DNS与BIND

第四版

Paul Albitz & Cricket Liu 著
雷迎春 龚奕利 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly & Associates, Inc. 授权中国电力出版社出版

中国电力出版社

图书在版编目（CIP）数据

DNS 与 BIND（第四版）/（美）阿尔比兹（Albitz, P.），刘（Liu, C.）著；雷迎春，龚奕利译．- 北京：中国电力出版社，2002.8
书名原文：DNS and BIND, Fourth Edition
ISBN 7-5083-0980-4

I. D... II. ①阿 ... ②刘 ... ③雷 ... ④龚 ... III. 网络服务器 IV. TP368.5
中国版本图书馆 CIP 数据核字（2002）第 053051 号

北京市版权局著作权合同登记
图字：01-2002-2274 号

书　　名 / DNS 与 BIND（第四版）
书　　号 / ISBN 7-5083-0980-4
责任编辑 / 程璐
封面设计 / Edie Freedman，张健
出版发行 / 中国电力出版社（www.infopower.com.cn）
地　　址 / 北京三里河路 6 号（邮政编码 100044）
经　　销 / 全国新华书店
印　　刷 / 北京市地矿印刷厂
开　　本 / 787 毫米× 1092 毫米　16 开本　42 印张　633 千字
版　　次 / 2002 年 8 月第一版　2002 年 8 月第一次印刷
印　　数 / 0001-5000 册
定　　价 / 69.00 元（册）

作者简介

Paul Albitz 现为惠普公司的软件工程师。他获得了威斯康星大学理学学士学位和普渡大学的理学硕士学位。

Paul 从事与 HP-UX 7.0 和 8.0 有关的 BIND 工作。工作期间，Paul 开发了用于管理 hp.com 域的工具。此后，Paul 就一直做着惠普 DesignJet 绘图仪互联和惠普的 OfficeJet 多功能传真子系统的工作。加盟惠普之前，Paul 在普渡大学计算机系担任系统管理员。作为系统管理员，Paul 使用了比与 4.3 BSD 一起发行的 BIND 更早版本的 BIND。现在，Paul 与他的妻子 Katherine 住在加州圣迭戈。

Cricket Liu 就读于加州伯克利分校，那里是自由演讲的阵营，有不受限制的 Unix 和便宜的比萨饼。他毕业后开始为惠普公司工作，一口气干了九年。

Cricket 在 Loma Prieta 地震后开始管理 *hp.com* 域。地震使得域的管理不得不从惠普的实验室搬到公司的办公室。他担任 *hostmaster@hp.com* 三年多，然后加入惠普的专业服务组织，创建了惠普的 Internet 咨询流程。

1997 年，Cricket 离开惠普，并和他的朋友（现在的合著者）Matt Larson 组建了 Acme Byte & Wire，一个 DNS 咨询和培训公司。Network Solutions 在 2000 年 6 月收购了 Acme，并在同一天与 VeriSign 合并。Cricket 现在是 VeriSign 全球注册服务的 DNS 产品管理主任。

Cricket 和他的妻子 Paige、儿子 Walt 以及两只爱犬 Annie 和 Dakota 住在科罗拉多州。在暖暖的周末下午，你也许能够看到他们正在荡秋千。

封面介绍

本书封面上的昆虫是蝗虫，它们遍及世界的每个角落。在北美5000多种昆虫中，蝗虫有100多种。蝗虫呈褐绿色，体长从0.5英寸到4英寸不等，翼展可达6英寸。它们的身体分为三部分：头、胸和腹部。

雄性蝗虫用后腿和前翅发出一种“ 唧唧 ”的声音。它们的后腿上有一排小的突起，摩擦前翅上硬化的血管，产生的震动听起来就像拉动弓弦的声音。

蝗虫是主要的田间害虫，特别是当它们成群结队的时候。一只蝗虫每天消耗30毫克的食物。如果蝗虫的密度为每平方码50只及以上时（爆发蝗灾时通常能达到这个密度），每英亩蝗虫的消耗同一头牛的消耗相当。除了侵食叶子外，蝗虫还袭击植物柔弱的部位，导致茎干折断，破坏植物的生长。

前言

到目前为止，你可能仍然对 DNS（Domain Name System，域名系统）所知甚少，但是无论你何时使用 Internet，都会用到 DNS。每次发送电子邮件或是在网上冲浪，你都必须依赖 DNS。

作为普通人，我们都宁愿记计算机的名字，而计算机却喜欢用数字（即，主机 IP 地址）来彼此称呼。在互联网络上，这样的地址是一个 32 位的数字，或者说是一个介于 0 到大约 40 亿之间的数字（注 1）。对于计算机来说这是很容易记住的，因为计算机的内存很适合存储数字，而对于我们人来说这就不那么好记了。从电话簿中随机挑出 10 个电话号码，试试记住它们。不容易吧？然后在每个电话号码前加上随机的区号，这就和记住 10 个任意的互联网络地址差不多难了。

这就是我们需要 DNS 的部分原因。DNS 负责主机名字之间和互联网络地址之间的映射，前者我们人类会觉得很方便，而后者是由计算机来处理的。实际上，DNS 是 Internet 上的一个标准机制，用来发布和访问关于主机的各种信息，而不只是地址。实际上几乎所有的网间互联软件都在使用 DNS，包括电子邮件、远程终端程序（如，Telnet）、文件传输程序（如，FTP）以及 Web 浏览器（如，网景的 Navigator 和微软的 Internet Explorer）。

注 1： 对于 IPv6 而言，它很快就是 128 位长的了，也就是介于 0 到一个 39 位的十进制数之间。

DNS的另一个重要特性就是它使主机信息在Internet上随处可得。将主机信息按照某种格式存为文件，放在某台计算机上，并只对那台计算机的用户有用。DNS则提供了一种远程检索信息的方式，你能从网络上任何一个地方查找信息。

除此之外，DNS还能将主机信息的管理分布到许多地点和组织。你不需要将数据提交给某个中心，或定期地检索中心的数据库，你只要保证你的名字服务器（name server）上称为区（zone）的那一部分是最新的即可。你的名字服务器会使网络上其他的名字服务器都能访问到你区中的数据。

因为数据库是分布式的，所以系统还需要能够通过搜索一些可能的位置来确定你要查找的数据在哪里。DNS使得名字服务器能够很聪明地在数据库之中查找，并找到任何区中的数据。

当然，DNS也有一些问题。例如，出于冗余考虑，系统允许多个名字服务器存储一个区的同样数据，这就会引发这些服务器上区数据之间的一致性问题。

不过关于DNS最糟糕的问题则是尽管它在Internet上广泛使用，却很少有关于如何管理和维护DNS方面的资料。Internet上大多数管理员使用的是商家认为应该提供的资料，再就是从相关领域的Internet邮件列表和Usenet新闻组中搜集到的一些信息。

缺乏资料就意味着，对这种当今Internet上最关键的服务的理解要么是从一个管理员传授给另一个管理员，就像祖传秘方；要么是从一个个互不相识的程序员和工程师那里重复搜集取得，并且新的系统管理员犯着无数人犯过的错。

我们写这本书的目的就是为了帮助解决这一问题。我们意识到你们当中并非所有人都想成为DNS专家。毕竟，大多数人除了管理一个区或名字服务器之外还有许多其他事情要做：系统管理、网络工程或软件开发。要一个人只负责DNS是不可想像的。我们会试着给你足够的信息，让你无论是运行一个小的区还是管理一个跨国的庞然大物，无论是照顾一个名字服务器还是管理上百个名字服务器，都只做需要做的事。你可以现在需要知道多少就读多少，等需要知道更多的时候，再回来接着读。

DNS是个很大的话题——至少大到需要两个作者，不过我们会尽力使它易于理解。本书的前两章是理论上的概述以及一些实际信息，余下来的章节讲的都是些核心细节。我们首先提供了一个路线指南，有了它，你可以根据自己的工作或兴趣选择合适的学习路线。

