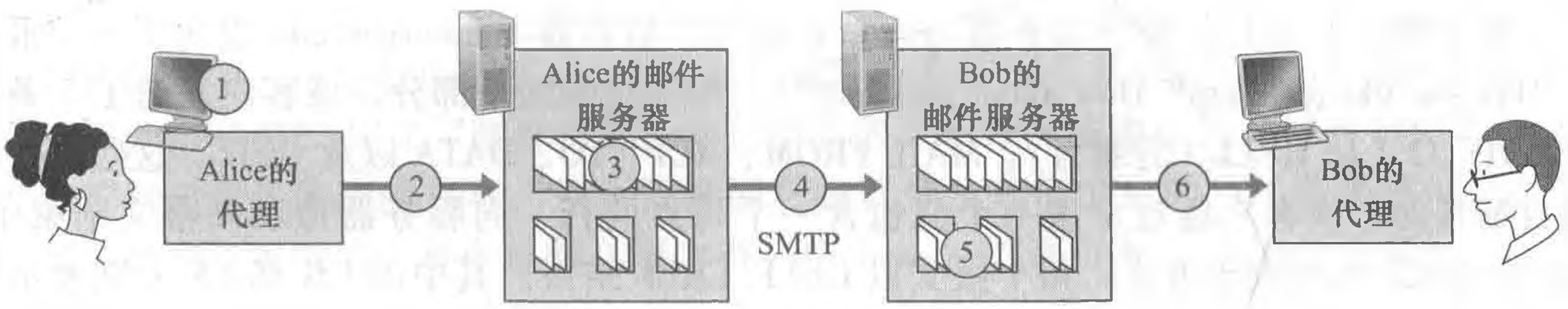

图 2-15 Alice 向 Bob 发送一条报文

观察到下述现象是重要的：SMTP 一般不使用中间邮件服务器发送邮件，即使这两个邮件服务器位于地球的两端也是这样。假设 Alice 的邮件服务器在中国香港，而 Bob 的服务器在美国圣路易斯，那么这个 TCP 连接也是从香港服务器到圣路易斯服务器之间的直接相连。特别是，如果 Bob 的邮件服务器没有开机，该报文会保留在 Alice 的邮件服务器上并等待进行新的尝试，这意味着邮件并不在中间的某个邮件服务器存留。

我们现在仔细观察一下，SMTP 是如何将一个报文从发送邮件服务器传送到接收邮件服务器的。我们将看到，SMTP 与人类面对面交往的行为方式有许多类似性。首先，客户 SMTP（运行在发送邮件服务器主机上）在 25 号端口建立一个到服务器 SMTP（运行在接收邮件服务器主机上）的 TCP 连接。如果服务器没有开机，客户会在稍后继续尝试连接。一旦连接建立，服务器和客户执行某些应用层的握手，就像人们在互相交流前先进行自我介绍一样。SMTP 的客户和服务器在传输信息前先相互介绍。在 SMTP 握手的阶段，SMTP 客户指示发送方的邮件地址（产生报文的那个人）和接收方的邮件地址。一旦该 SMTP 客户和服务器彼此介绍之后，客户发送该报文。SMTP 能依赖 TCP 提供的可靠数据传输无差错地将邮件投递到接收服务器。该客户如果有另外的报文要发送到该服务器，就在该相同的 TCP 连接上重复这种处理；否则，它指示 TCP 关闭连接。

接下来我们分析一个在SMTP客户（C）和SMTP服务器（S）之间交换报文文本的例子。客户的主机名为crepes. fr，服务器的主机名为hamburger. edu。以C：开头的ASCII码文本行正是客户交给其TCP套接字的那些行，以S：开头的ASCII码则是服务器发送给其TCP套接字的那些行。一旦创建了TCP连接，就开始了下列过程。

```
S:  220 hamburger.edu
C:  HELO crepes.fr
S:  250 Hello crepes.fr, pleased to meet you
C:  MAIL FROM: <alice@crepes.fr>
S:  250 alice@crepes.fr ... Sender ok
C:  RCPT TO: <bob@hamburger.edu>
S:  250 bob@hamburger.edu ... Recipient ok
C:  DATA
S:  354 Enter mail, end with "." on a line by itself
C:  Do you like ketchup?
C:  How about pickles?
C:  .
S:  250 Message accepted for delivery
C:  QUIT
S:  221 hamburger.edu closing connection
```

在上例中，客户从邮件服务器crepes. fr向邮件服务器hamburger. edu发送了一个报文（“Do you like ketchup? How about pickles?”）。作为对话的一部分，该客户发送了5条命令：HELO（是HELLO的缩写）、MAIL FROM、RCPT TO、DATA以及QUIT。这些命令都是自解释的。该客户通过发送一个只包含一个句点的行，向服务器指示该报文结束了。（按照ASCII码的表示方法，每个报文以CRLF. CRLF结束，其中的CR和LF分别表示回车和换行。）服务器对每条命令做出回答，其中每个回答含有一个回答码和一些（可选的）英文解释。我们在这里指出SMTP用的是持续连接：如果发送邮件服务器有几个报文发往同一个接收邮件服务器，它可以通过同一个TCP连接发送这些所有的报文。对每个报文，该客户用一个新的MAIL FROM：crepes. fr开始，用一个独立的句点指示该邮件的结束，并且仅当所有邮件发送完后才发送QUIT。