谈到实际的DNS软件时，我们主要讲的是BIND（Berkeley Internet Name Domain）软件，它是DNS规范的一种最为常见的实现（也是我们所知道的最好的）。我们尽力将自己使用BIND管理和维护区的经验浓缩在本书当中。（我们所管理的一个区曾经是Internet上最大的区，不过那已经是很久以前的事了。）只要有可能，我们就会给出在管理中实际用到的程序，为了提高速度和效率，其中许多都用Perl重写了。

如果你还是个新手的话，我们希望本书能帮助你熟悉DNS和BIND；如果你已经对DNS有所了解，我们希望能增进你的理解。即使你对DNS已经了如指掌，我们还是希望能给你一些有价值的见解和经验。

版本

本书的第四版主要是讲新的9.1.0和8.2.3版的BIND，同时也涉及较早的4.9版。虽然9.1.0和8.2.3是我们撰写本书时最新的BIND版本，但是许多供应商的Unix版本中还不包括它们，部分原因是由于这两个版本最近才刚刚发布，同时也因为许多供应商对于使用这些新软件还是抱着小心谨慎的态度。我们也会偶尔提到其他版本的BIND，特别是4.8.3版，因为许多供应商的Unix产品中包括的代码还是基于这个老版本的BIND。如果某个特性只适用于4.9、8.2.3或9.1.0版，或者在不同版本中使用情况有所不同的话，我们将会分别讨论各种版本。

我们在例子中大量使用了*nslookup*这种名字服务器实用程序。我们所使用的*nslookup*是同BIND 8.2.3代码封装在一起的那个版本。较早版本的*nslookup*也提供了8.2.3中*nslookup*大部分的功能，不过并非全部（注2）。在例子中，我们尽量使用对大多数*nslookup*都通用的命令，如果无法做到这一点，会特别注明的。

第四版新增加的内容

除了更新内容以涵盖最新的BIND版本之外，在第四版中我们还增加了相当多的新东西：

注2： 在BIND 9中封装的*nslookup*版本也是如此。详情请见第十二章。

- 更多的动态更新和NOTIFY内容，包括带签名的动态更新和BIND 9当中新的更新策略（update-policy）机制，参见第十章。
- 增量区传送（incremental zone transfer），参见第十章。
- 支持条件转发的转发（forward）区，参见第十章。
- 使用新的A6和DNAME记录以及位串标号的IPv6正向和反向地址映射，参见第十章的最后部分。
- 一种新的事务认证机制 —— 事务签名（transaction signature），又称为TSIG，参见第十一章。
- 扩展了保护名字服务器安全的部分，参见第十一章。
- 扩展了有关Internet防火墙的部分，参见第十一章。
- 包括了一种新的对区数据进行数字签名的机制—— DNS安全性扩展（DNS Security Extension），简称DNSSEC，参见第十一章。
- 专门有一节讲述Windows 2000客户机、服务器和域控制器（Domain Controller）与BIND的兼容问题，参见第十六章。

组织

本书的内容或多或少是按照区及其管理员的不断发展过程来组织的。第一、二章讨论了关于DNS的理论。第三章到第六章帮助你决定是否要建立你自己的区，还讲述了如果选择建立了自己的区，又该如何来做。中间的几章，第七章到第十一章讲的是如何维护区、如何配置主机使之使用指定的名字服务器、如何规划区的发展、如何创建子域，以及如何保护名字服务器。最后几章，第十二章到第十六章，讲的是一些有关排错工具、常见问题，以及使用解析器库例程编程的技术和技巧。

下面是关于每一章更详细的介绍：

- 第一章“ 背景 ”, 提供了一些历史资料, 讨论促使DNS发展的问题, 然后是DNS理论的概述。
- 第二章“ DNS是如何工作的？ ”，更详细地回顾了DNS理论，包括DNS名字

空间、域、区和名字服务器的组织。另外还介绍了一些很重要的概念，比如名字解析和缓存。

- 第三章“我该从哪里开始？”，谈到了如果你还没有DNS软件的话，该如何获取BIND，以及得到之后又该怎么办：如何确定你的域名是什么，以及如何同区的授权组织联系。
- 第四章“建立BIND”，详细介绍了如何建立你的头两个BIND名字服务器，包括创建你的名字服务器数据库、启动你的名字服务器和检查它们的操作。
- 第五章“DNS和电子邮件”，讲的是DNS的MX记录，它允许管理员指定其他主机来处理发往给定目的主机的邮件。这一章涉及对各种网络和主机的邮件路由策略，包括有Internet防火墙的网络和没有直接连到Internet的主机。
- 第六章“配置主机”，解释了如何配置一个BIND解析器。我们还将注明许多常见Unix厂商的解析器实现的特性，同时还会谈到Windows 95、NT和2000的解析器。
- 第七章“维护BIND”，讲述了为保证区的平稳运行，管理员所需要做的定期维护工作，比如说检查名字服务器是否正常以及它的授权状况。
- 第八章“扩展你的域”，涉及了如何规划和发展你的区，包括如何扩大、如何为移动用户和故障做准备。
- 第九章“担当父域”，探讨了成为父区的快乐。我们解释了何时成为一个父区（创建子域）、如何命名你的孩子以及如何创建（!）和监控它们。
- 第十章“高级特性”，讲述了一些不太常用的名字服务器配置选项，它们能帮助你优化名字服务器的操作，使管理更轻松。
- 第十一章“安全”，讲述了如何保护名字服务器，以及如何配置名字服务器同Internet防火墙一起工作，此外还讲述了DNS的两个新增的安全性功能：DNS安全性扩展和事务签名。
- 第十二章“nslookup和dig”，详细介绍了最常用的调试DNS的工具，包括从远程名字服务器挖掘模糊信息的技术。
- 第十三章“阅读BIND的调试输出”，这些输出开始时就像是罗赛塔石碑上的文字那样神秘。这一章将会有助于你理解那些BIND显示的神秘信息的意义，从而使你能更好地了解名字服务器。

- 第十四章“ DNS和BIND排错 ”，涉及了许多常见的DNS和BIND问题及其解决方法，然后还讲述了一些不太常见、较难诊断的情况。
- 第十五章“ 用解析器和名字服务器的库例程编程 ”，演示了如何在一个C程序或Perl脚本中使用BIND的解析器例程来查询名字服务器和从中检索数据。我们还加入了一个有用的程序（希望如此！），它可以用来检查名字服务器正常与否以及授权情况。
- 第十六章“ 其他问题 ”，将所有松散的头绪连在一起。我们讲到了DNS通配符、通过拨号断断续续地连接到Internet的主机和网络、网络名字编码、试验性的记录类型以及 Windows 2000。
- 附录一“ DNS消息格式和资源记录 ”，一个字节一个字节地分解DNS查询和响应中使用的格式，另外还有当前定义的资源记录类型的综合列表。
- 附录二“ BIND兼容性真值表 ”，该表列举了常用BIND版本的最重要特性。
- 附录三“ 在Linux上编译和安装BIND ”，包含了关于如何在Linux上编译BIND 8.2.3的一步步指令。
- 附录四“ 顶级域 ”，列出了目前Internet域名空间中的顶级域名。
- 附录五“ BIND名字服务器和解析器配置 ”，总结了用来配置名字服务器和解析器的每一个参数的语法和语义。

读者

本书主要是为管理区以及一个或多个名字服务器的系统和网络管理员而写的，但是它也适用于网络工程师、邮件管理员以及其他一些人。不过，不同读者对各个章节的兴趣也不一样，你不一定要读完所有的十六章才能找到与你工作相关的信息。我们希望下面这个路线指南能帮助你计划好自己的阅读路线。

建立自己第一个区的系统管理员要了解DNS的理论，应该读第一、二章；要了解如何开始和选择一个好域名，应该读第三章；要学习第一次如何建立区，应该读第四章和第五章。第六章解释了如何配置主机使其使用新名字服务器。接下来就该读第七章了，这一章介绍了如何使他们的域“ 有血有肉 ”，比如建立其他的名字服务器和添加其他的区数据。第十二、十三和十四章讲述了排错工具和技术。

有经验的管理员可以读读第六章，能够学习到如何在不同主机上配置DNS解析器，读第七章能学习到区的维护方面的知识。第八章包括了如何规划区的扩大和发展，这对于较大区的管理员尤为有价值。第九章解释了如何做一个父域 —— 创建子域，这对正在考虑要向大区发展的管理员来说是很有必要一看的。第十章包括了许多BIND 8.2.3和9.1.0名字服务器的新的高级特性。第十一章涉及了保护名字服务器，这对有经验的管理员来说是很有意义的。第十二到十四章描述了排错的工具和技术，即使是对高级管理员来说也是值得一读的。

没有完全连接到Internet的网络的系统管理员应该读一读第五章，学习一下如何在这类网络上配置邮件，还应该读一读第十一章，学习一下如何建立一个独立的DNS基础设施。

不直接负责域的网络管理员还是应该读一下第一、二章，了解一下DNS理论，然后是第十五章，详细了解一下如何用BIND解析器库例程编程。

邮件管理员应该读第一、二章，了解一下DNS理论，还有第五章，学习DNS和电子邮件是如何共存的。第十二章描述了*nslookup*和*dig*，这将有助于邮件管理员从域名空间中抽取邮件路由信息。

感兴趣的读者可以读一读第一、二章，学习学习DNS理论，除此之外，想读什么就读什么！

注意，我们假设你很熟悉基本的Unix系统管理以及TCP/IP网络工作原理，并且能够使用简单的shell脚本和Perl来编程。除此之外，不要求你有任何其他专业知识。当提到一个新的术语或概念的时候，我们会尽力定义或解释的。只要可能，我们将用Unix（以及现实世界）来与之相类比，来帮助你理解。

获取示例程序

本书中的示例程序可以通过FTP从下面的URL获得：

ftp://ftp.uu.net/published/oreilly/nutshell/dnsbind/dns.tar.Z
ftp://ftp.oreilly.com/published/oreilly/nutshell/dnsbind/

无论从何处下载，要解开压缩文件都可以输入：

```
%  zcat dns.tar.Z | tar xf -
```

系统 V 中则要求输入下面的 *tar* 命令：

```
%  zcat dns.tar.Z | tar xof -
```

如果你的系统上没有 *zcat*，可以分别使用 *uncompress* 和 *tar* 命令。

如果不能直接通过 Internet 得到这些例子，但可以收发邮件，你可以用 *ftpmail* 来获取它们。要想了解如何使用 *ftpmail*，发送电子邮件到 *ftpmail@online.oreilly.com*，不用写主题，只要在消息的正文中写一个单词“ help ”即可。

建议与评论

本书的内容都经过测试，尽管我们做了最大的努力，但错误和疏忽仍然是在所难免的。如果你发现有什么错误，或者是对将来的版本有什么建议，请通过下面的地址告诉我们：

美国：

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472

中国：

100080 北京市海淀区知春路 49 号希格玛公寓 B 座 809 室
奥莱理软件（北京）有限公司

本书有一个网页，上面附有勘误表、实例和其他附加信息。可以访问以下网页：

http://www.oreilly com/catalog/devbioinfo/

如评论或探讨技术问题；可发 Email 至：

bookquestions@oreilly.com

info@mail.oreilly.com.cn

要查找有关本书更多的内容、评论、软件、资源中心以及O'Reilly网络，请访问我们的网站：

http://www.oreilly.com
http://www.oreilly.com.cn

本书中所使用的约定

我们用如下所示的字体和格式惯例来表示Unix命令、实用程序和系统调用：

- 脚本或配置文件的摘录以等宽体来显示：

```
if test -x /usr/sbin/named -a -f /etc/named.conf
then
    /usr/sbin/named
fi
```

- 带命令行输入和相应输出的交互式会话的例子以等宽体显示，其中需要用户输入的用黑体显示：

```
%      cat /var/run/named.pid
78
```

- 如果命令必须由超级用户（root）输入，那么我们在前面使用#符号：

```
#    /usr/sbin/named
```

- 代码中可替代的项以等宽斜体显示。
- 在一段文字中出现的域名、文件名、函数、命令、Unix联机手册和摘自代码片段的编程语言的元素也用斜体表示。

引语

每一章开头的引语是从古登堡计划电子版的千年支点中Lewis Carroll的2.9版《爱丽丝漫游仙境》和1.7版《镜中世界》两部小说中摘录的。第一、二、五、六、八和十四章中的引语来自于《爱丽丝漫游仙境》；第三、四、七、九、十、十一、十二、十三、十五和十六章的引语则来自于《镜中世界》。

致谢

本书的作者非常感谢Ken Stone，Jerry McCollom，Peter Jeffe，Hal Stern，Christopher Durham，Bill Wisner，Dave Curry，Jeff Okamoto，Brad Knowles，K. Robert Elz和Paul Vixie为本书做出的宝贵贡献。我们同样要感谢我们的审阅人员Eric Pearce，Jack Repenning，Andrew Cherenson，Dan Trinkle，Bill LeFebvre和John Sechrest提出的批评和建议。没有他们的帮助，本书不可能是你现在看起来的样子（将会短很多！）。

对于第二版，作者还将感谢下列复审人员：Dave Barr，Nigel Campbell，Bill LeFebvre，Mike Milligan和Dan Trinkle。对于第三版，作者要向他们的技术审校“梦之组”致敬：Bob Halley，Barry Margolin和Paul Vixie。对于第四版，作者要感谢Kevin Dunlap，Edward Lewis和Brian Wellington，他们是最棒的复审小组。

Cricket特别要感谢他的前任经理Rick Nordensten，一个现代HP经理的典范，他阅读了本书的第一版；还要感谢长期容忍他偶尔暴躁的邻居；当然还要感谢他的妻子Paige，她不仅给予了不懈的支持，还容忍了他在她睡觉时敲键盘。关于第二版，Cricket要对他的前任经理Regina Kershner和Paul Klouda对他互联网工作的支持，说一声“谢谢你们”。对于第三版，Cricket向他的伙伴Matt Larson在共同开发Acme Razor中的工作表示深切的感激。关于第四版，Cricket要感谢他忠实的宠物伙伴Dakota和Annie赠予他的吻和陪伴，还有Walter B.在办公室里时不时地伸一下脑袋检查他爸爸的工作。Paul要感谢他妻子Katherine的耐心、多次的评论，并证明她可以在业余时间比她丈夫写完他那半本书还要快地做好一床棉被。

我们同样要感谢在O'Reilly & Associates工作的同仁的艰苦工作和耐性。特别要感谢我们的编辑Mike Loukides（第一至第三版）和Debra Cameron（第四版），以及为这几版书工作过的数不清的人们：Nancy Kotary，Ellie Fountain Maden，Robert Romano，Steven Abrams，Kismet McDonough-Chan，Seth Maislin，Ellie Cutler，Mike Sierra，Lenny Muellner，Chris Reilley，Emily Quill，Anne-Marie Vaduva和Brenda Miller。还要感谢Jerry Peek给我们的各种其他帮助，还有Tim O'Reilly，是他激励我们最终将本书付梓。

谢谢，Edie，为我们在封面加上了蟋蟀！

目录

本章内容：

- Internet 简史
- Internet 和 internet
- DNS 简述
- BIND 的历史
- 我一定要使用 DNS 吗？

第一章

背景

白兔戴上眼镜，问道：“ 陛下，请问
我们应该从哪里开始呢？ ”
国王严肃地说：“ 从开始处开始，
一直到结束，然后停止。”

要想理解 DNS（域名系统），了解一点儿 ARPAnet 的历史是很必要的。DNS 就是针对 ARPAnet 的一些特殊问题发展而来的，而现在，源自 ARPAnet 的 Internet 仍然是 DNS 的主要使用者。