我们强烈推荐你使用Telnet与一个SMTP服务器进行一次直接对话。使用的命令是

```
telnet serverName 25
```

其中serverName是本地邮件服务器的名称。当你这么做时，就直接在本地主机与邮件服务器之间建立一个TCP连接。输完上述命令后，你立即会从该服务器收到220回答。接下来，在适当的时机发出HELO、MAIL FROM、RCPT TO、DATA、CRLF. CRLF以及QUIT等SMTP命令。强烈推荐你做本章后面的编程作业3。在该作业中，你将在SMTP的客户端实现一个简单的用户代理，它允许你经本地邮件服务器向任意的接收方发送电子邮件报文。

2.3.2　与HTTP的对比

我们简要地比较一下SMTP和HTTP。这两个协议都用于从一台主机向另一台主机传送文件：HTTP从Web服务器向Web客户（通常是一个浏览器）传送文件（也称为对象）；SMTP从一个邮件服务器向另一个邮件服务器传送文件（即电子邮件报文）。当进行文件传送时，持续的HTTP和SMTP都使用持续连接。因此，这两个协议有一些共同特征。然而，两者之间也有一些重要的区别。首先，HTTP主要是一个**拉协议**（pull protocol），即在方便的时候，某些人在Web服务器上装载信息，用户使用HTTP从该服务器拉取这些

信息。特别是 TCP 连接是由想接收文件的机器发起的。另一方面，SMTP 基本上是一个**推协议**（push protocol），即发送邮件服务器把文件推向接收邮件服务器。特别是，这个 TCP 连接是由要发送该文件的机器发起的。

第二个区别就是我们前面间接地提到过的，SMTP 要求每个报文（包括它们的体）采用 7 比特 ASCII 码格式。如果某报文包含了非 7 比特 ASCII 字符（如具有重音的法文字符）或二进制数据（如图形文件），则该报文必须按照 7 比特 ASCII 码进行编码。HTTP 数据则不受这种限制。

第三个重要区别是如何处理一个既包含文本又包含图形（也可能是其他媒体类型）的文档。如我们在 2.2 节知道的那样，HTTP 把每个对象封装到它自己的 HTTP 响应报文中，而 SMTP 则把所有报文对象放在一个报文之中。

2.3.3 邮件报文格式

当 Alice 给 Bob 写一封邮寄时间很长的普通信件时，她可能要在信的上部包含各种各样的环境首部信息，如 Bob 的地址、她自己的回复地址以及日期等。同样，当一个人给另一个人发送电子邮件时，一个包含环境信息的首部位于报文体前面。这些环境信息包括在一系列首部行中，这些行由 RFC 5322 定义。首部行和该报文的体用空行（即回车换行）进行分隔。RFC 5322 定义了邮件首部行和它们的语义解释的精确格式。如同 HTTP 协议，每个首部行包含了可读的文本，是由关键词后跟冒号及其值组成的。某些关键词是必需的，另一些则是可选的。每个首部必须含有一个 From：首部行和一个 To：首部行；一个首部也许包含一个 Subject：首部行以及其他可选的首部行。重要的是注意到下列事实：这些首部行不同于我们在 2.3.1 节所学到的 SMTP 命令（即使那里包含了某些相同的词汇，如 from 和 to）。那节中的命令是 SMTP 握手协议的一部分；本节中考察的首部行则是邮件报文自身的一部分。

一个典型的报文首部看起来如下：

```
From: alice@crepes.fr
To: bob@hamburger.edu
Subject: Searching for the meaning of life.
```

在报文首部之后，紧接着一个空白行，然后是以 ACSII 格式表示的报文体。你应当用 Telnet 向邮件服务器发送包含一些首部行的报文，包括 Subject：首部行。为此，输入命令 telnet serverName 25，如在 2.3.1 节中讨论的那样。

2.3.4 邮件访问协议

一旦 SMTP 将邮件报文从 Alice 的邮件服务器交付给 Bob 的邮件服务器，该报文就被放入了 Bob 的邮箱中。在此讨论中，我们按惯例假定 Bob 是通过登录到服务器主机，并直接在该主机上运行一个邮件阅读程序来阅读他的邮件的。直到 20 世纪 90 年代早期，这都是一种标准方式。而在今天，邮件访问使用了一种客户 - 服务器体系结构，即典型的用户通过在用户端系统上运行的客户程序来阅读电子邮件，这里的端系统可能是办公室的 PC、便携机或者是智能手机。通过在本地主机上运行邮件客户程序，用户享受一系列丰富的特性，包括查看多媒体报文和附件的能力。

假设 Bob（接收方）在其本地 PC 上运行用户代理程序，考虑在他的本地 PC 上也放置一个邮件服务器是自然而然的事。在这种情况下，Alice 的邮件服务器就能直接

与 Bob 的 PC 进行对话了。然而这种方法会有一个问题。前面讲过邮件服务器管理用户的邮箱，并且运行 SMTP 的客户端和服务器端。如果 Bob 的邮件服务器位于他的 PC 上，那么为了能够及时接收可能在任何时候到达的新邮件，他的 PC 必须总是不间断地运行着并一直保持在线。这对于许多因特网用户而言是不现实的。相反，典型的用户通常在本地 PC 上运行一个用户代理程序，而它访问存储在总是保持开机的共享邮件服务器上的邮箱。该邮件服务器与其他用户共享，并且通常由用户的 ISP 进行维护（如大学或公司）。