如果你已经使用 Internet 多年，那就可以跳过这一章。如果你没有，我们希望它能使你了解到足够的背景，理解是什么推动了 DNS 的发展。

Internet 简史

20 世纪 60 年代末，美国国防部高级研究计划署，也就是 ARPA（后来的 DARPA），开始资助试验性的广域计算机网络，称为 ARPAnet，它连接了全美重要的研究机构。建立 ARPAnet 的初衷是使政府机构能共享昂贵或稀缺的计算资源。然而从一开始，ARPAnet 的使用者们就利用网络来合作。这些合作包括在今天看来非常普通的共享文件和软件、交换电子邮件，以及通过共享远程计算机来联合开发和研究。

TCP/IP（传输控制协议 /Internet 协议）协议是在 20 世纪 80 年代初发展起来的，并

迅速成为ARPAnet的标准主机网络协议。流行的加州大学伯克利分校的BSD Unix操作系统采用了这个协议族，这为普及Internet连接起了很大的作用。BSD Unix操作系统对大学来说实际上是免费的。这就意味着，对于那些更多的、原来没有连接到ARPAnet上的机构来说，实现Internet连接 —— 以及连接到ARPAnet，一下子变得便宜而可行。许多连接到ARPAnet的计算机已经连在局域网（LAN）上了，很快地，局域网上其他计算机也开始通过ARPAnet相互通信。

这个网络从只有屈指可数的几台主机发展到拥有成百上千的主机。原来的ARPAnet成为基于TCP/IP协议的局域网和区域联合网络的主干，被称为Internet。

然而在1988年，DARPA决定结束试验。国防部开始拆除ARPAnet。由美国国家科学基金会资助的另一个网NSFNET，取而代之成为Internet的主干。

后来在1995年春，Internet完成了从由公共NSFNET作为主干网到使用多个商业主干网的转变，这个主干网由MCI、Sprint这样的长途电信运营商和久负盛名的商业网PSINet、UUNet共同组成。

今天Internet连接了世界各地数以百万计的主机。实际上，世界上的非PC计算机有相当一部分是连在Internet上的。一些新建的商业网具有每秒数千兆的容量，这个带宽是原来ARPAnet的千万倍。每天都有成千上万的人在使用网络进行通信和协作。

Internet和internet

在大多数人看来，Internet和internet是一个意思。从字面上看，这两个词的差别很小：一个总是首字母大写，另一个则不是。然而它们的含义却截然不同。Internet（因特网），首字母I大写，指的是由ARPAnet开始，延续到今天成为所有直接或间接连接到美国商业主干网的、使用TCP/IP协议的网络的联合网络。从内部来看，它实际上是许多不同网络 —— 商业TCP/IP主干网、公司和美国政府TCP/IP网络以及其他国家的TCP/IP网络，通过路由器和高速数字线路相互连接而成的。

另一方面，小写的internet（互联网络），仅仅是指任何由多个小网络使用相同网络协议组成的网络。互联网络其实既不一定要连接到Internet，也不一定要使用TCP/

IP作为它的网络协议。实际上，就有一些独立的公司互联网络存在，像基于施乐XNS协议的互联网络和基于 DECnet 协议的互联网络。

一个相对较新的术语 intranet（内联网），其实就是基于 TCP/IP 协议的互联网络（internet）的通俗说法，用以强调它把在Internet上发展和流行起来的技术应用于公司的内部网中。另外，extranet是指连接伙伴公司，或将一个公司连接到其分销商、供应商和客户的基于 TCP/IP 的互联网络。

DNS 的历史

整个 20 世纪 70 年代，ARPAnet 只是一个拥有几百台主机的很小很友好的网络。仅仅需要一个名为*HOSTS.TXT*的文件就能容纳所有需要了解的主机信息：它包含所有连接到 ARPAnet 的主机的名字—地址映射（name-to-address mapping）。与它极为相似的 Unix 主机表*/etc/hosts*，就是从*HOSTS.TXT*变化而来的（主要是从中删除了 Unix 不用的字段）。

*HOSTS.TXT*文件是由SRI的网络信息中心（Network Information Center，简称NIC）负责维护，并且从一台主机 SRI-NIC 上分发到整个网络（注 1）。ARPAnet 的管理员们通常通过电子邮件的方式将他们的变更通知NIC，同时还定期FTP到SRI-NIC，以获取最新的 *HOSTS.TXT* 文件。每周进行一、两次更新，将这些变更汇编成新的 *HOSTS.TXT* 文件。但是随着 ARPAnet 的增长，这样的方法行不通了。*HOSTS.TXT* 文件大小的增长与 ARPAnet 上主机数量的增加成正比。更重要的是，由于更新过程而引起的网络流量的增加更快：每增加一台主机不仅仅意味着要在*HOSTS.TXT*文件中增加一行，更隐含着其他主机需要从 SRI-NIC 进行更新。

当 ARPAnet 采用 TCP/IP 协议后，网络规模爆炸性地增长。现在由一台主机来管理 *HOSTS.TXT* 文件就出现了下列的许多问题：

流量和负载

由于分发文件所引起的网络流量和处理器负载使得SRI-NIC的线路变得不堪重负。

注 1： SRI 是位于加州 Menlo Park 的斯坦福研究院（Stanford Research Institute）的前身，它从事许多不同领域的研究，其中包括计算机网络。

名字冲突

在*HOSTS.TXT*文件中任何两台主机都不能重名。虽然NIC能以某种方式在分配地址时保证惟一性，但是它对主机命名是没有权利过问的。如果有人添加了重名的主机，打乱整个设计，系统也无能为力。例如，如果有人添加了一台与主邮件分发器（mail hub）同名的主机，这将破坏ARPAnet上许多正常的邮件服务。

一致性

在不断扩张的网络上，要想维持*HOSTS.TXT*文件的一致性变得越来越难。新的*HOSTS.TXT*文件还没有到达庞大的ARPAnet最边缘时，网络另一端的主机地址已经改变，或是又有用户要添加新主机了。

关键的问题是*HOSTS.TXT*文件的结构并不是很好。颇具讽刺意味的是，ARPAnet作为一项试验的成功却导致*HOSTS.TXT*文件的失败和落伍。

ARPAnet的管理者们开始投入研究，为*HOSTS.TXT*文件寻求继任者。他们的目标是创造一个系统，它能解决单一主机表系统本身所固有的问题。新系统应该允许本地管理数据，同时数据又能被整个网络所使用。管理的分散化能消除单一主机的瓶颈，缓解流量问题。同时本地管理能使及时更新数据变得简单得多。新系统应该使用层次结构的名字空间来为主机命名，从而确保名字的惟一性。

Paul Mockapetris，当时在南加州大学的信息科学所，负责设计新系统的体系结构。1984年，他发布了描述DNS的RFC 882和883。RFC 882和883后来被RFC 1034和1035所取代，也就是目前的DNS规范（注2）。目前，还有许多其他RFC补充RFC 1034和1035的内容，它们描述了DNS潜在的安全问题、实现问题、管理问题、动态更新名字服务器的机制和保护区数据，等等。

DNS简述

实际上，DNS是一个分布式数据库。它允许对整个数据库的各个部分进行本地控制；

注2： RFC就是请求注解（Request for Comments）文档，是推荐Internet新技术的非正式过程的一部分。RFC通常免费分发，包含对新技术的技术性描述，通常是为实现者服务的。

同时整个网络也能通过客户—服务器方式访问每个部分的数据。借助备份和缓存机制，DNS 将具有强壮性和足够的性能。

被称为名字服务器（name server）的程序构成了 DNS 客户—服务器机制的服务器一端。名字服务器包含数据库中某些部分的信息，并使得被称为解析器（resolver）的客户端程序能访问到这些信息。解析器往往是创建查询请求并通过网络将它们发送到名字服务器的库例程。

DNS 数据库的结构如图 1-1 所示，同 Unix 文件系统的结构非常相似。整个数据库（或文件系统）用图来表示就像一棵倒着的树，根节点在顶端。树中每个节点都有一个文本标号，可用来标识该节点与父节点的相对关系。这和文件系统中的相对路径名大体相似，如*bin*目录。空标号“ ”为根节点所保留。在本书中，根节点写作“ . ”；在 Unix 文件系统中，根用“ / ”表示。

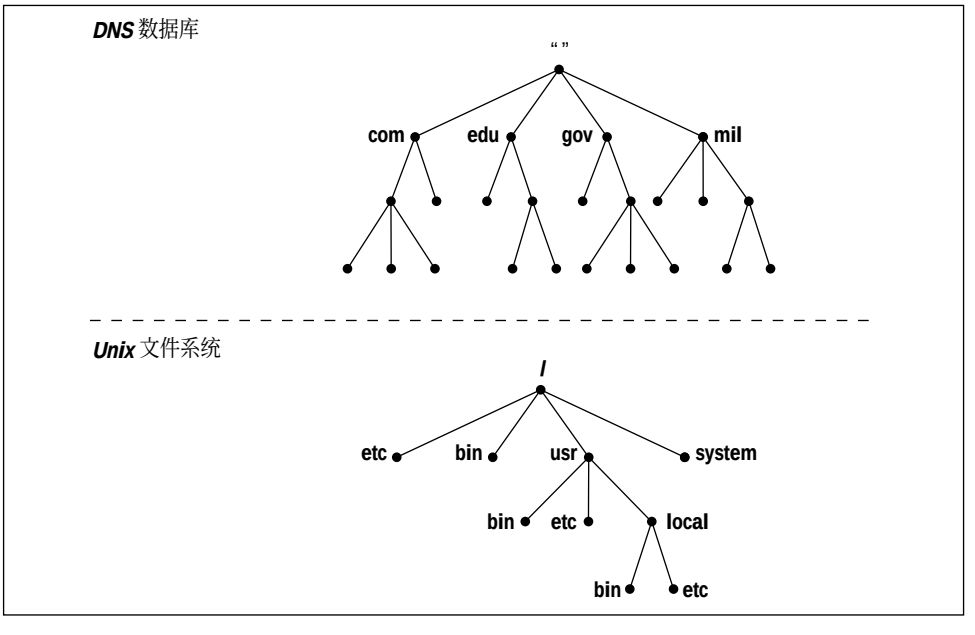


图 1-1　DNS 数据库和 Unix 文件系统的比较

每个节点同时也是整棵树中新子树的根。每棵子树都代表整个数据库的一个部分（partition），对应到Unix文件系统中的一个目录或是DNS中的一个域。每个域或目

录又能进一步被划分成更多的部分，在DNS中这叫做子域（subdomain），而在文件系统中则称为子目录。子域和子目录一样，在图中被画成它们父域的孩子。

与目录一样，每个域都有惟一的名字。域的域名（domain name）标识它在数据库中的位置，就像目录的绝对路径名标识它在文件系统中的位置一样。在DNS中，域名是从域根所在节点到整个树的根节点的标号的顺序连接，标号间用“.”来分隔。在Unix 文件系统中，目录的绝对路径名是从根到叶子相对名字的序列（与DNS刚好相反，见图 1-2），名字间用斜杠“/”来分隔。

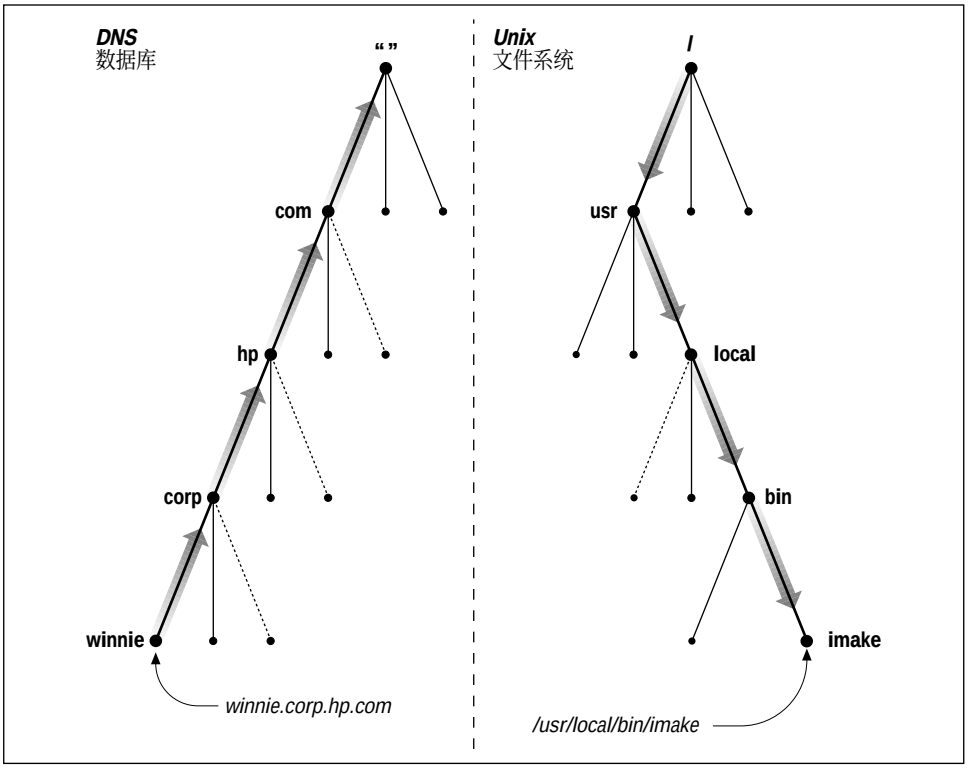


图 1-2　在 DNS 和 Unix 文件系统中读名字

在DNS中，每个域都能被分成许多子域，而这些子域又可由不同的组织来管理。举个例子来说，Network Solutions负责管理*edu*（educational）域，却授权加州大学伯克利分校管理*berkeley.edu*子域（见图1-3）。这就好像远程安装（mount）一个文件系统：文件系统中的某些目录可能实际上是其他主机上的文件系统，只是从远程

主机上安装过来的。譬如（也见图 1-3）主机 *winken* 的管理员要负责管理在本地主机上以目录 */usr/nfs/winken* 出现的文件系统。

将 *berkeley.edu* 授权给加州大学伯克利分校要创建一个新的区（zone），区是域名空间中可以自治管理的单位。现在区 *berkeley.edu* 独立于 *edu*，它包括所有以 *berkeley.*