现在我们考虑当从 Alice 向 Bob 发送一个电子邮件报文时所取的路径。我们刚才已经知道，在沿着该路径的某些点上，需要将电子邮件报文存放在 Bob 的邮件服务器上。通过让Alice 的用户代理直接向 Bob 的邮件服务器发送报文，就能够做到这一点。这能够由 SMTP 来完成：实际上，SMTP 被设计成将电子邮件从一台主机推到另一台主机。然而，通常Alice 的用户代理和 Bob 的邮件服务器之间并没有一个直接的 SMTP 对话。相反，如图 2-16 所示，Alice 的用户代理用 SMTP 将电子邮件报文推入她的邮件服务器，接着她的邮件服务器（作为一个 SMTP 客户）再用 SMTP 将该邮件中继到 Bob 的邮件服务器。为什么该过程要分成两步呢？主要是因为不通过 Alice 的邮件服务器进行中继，Alice 的用户代理将没有任何办法到达一个不可达的目的地接收服务器。通过首先将邮件存放在自己的邮件服务器中，Alice 的邮件服务器可以重复地尝试向 Bob 的邮件服务器发送该报文，如每 30 分钟一次，直到 Bob 的邮件服务器变得运行为止。（并且如果 Alice 的邮件服务器关机，她则能向系统管理员进行申告！）SMTP RFC 文档定义了如何使用 SMTP 命令经过多个 SMTP 服务器进行报文中继。

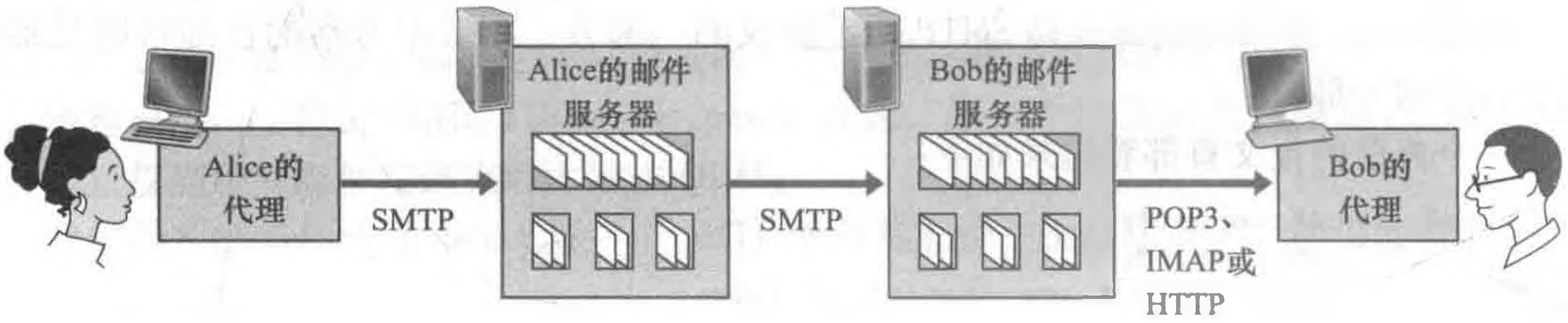

图 2-16　电子邮件协议及其通信实体

但是对于该难题仍然有一个疏漏的环节！像 Bob 这样的接收方，是如何通过运行其本地 PC 上的用户代理，获得位于他的某 ISP 的邮件服务器上的邮件呢？值得注意的是 Bob 的用户代理不能使用 SMTP 得到报文，因为取报文是一个拉操作，而 SMTP 协议是一个推协议。通过引入一个特殊的邮件访问协议来解决这个难题，该协议将 Bob 邮件服务器上的报文传送给他的本地 PC。目前有一些流行的邮件访问协议，包括**第三版的邮局协议**（Post Office Protocol—Version 3，POP3）、**因特网邮件访问协议**（Internet Mail Access Protocol，IMAP）以及 HTTP。

图 2-16 总结了应用于因特网电子邮件的一些协议：SMTP 用来将邮件从发送方的邮件服务器传输到接收方的邮件服务器；SMTP 也用来将邮件从发送方的用户代理传送到发送方的邮件服务器。如 POP3 这样的邮件访问协议用来将邮件从接收方的邮件服务器传送到接收方的用户代理。

1. POP3

POP3 是一个极为简单的邮件访问协议，由 RFC 1939 进行定义。文档 RFC 1939 简短且可读性强。因为该协议非常简单，故其功能相当有限。当用户代理（客户）打开了一个到邮件服务器（服务器）端口 110 上的 TCP 连接后，POP3 就开始工作了。随着建立 TCP 连接，POP3 按照三个阶段进行工作：特许（authorization）、事务处理以及更新。在第一个阶段即特许阶段，用户代理发送（以明文形式）用户名和口令以鉴别用户。在第二个阶段即事务处理阶段，用户代理取回报文；同时在这个阶段用户代理还能进行如下操作，对报文做删除标记，取消报文删除标记，以及获取邮件的统计信息。在第三个阶段即更新阶段，它出现在客户发出了 quit 命令之后，目的是结束该 POP3 会话；这时，该邮件服务器删除那些被标记为删除的报文。

在 POP3 的事务处理过程中，用户代理发出一些命令，服务器对每个命令做出回答。回答可能有两种：+OK（有时后面还跟有服务器到客户的数据），被服务器用来指示前面的命令是正常的；-ERR，被服务器用来指示前面的命令出现了某些差错。

特许阶段有两个主要的命令：user <user name> 和 pass <password>。为了举例说明这两个命令，我们建议你直接用 Telnet 登录到 POP3 服务器的 110 端口，然后发出这两个命令。假设邮件服务器的名字为 mailServer，那么你将看到类似的过程：

```
telnet mailServer 110
+OK POP3 server ready
user bob
+OK
pass hungry
+OK user successfully logged on
```