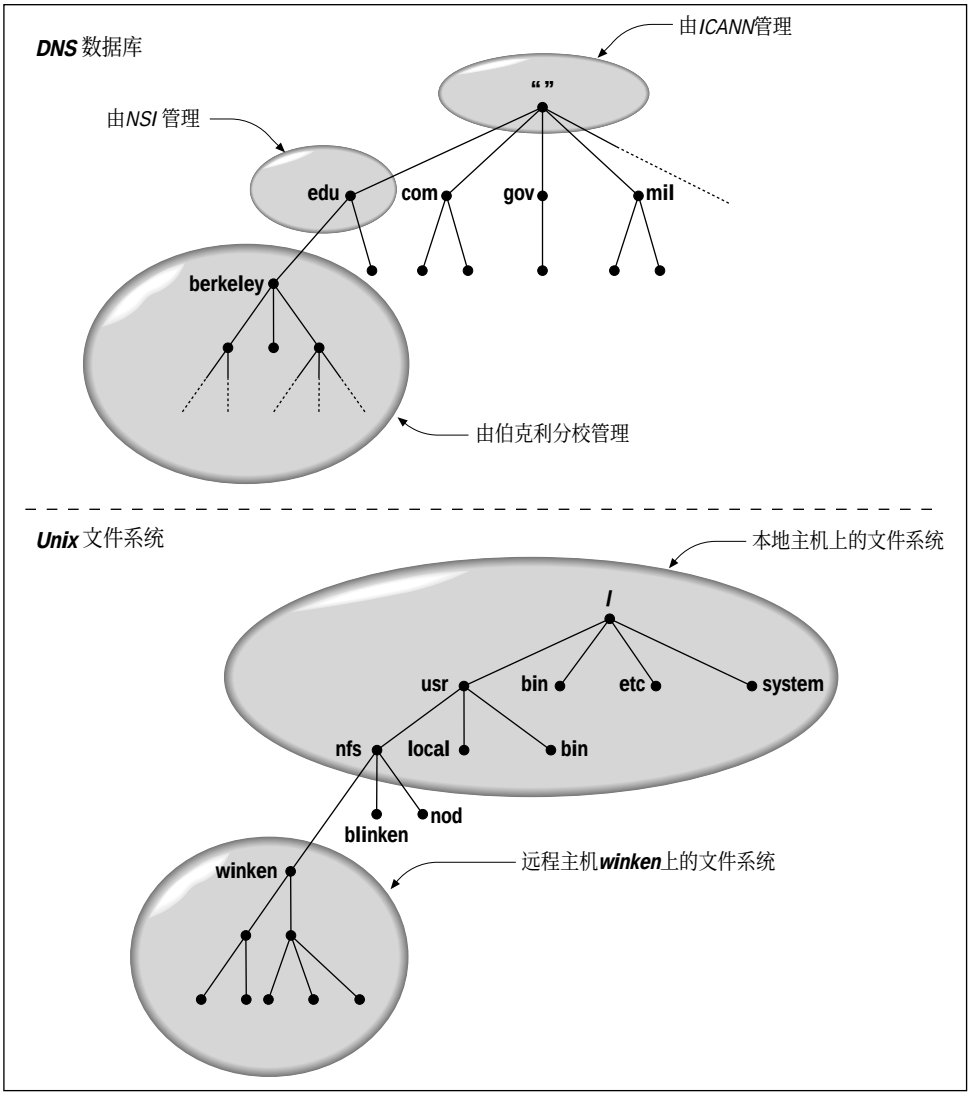


图 1-3　子域和文件系统的远程管理

edu 结尾的域名。另一方面，区 *edu* 只包括那些以 *edu* 结尾但又不在像 *berkeley.edu* 这样已授权出去的区里的域名。*berkeley.edu* 还可再分成一些子域，如 *cs.berkeley.edu*，而如果 *berkeley.edu* 的管理员将管理权授予其他组织，这些子域也能成为独立的区。如果 *cs.berkeley.edu* 是一个独立的区，那么区 *berkeley.edu* 中将不包括以 *cs.berkeley.edu* 结尾的域名（见图 1-4）。

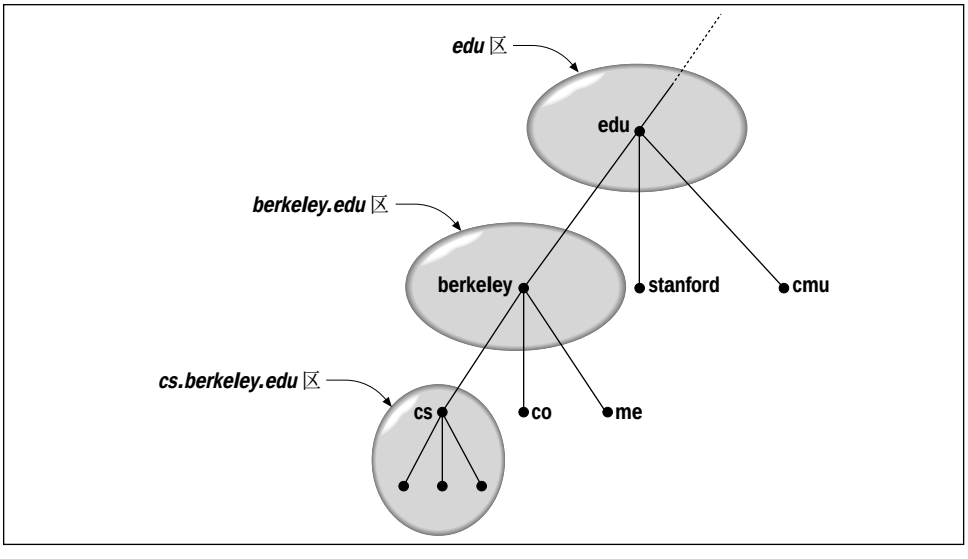


图 1-4　edu、berkeley.edu 和 cs.berkeley.edu 区

域名是作为DNS数据库的索引来使用的。可以把域名和DNS中的数据对应起来。在文件系统中，目录包含文件和子目录。同样地，域也同时含有主机和子域。域包含域名在该域中的主机和子域。

网上的任何一台主机都有域名，它指向关于该主机的信息（见图 1-5）。这些信息可以包括IP地址、邮件路由信息等。主机还可以拥有一个或多个域别名（domain name aliase），它们都是一些从别名指向正式或规范域名的指针。在图中，“ mailhub.nv... ”就是 “ rincon.ba.ca... ” 的别名。

所有这些复杂的结构是为了什么呢？是为了解决*HOSTS.TXT*文件所存在的问题。例如，域名采用层次结构消除了名字冲突的缺陷。每个域有惟一的域名，因而管理这个域名的组织可以自由地为该域中的主机和子域命名。无论他们为主机或子域选择

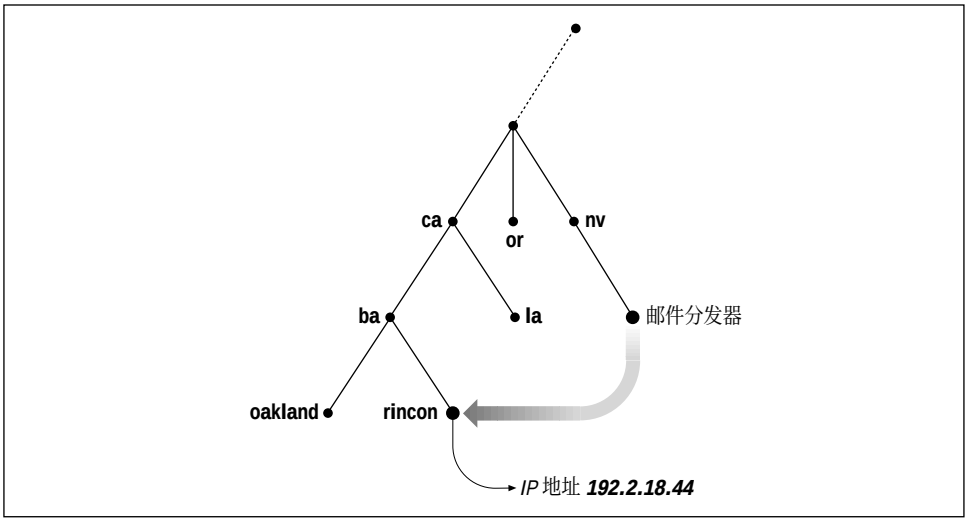


图 1-5　DNS 中指向规范名的别名

什么样的名字，都不会和其他组织的域名冲突，因为这个名字会以该组织惟一的域名为后缀。比如管理 *hic.com* 的组织能命名一台主机为 *puella*（见图 1-6），因为它知道这个主机的域名会以 *hic.com* 结尾，从而成为一个惟一的域名。

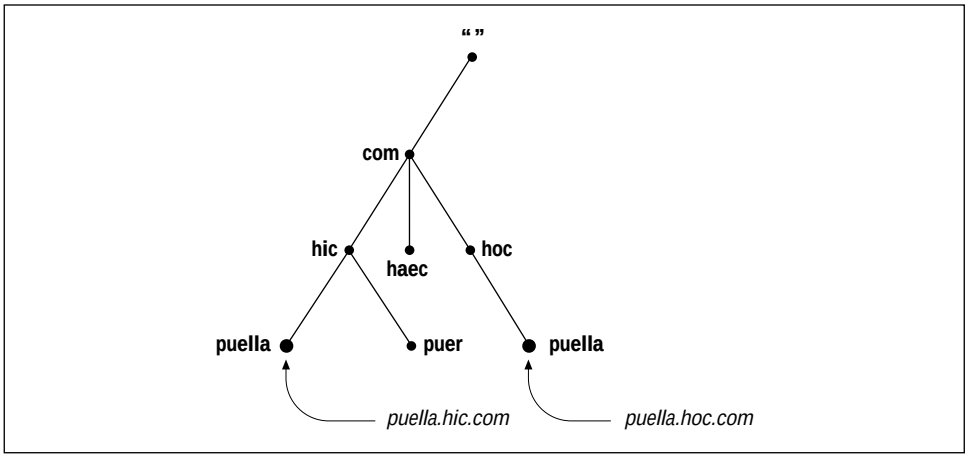


图 1-6　解决名字冲突问题

BIND 的历史

DNS 的第一个实现是 JEEVES，是由 Paul Mochapetris 亲自开发的。接下来就是 BIND，Berkeley Internet Name Domain 的缩写，它由 Kevin Dunlap 为伯克利的 4.3 BSD Unix 操作系统编写。BIND 现在由 Internet 软件协会维护（注 3）。