如果你的命令拼写错了，该 POP3 服务器将返回一个 -ERR 报文。

现在我们来看一下事务处理过程。使用 POP3 的用户代理通常被用户配置为“下载并删除”或者“下载并保留”方式。POP3 用户代理发出的命令序列取决于用户代理程序被配置为这两种工作方式的哪一种。使用下载并删除方式，用户代理发出 list、retr 和 dele 命令。举例来说，假设用户在他（她）的邮箱里有两个报文。在下面的对话中，C:（代表客户）是用户代理，S:（代表服务器）是邮件服务器。事务处理过程将类似于如下过程：

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: (blah blah ...
S: ..................
S: ..........blah)
S: .
C: dele 1
C: retr 2
S: (blah blah ...
S: ..................
S: ..........blah)
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

用户代理首先请求邮件服务器列出所有存储的报文的长度。接着用户代理从邮件服务

器取回并删除每封邮件。注意到在特许阶段以后，用户代理仅使用四个命令 list、retr、dele 和 quit，这些命令的语法定义在 RFC 1939 中。在处理 quit 命令后，POP3 服务器进入更新阶段，从用户的邮箱中删除邮件 1 和 2。

使用下载并删除方式存在的问题是，邮件接收方 Bob 可能是移动的，可能希望从多个不同的机器访问他的邮件报文，如从办公室的 PC、家里的 PC 或他的便携机来访问邮件。下载并删除方式将对 Bob 的邮件报文根据这 3 台机器进行划分，特别是如果 Bob 最先是在他办公室的 PC 上收取了一条邮件，那么晚上当他在家里时，通过他的便携机将不能再收取该邮件。使用下载并保留方式，用户代理下载某邮件后，该邮件仍保留在邮件服务器上。这时，Bob 就能通过不同的机器重新读取这些邮件；他能在工作时收取一封报文，而在工作回家后再次访问它。

在用户代理与邮件服务器之间的 POP3 会话期间，该 POP3 服务器保留了一些状态信息；特别是记录了哪些用户报文被标记为删除了。然而，POP3 服务器并不在 POP3 会话过程中携带状态信息。会话中不包括状态信息大大简化了 POP3 服务的实现。

2. IMAP

使用 POP3 访问时，一旦 Bob 将邮件下载到本地主机后，他就能建立邮件文件夹，并将下载的邮件放入该文件夹中。然后 Bob 可以删除报文，在文件夹之间移动报文，并查询报文（通过发送方的名字或报文主题）。但是这种文件夹和报文存放在本地主机上的方式，会给移动用户带来问题，因为他更喜欢使用一个在远程服务器上的层次文件夹，这样他可以从任何一台机器上对所有报文进行访问。使用 POP3 是不可能做到这一点的，POP3 协议没有给用户提供任何创建远程文件夹并为报文指派文件夹的方法。

为了解决这个或其他一些问题，由 RFC 3501 定义的因特网邮件访问协议（IMAP）应运而生。和 POP3 一样，IMAP 是一个邮件访问协议，但是它比 POP3 具有更多的特色，不过也比 POP3 复杂得多。（因此客户和服务器端的实现也都复杂得多。）

IMAP 服务器把每个报文与一个文件夹联系起来；当报文第一次到达服务器时，它与收件人的 INBOX 文件夹相关联。收件人则能够把邮件移到一个新的、用户创建的文件夹中，阅读邮件，删除邮件等。IMAP 协议为用户提供了创建文件夹以及将邮件从一个文件夹移动到另一个文件夹的命令。IMAP 还为用户提供了在远程文件夹中查询邮件的命令，按指定条件去查询匹配的邮件。值得注意的是，与 POP3 不同，IMAP 服务器维护了 IMAP 会话的用户状态信息，例如，文件夹的名字以及哪些报文与哪些文件夹相关联。

IMAP 的另一个重要特性是它具有允许用户代理获取报文某些部分的命令。例如，一个用户代理可以只读取一个报文的报文首部，或只是一个多部分 MIME 报文的一部分。当用户代理和其邮件服务器之间使用低带宽连接（如一个低速调制解调器链路）的时候，这个特性非常有用。使用这种低带宽连接时，用户可能并不想取回他邮箱中的所有邮件，尤其要避免可能包含如音频或视频片断的大邮件。

3. 基于 Web 的电子邮件

今天越来越多的用户使用他们的 Web 浏览器收发电子邮件。20 世纪 90 年代中期 Hotmail 引入了基于 Web 的接入。今天，谷歌、雅虎以及几乎所有重要的大学或者公

司也提供了基于 Web 的电子邮件。使用这种服务，用户代理就是普通的浏览器，用户和他远程邮箱之间的通信则通过 HTTP 进行。当一个收件人（如 Bob），想从他的邮箱中访问一个报文时，该电子邮件报文从 Bob 的邮件服务器发送到他的浏览器，使用的是 HTTP 而不是 POP3 或者 IMAP 协议。当发件人（如 Alice）要发送一封电子邮件报文时，该电子邮件报文从 Alice 的浏览器发送到她的邮件服务器，使用的是 HTTP 而不是 SMTP。然而，Alice 的邮件服务器在与其他的邮件服务器之间发送和接收邮件时，仍然使用的是 SMTP。

2.4 DNS：因特网的目录服务

人类能以很多方式来标识。例如，我们能够通过出生证书上的名字来标识；能够通过社会保险号码来标识；也能够通过驾驶执照上的号码来标识。尽管这些标识办法都可以用来识别一个人，但是在特定环境下，某种识别方法可能比另一种方法更为适合。例如，IRS（美国那个声名狼藉的税务征收机构）的计算机更喜欢使用定长的社会保险号码而不是出生证书上的姓名。另一方面，普通人乐于使用更好记的出生证书上的姓名而不是社会保险号码。（毫无疑问，你能想象人们之间以这种方式说话吗？如“你好，我叫132-67-9875。请找一下我的丈夫 178-87-1146”。）

因特网上的主机和人类一样，可以使用多种方式进行标识。主机的一种标识方法是用它的**主机名**（hostname），如 www. facebook. com、www. google. com、gaia. cs. umass. edu 以及 cis. poly. edu 等，这些名字便于记忆也乐于被人们接受。然而，主机名几乎没有提供（即使有也很少）关于主机在因特网中位置的信息。（一个名为 www. eurecom. fr 的主机以国家码 .fr 结束，告诉我们该主机很可能在法国，仅此而已。）况且，因为主机名可能由不定长的字母数字组成，路由器难以处理。由于这些原因，主机也可以使用所谓 **IP 地址**（IP address）进行标识。

我们将在第 4 章更为详细地讨论 IP 地址，但现在简略地介绍一下还是有必要的。一个 IP 地址由 4 个字节组成，并有着严格的层次结构。例如 121. 7. 106. 83 这样一个 IP 地址，其中的每个字节都被句点分隔开来，表示了 0 ~ 255 的十进制数字。我们说 IP 地址具有层次结构，是因为当我们从左至右扫描它时，我们会得到越来越具体的关于主机位于因特网何处的信息（即在众多网络的哪个网络里）。类似地，当我们从下向上查看邮政地址时，我们能够获得该地址位于何处的越来越具体的信息。

2.4.1 DNS 提供的服务

我们刚刚看到了识别主机有两种方式，通过主机名或者 IP 地址。人们喜欢便于记忆的主机名标识方式，而路由器则喜欢定长的、有着层次结构的 IP 地址。为了折中这些不同的偏好，我们需要一种能进行主机名到 IP 地址转换的目录服务。这就是**域名系统**（Domain Name System，DNS）的主要任务。DNS 是：①一个由分层的 **DNS 服务器**（DNS server）实现的分布式数据库；②一个使得主机能够查询分布式数据库的应用层协议。DNS 服务器通常是运行 BIND（Berkeley Internet Name Domain）软件［BIND 2012］的 UNIX 机器。DNS 协议运行在 UDP 之上，使用 53 号端口。

实践原则

DNS：通过客户-服务器模式提供的重要网络功能

与HTTP、FTP和SMTP协议一样，DNS协议是应用层协议，其原因在于：①使用客户-服务器模式运行在通信的端系统之间；②在通信的端系统之间通过下面的端到端运输协议来传送DNS报文。然而，在其他意义上，DNS的作用非常不同于Web应用、文件传输应用以及电子邮件应用。与这些应用程序不同之处在于，DNS不是一个直接和用户打交道的应用。相反，DNS是为因特网上的用户应用程序以及其他软件提供一种核心功能，即将主机名转换为其背后的IP地址。我们在1.2节就提到，因特网体系结构的复杂性大多数位于网络的“边缘”。DNS通过采用了位于网络边缘的客户和服务器，实现了关键的名字到地址转换功能，它还是这种设计原理的另一个范例。

DNS通常是由其他应用层协议所使用的，包括HTTP、SMTP和FTP，将用户提供的主机名解析为IP地址。举一个例子，考虑运行在某用户主机上的一个浏览器（即一个HTTP客户）请求URL www.someschool.edu/index.html页面时会发生什么现象。为了使用户的主机能够将一个HTTP请求报文发送到Web服务器www.someschool.edu，该用户主机必须获得www.someschool.edu的IP地址。其做法如下。

1）同一台用户主机上运行着DNS应用的客户端。

2）浏览器从上述URL中抽取出主机名www.someschool.edu，并将这台主机名传给DNS应用的客户端。

3）DNS客户向DNS服务器发送一个包含主机名的请求。

4）DNS客户最终会收到一份回答报文，其中含有对应于该主机名的IP地址。

5）一旦浏览器接收到来自DNS的该IP地址，它能够向位于该IP地址80端口的HTTP服务器进程发起一个TCP连接。

从这个例子中，我们可以看到DNS给使用它的因特网应用带来了额外的时延，有时还相当可观。幸运的是，如我们下面讨论的那样，想获得的IP地址通常就缓存在一个“附近的”DNS服务器中，这有助于减少DNS的网络流量和DNS的平均时延。

除了进行主机名到IP地址的转换外，DNS还提供了一些重要的服务：

- **主机别名**（host aliasing）。有着复杂主机名的主机能拥有一个或者多个别名。例如，一台名为relay1.west-coast.enterprise.com的主机，可能还有两个别名为enterprise.com和www.enterprise.com。在这种情况下，relay1.west-coast.enterprise.com也称为**规范主机名**（canonical hostname）。主机别名（当存在时）比主机规范名更加容易记忆。应用程序可以调用DNS来获得主机别名对应的规范主机名以及主机的IP地址。
- **邮件服务器别名**（mail server aliasing）。显而易见，人们也非常希望电子邮件地址好记忆。例如，如果Bob在雅虎邮件上有一个账户，Bob的邮件地址就像bob@yahoo.com这样简单。然而，雅虎邮件服务器的主机名可能更为复杂，不像yahoo.com那样简单好记（例如，规范主机名可能像relay1.west-coast.hotmail.com那样）。电子邮件应用程序可以调用DNS，对提供的主机名别名进行解析，以获得该主机的规范主机名及其IP地址。事实上，MX记录（参见后面）允许一个公司