BIND 是我们将在本书中集中讨论的一种实现，也是目前最为常见的 DNS 实现。BIND 是迄今为止最流行的 DNS 系统。它已经被移植到大多数 Unix 变种上，并且被作为许多供应商的Unix标准配置封装在产品中。BIND甚至已经被移植到微软的 Windows NT 上。

我一定要使用 DNS 吗？

虽然DNS有许多优点，但在某些情况下，并不值得采用DNS。除DNS外，还有其他名字解析机制，某些还可能是你的操作系统的标准部件。有时管理区（zone）和名字服务器的开销要大于它带来的好处。然而有时你又别无选择，只能建立并管理名字服务器。下面就是一些帮助你做出决定的准则：

如果你已连接到 Internet...

你必须选用 DNS。DNS 可以看成是Internet 的通用语言：几乎所有 Internet 的网络服务都使用DNS。其中包括WWW、电子邮件、远程终端访问和文件传输。

不过，这并不意味着你必须自己动手来建立和管理自己的区。如果你只有很少量的主机，你可以找到一个已存在的区，使自己成为它的一部分（参见第三章），或者你可以让别人来帮你管理区。如果你是向Internet服务提供商（ISP）购买的 Internet 连接，可以询问他们是否也能帮你管理区。除此之外，还有专门的公司能帮助你解决这个问题，当然，这是要收费的。

如果你有大量主机，可能就需要自己的区了。而如果你又想直接控制你的区和名字服务器，就必须自己来管理它。接着往下读吧！

注 3： 要想了解 Internet 软件协会（Internet Software Consortium）及其关于 BIND 工作的更多信息，请访问 *http://www.isc.org/bind.html*。

如果你有自己的基于 TCP/IP 协议的互联网络 ...

你可能需要DNS。这里我们所说的互联网络并不是指采用TCP /IP协议的单个以太网（如果你特指以太网，请看下一节）；我们指的是相当复杂的“ 由多个网络组成的网络（network of networks）”。比如说，你有一大堆 AppleTalk 网以及一些 Apollo 令牌环网。

如果你的互联网结构基本上是同构的，而你的主机又不需要使用 DNS（比如，你有一个规模很大的 DECnet 或者 OSI 互联网络），没有 DNS 也行。但是如果你的主机上运行了多种系统，特别是其中有一些运行着不同版本的Unix，那么你就需要 DNS 了。它将简化主机信息的分发，消除你可能会面对的各种不同的主机表分发机制。

如果你有自己的局域网或区域网 ...

而这个网络又没有连接到更大的网络上，你可能就不需要使用 DNS 了。你可以试着考虑使用微软的 WINS（Windows Internet Naming Service，Windows Internet 命名服务）、主机表或者 Sun 的 NIS（Network Information Service，网络信息服务）产品。

但是，如果你需要分布式的管理，或是在保持网络数据的一致性上有麻烦，DNS 或许会适合你。而且，如果你的网络很快就要连接到其他网络上，比如你的公司互联网或是 Internet，那么现在就建立起你自己的区将会是明智之举。

第二章

DNS 是如何工作的?

本章内容：

- 域名空间
- Internet 上的域名空间
- 授权
- 名字服务器和区
- 解析器
- 解析
- 缓存

爱丽丝想：“ 如果没有图片和对话，
书还有什么用呢？ ”

DNS 主要是关于主机信息的数据库。毫无疑问，你可以从中得到许多信息：有趣的点分名字、联网的名字服务器、影子（shadowy）名字空间。但要记住，DNS 服务最终要提供的是所有互联网络主机的信息。

前一章，我们已经讲了 DNS 的一些重要方面，包括客户－服务器体系结构和 DNS 数据库的结构。但我们没有涉及过多的细节，也没有解释 DNS 操作的具体内容。

在本章，我们将解释和说明DNS 的工作机制。此外，还要向你介绍一些在阅读本书其他部分时所需要了解的术语（这些术语也将有助于你更好地与你的区管理员交流）。

首先，还是让我们来进一步看看上一章所介绍的一些概念，做点补充。

域名空间

DNS的分布式数据库是以域名为索引的。每个域名实际上就是一棵很大的逆向树中的路径，这棵逆向树称为域名空间（domain name space）。这棵树的层次结构如图

2-1 所示，和 Unix 文件系统的结构非常相似。在顶端，树有惟一的一个根（注 1）。在 Unix 文件系统中，这被称为根目录（root directory），用斜杠“ / ”表示，而在 DNS 中，就叫做“ 根 ”（root）。同文件系统一样，DNS 的树在每个分叉点可以有任意多个分支，这些分叉点称为“ 节点 ”（node）。树的深度不得超过 127 层（实际应用中，也不太可能超过）。

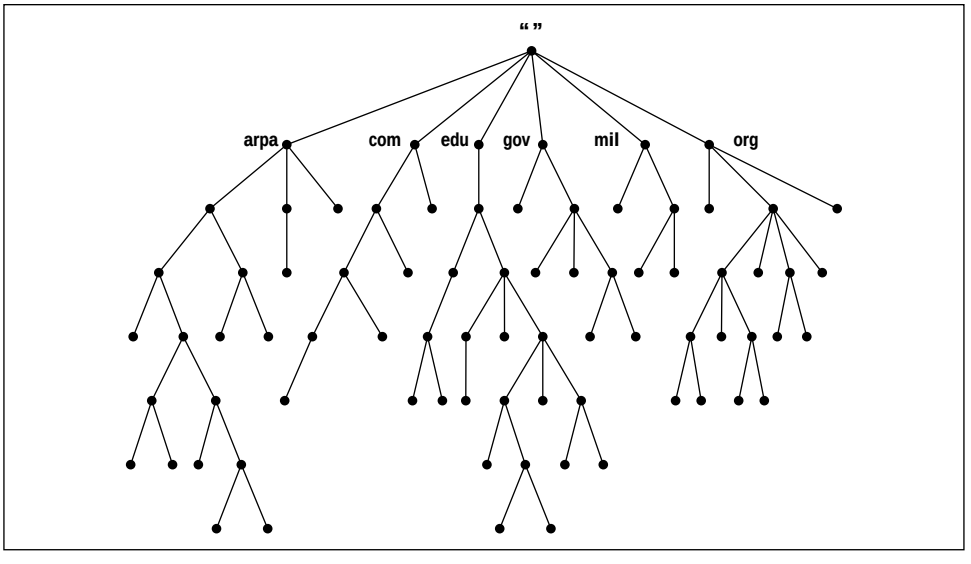


图 2-1　DNS 名字空间的结构

域名

树中每个节点都有一个可以长达 63 个字符的文本标号（text label）（不含“ . ”）。空标号（长度为 0）是为根保留的。树中任何一个节点的完整域名（domain name）都是从该节点到根的路径上所有节点标号的顺序连接。域名总是从节点向根（向上）的方向读，并用“ . ”来分隔路径上所出现的各个名字（文本标号）。