的邮件服务器和 Web 服务器使用相同（别名化的）的主机名；例如，一个公司的 Web 服务器和邮件服务器都能叫作 enterprise. com。

- **负载分配**（load distribution）。DNS 也用于在冗余的服务器（如冗余的 Web 服务器等）之间进行负载分配。繁忙的站点（如 cnn. com）被冗余分布在多台服务器上，每台服务器均运行在不同的端系统上，每个都有着不同的 IP 地址。由于这些冗余的 Web 服务器，一个 IP 地址集合因此与同一个规范主机名相联系。DNS 数据库中存储着这些 IP 地址集合。当客户对映射到某地址集合的名字发出一个 DNS 请求时，该服务器用 IP 地址的整个集合进行响应，但在每个回答中循环这些地址次序。因为客户通常总是向 IP 地址排在最前面的服务器发送 HTTP 请求报文，所以 DNS 就在所有这些冗余的 Web 服务器之间循环分配了负载。DNS 的循环同样可以用于邮件服务器，因此，多个邮件服务器可以具有相同的别名。一些内容分发公司如 Akamai 也以更加复杂的方式使用 DNS［Dilley 2002］，以提供 Web 内容分发（参见 2.6.3 节）。

DNS 由 RFC 1034 和 RFC 1035 定义，并且在几个附加的 RFC 中进行了更新。DNS 是一个复杂的系统，我们在这里只是就其运行的主要方面进行学习。感兴趣的读者可以参考这些 RFC 文档和 Albitz 和 Liu 写的书［Albitz 1993］；亦可参阅文章［Mockapetris 1998］和［Mockapetris 2005］，其中［Mockapetris 1998］是回顾性的文章，它提供了 DNS 组成和工作原理的精细的描述。

2.4.2 DNS 工作机理概述

下面给出一个 DNS 工作过程的总体概括，我们的讨论将集中在主机名到 IP 地址转换服务方面。

假设运行在用户主机上的某些应用程序（如 Web 浏览器或邮件阅读器）需要将主机名转换为 IP 地址。这些应用程序将调用 DNS 的客户端，并指明需要被转换的主机名（在很多基于 UNIX 的机器上，应用程序为了执行这种转换需要调用函数 gethostbyname()）。用户主机上的 DNS 接收到后，向网络中发送一个 DNS 查询报文。所有的 DNS 请求和回答报文使用 UDP 数据报经端口 53 发送。经过若干毫秒到若干秒的时延后，用户主机上的 DNS 接收到一个提供所希望映射的 DNS 回答报文。这个映射结果则被传递到调用 DNS 的应用程序。因此，从用户主机上调用应用程序的角度看，DNS 是一个提供简单、直接的转换服务的黑盒子。但事实上，实现这个服务的黑盒子非常复杂，它由分布于全球的大量 DNS 服务器以及定义了 DNS 服务器与查询主机通信方式的应用层协议组成。

DNS 的一种简单设计是在因特网上只使用一个 DNS 服务器，该服务器包含所有的映射。在这种集中式设计中，客户直接将所有查询直接发往单一的 DNS 服务器，同时该 DNS 服务器直接对所有的查询客户做出响应。尽管这种设计的简单性非常具有吸引力，但它不适用于当今的因特网，因为因特网有着数量巨大（并持续增长）的主机。这种集中式设计的问题包括：

- **单点故障**（a single point of failure）。如果该 DNS 服务器崩溃，整个因特网随之瘫痪！
- **通信容量**（traffic volume）。单个 DNS 服务器不得不处理所有的 DNS 查询（用于为上亿台主机产生的所有 HTTP 请求报文和电子邮件报文服务）。

- **远距离的集中式数据库**（distant centralized database）。单个 DNS 服务器不可能“邻近”所有查询客户。如果我们将单台 DNS 服务器放在纽约市，那么所有来自澳大利亚的查询必须传播到地球的另一边，中间也许还要经过低速和拥塞的链路。这将导致严重的时延。
- **维护**（maintenance）。单个 DNS 服务器将不得不为所有的因特网主机保留记录。这不仅将使这个中央数据库庞大，而且它还不得不为解决每个新添加的主机而频繁更新。

总的来说，在单一 DNS 服务器上运行集中式数据库完全没有可扩展能力。因此，DNS 采用了分布式的设计方案。事实上，DNS 是一个在因特网上实现分布式数据库的精彩范例。

1. 分布式、层次数据库

为了处理扩展性问题，DNS 使用了大量的 DNS 服务器，它们以层次方式组织，并且分布在全世界范围内。没有一台 DNS 服务器拥有因特网上所有主机的映射。相反，这些映射分布在所有的 DNS 服务器上。大致说来，有 3 种类型的 DNS 服务器：根 DNS 服务器、顶级域（Top-Level Domain，TLD）DNS 服务器和权威 DNS 服务器。这些服务器以图 2-17 中所示的层次结构组织起来。为了理解这 3 种类型的 DNS 服务器交互的方式，假定一个 DNS 客户要决定主机名 www. amazon. com 的 IP 地址。粗略说来，将发生下列事件。客户首先与根服务器之一联系，它将返回顶级域名 com 的 TLD 服务器的 IP 地址。该客户则与这些 TLD 服务器之一联系，它将为 amazon. com 返回权威服务器的 IP 地址。最后，该客户与 amazon. com 权威服务器之一联系，它为主机名 www. amazon. com 返回其 IP 地址。我们将很快更为详细地考察 DNS 查找过程。不过我们先仔细看一下这 3 种类型的 DNS 服务器。

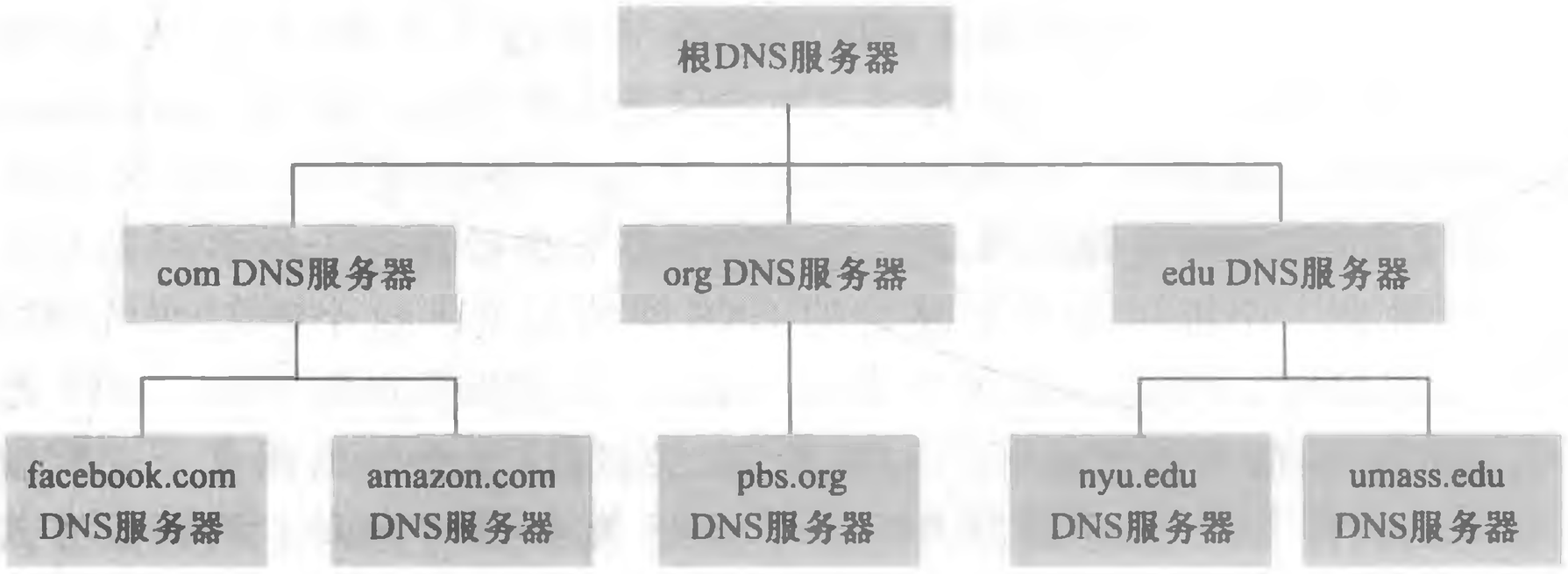

图 2-17　部分 DNS 服务器的层次结构

- **根 DNS 服务器**。有 400 多个根名字服务器遍及全世界。这些根名字服务器由 13 个不同的组织管理。根名字服务器的全部清单连同管理它们的组织及其 IP 地址可以在［Root Servers 2016］中找到。根名字服务器提供 TLD 服务器的 IP 地址。
- **顶级域（DNS）服务器**。对于每个顶级域（如 com、org、net、edu 和 gov）和所有国家的顶级域（如 uk、fr、ca 和 jp），都有 TLD 服务器（或服务器集群）。Verisign Global Registry Services 公司维护 com 顶级域的 TLD 服务器，Educause 公司维护 edu 顶级域的 TLD 服务器。支持 TLD 的网络基础设施可能是大而复杂的，［Osterweil 2012］对 Verisign 网络进行了很好的概述。所有顶级域的列表参见［TLD list

2016]。TLD 服务器提供了权威 DNS 服务器的 IP 地址。

- **权威 DNS 服务器**。在因特网上具有公共可访问主机（如 Web 服务器和邮件服务器）的每个组织机构必须提供公共可访问的 DNS 记录，这些记录将这些主机的名字映射为 IP 地址。一个组织机构的权威 DNS 服务器收藏了这些 DNS 记录。一个组织机构能够选择实现它自己的权威 DNS 服务器以保存这些记录；另一种方法是，该组织能够支付费用，让这些记录存储在某个服务提供商的一个权威 DNS 服务器中。多数大学和大公司实现和维护它们自己基本和辅助（备份）的权威 DNS 服务器。

根、TLD 和权威 DNS 服务器都处在该 DNS 服务器的层次结构中，如图 2-17 所示。还有另一类重要的 DNS 服务器，称为**本地 DNS 服务器**（local DNS server）。严格说来，一个本地 DNS 服务器并不属于该服务器的层次结构，但它对 DNS 层次结构是至关重要的。每个 ISP（如一个居民区的 ISP 或一个机构的 ISP）都有一台本地 DNS 服务器（也叫默认名字服务器）。当主机与某个 ISP 连接时，该 ISP 提供一台主机的 IP 地址，该主机具有一台或多台其本地 DNS 服务器的 IP 地址（通常通过 DHCP，将在第 4 章中讨论）。通过访问 Windows 或 UNIX 的网络状态窗口，用户能够容易地确定他的本地 DNS 服务器的 IP 地址。

主机的本地 DNS 服务器通常“邻近”本主机。对某机构 ISP 而言，本地 DNS 服务器可能就与主机在同一个局域网中；对于某居民区 ISP 来说，本地 DNS 服务器通常与主机相隔不超过几台路由器。当主机发出 DNS 请求时，该请求被发往本地 DNS 服务器，它起着代理的作用，并将该请求转发到 DNS 服务器层次结构中，我们下面将更为详细地讨论。

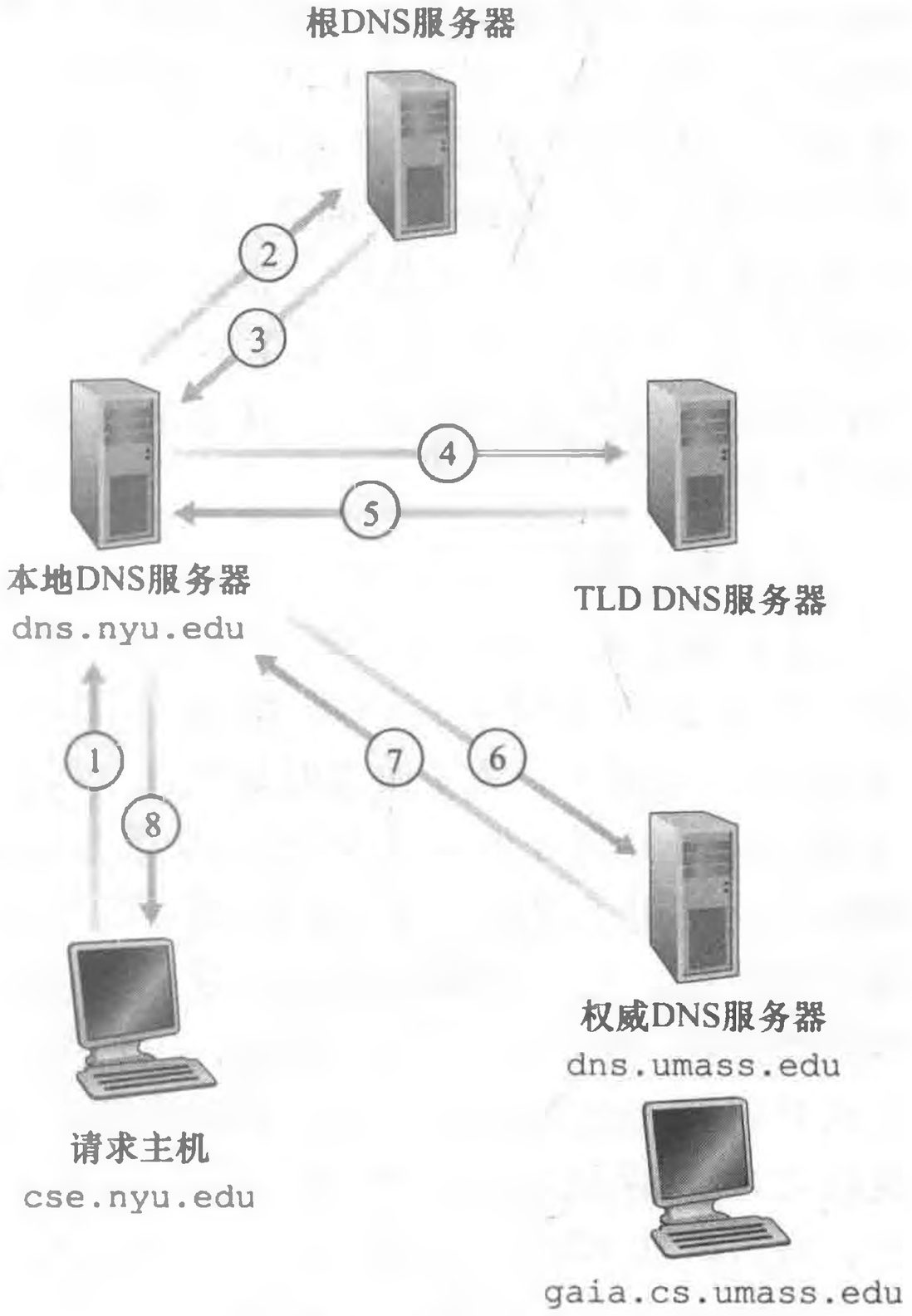

图 2-18 各种 DNS 服务器的交互

我们来看一个简单的例子，假设主机 cse. nyu. edu 想知道主机 gaia. cs. umass. edu 的 IP 地址。同时假设纽约大学（NYU）的 cse. nyu. edu 主机的本地 DNS 服务器为 dns. nyu. edu，并且 gaia. cs. umass. edu 的权威 DNS 服务器为 dns. umass. edu。如图 2-18 所示，主机 cse. nyu. edu 首先向它的本地 DNS 服务器 dns. nyu. edu 发送一个 DNS 查询报文。该查询报文含有被转换的主机名 gaia. cs. umass. edu。本地 DNS 服务器将该报文转发到根 DNS 服务器。该根 DNS 服务器注意到其 edu 前缀并向本地 DNS 服务器返回负责 edu 的 TLD 的 IP 地址列表。该本地 DNS 服务器则再次向这些 TLD 服务器之一发送查询报文。该 TLD 服务器注意到 umass. edu 前缀，并用权威 DNS 服务器的 IP 地址进行响应，该权威 DNS 服务器是负责马萨诸塞大学的 dns. umass. edu。最后，本地 DNS 服务器直接向 dns. umass. edu 重发查询报