如果根节点的标号出现在一个节点的域名中，那么这个名字看上去就像以“ . ”结束，比如 *www.oreilly.com.*，但它实际上是以分隔符“ . ”和根的空标号结束的。当根的空标号单独出现时，简写成一个单独的点“ . ”。因而，有些软件将域名末端出现的“ . ”解释为这个域名是绝对（absolute）域名。绝对域名是相对于根而言的，它惟一

注 1：　很明显，这里的树是计算机科学家的树，而非植物学家的树。

确定一个节点在层次结构中的位置。绝对域名也用以指代全限定域名（fully qualified domain name），通常简写成 FQDN。不以“.”结束的域名有时被解释为是相对于某个非根域的，就像不以“\”开始的目录名通常被解释为是相对于当前目录的一样。

DNS 要求兄弟节点（就是具有同一父节点的孩子）要有不同的标号，这可以保证一个域名和树中的一个节点是一一对应的。这个限定其实并不会带来什么不便，因为标号只需要在同一节点的所有孩子节点中保持惟一，而不是在这棵树的所有节点中保持惟一。Unix 文件系统中也有同样的规定：两个兄弟目录或同一目录中的两个文件不能有相同的名字。如同在一个名字空间中不能有两个 *hobbes.pa.ca.us* 节点一样，Unix 文件系统中也不能有两个 */usr/sbin* 目录（见图 2-2）。相反地，当名字空间可以同时具有 *hobbes.pa.ca.us* 节点和 *hobbes.lg.ca.us* 节点时，NT 文件系统也可以既有 */bin* 目录也有 */usr/bin* 目录。

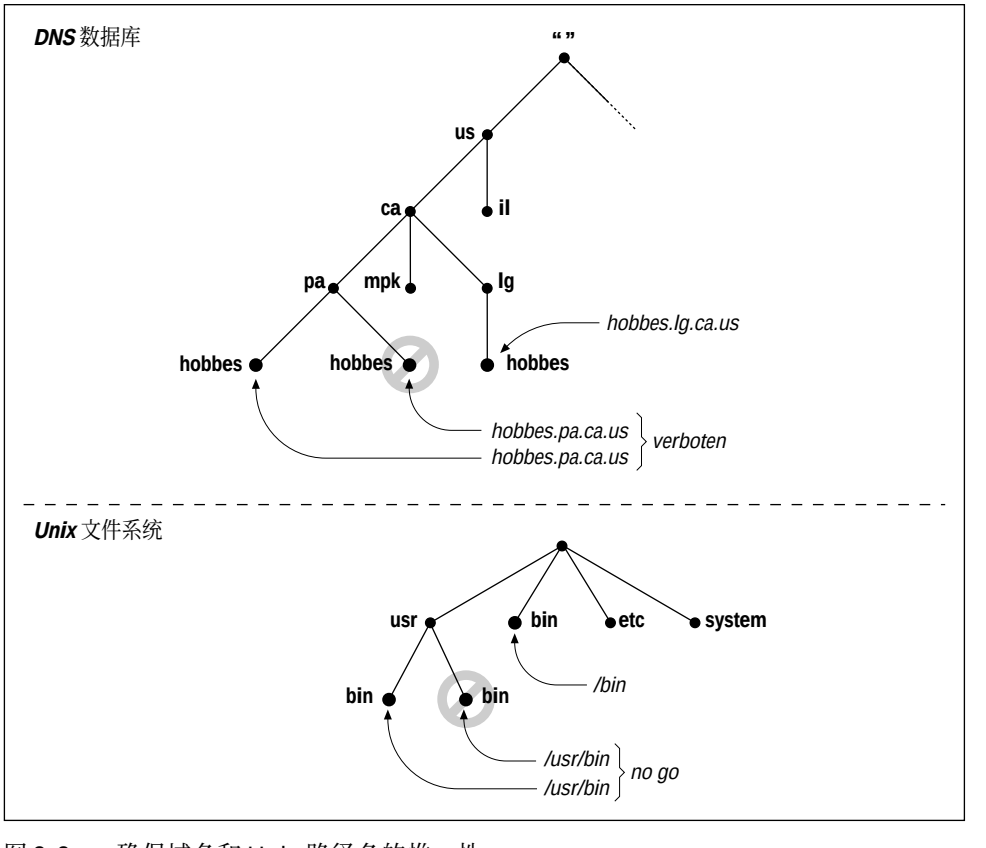


图 2-2　确保域名和 Unix 路径名的惟一性

域

一个域（domain）就是域名空间中的一棵子树。域的名字也就是这棵子树的顶端节点的域名。在图 2-3 中，*purdue.edu* 域的顶端节点就是 *purdue.edu*。

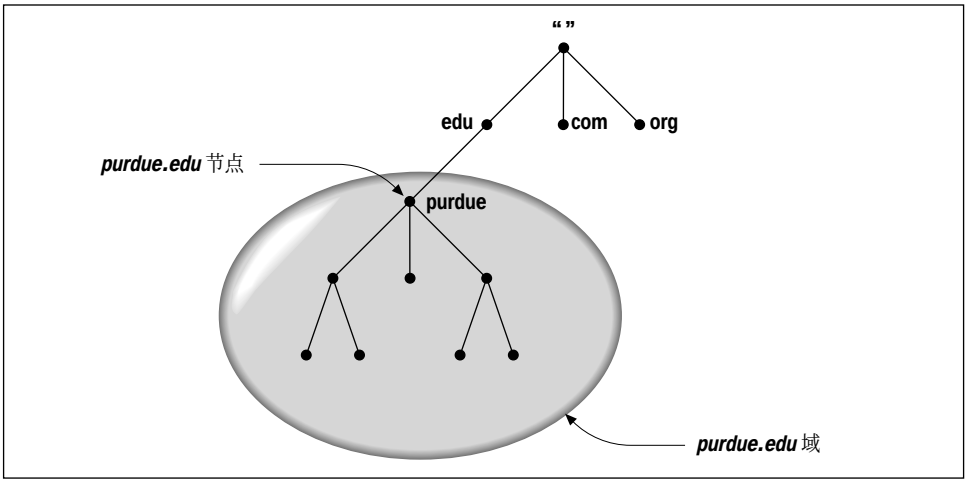


图 2-3　purdue.edu 域

同样地，在文件系统中，*/usr* 目录的顶端一定有叫做 */usr* 的节点，如图 2-4 所示。

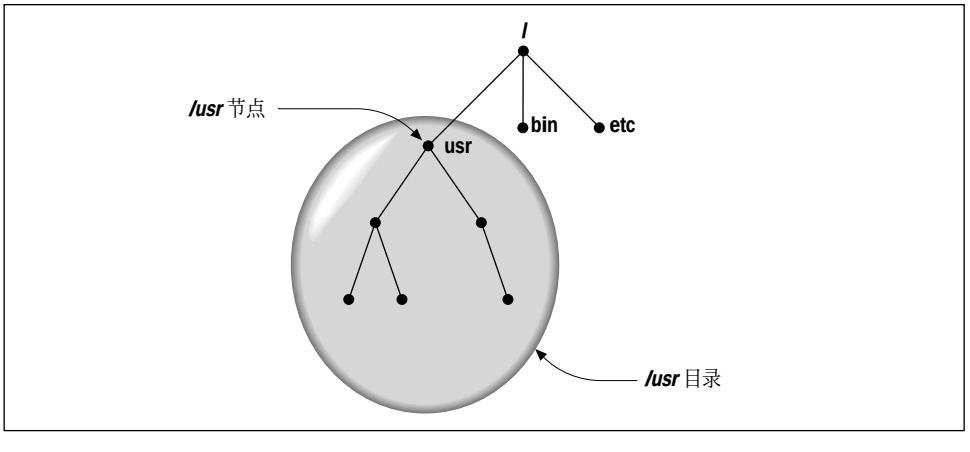


图 2-4　/usr 目录

子树（即域）中的任何域名都被看做是域的一部分。一个域名可以出现在多个子树

中，所以一个域名也能出现在多个域 中。如图 2-5 所示，*pa.ca.us* 是 *ca.us* 域的一部分，同时也是 *us* 域的一部分。

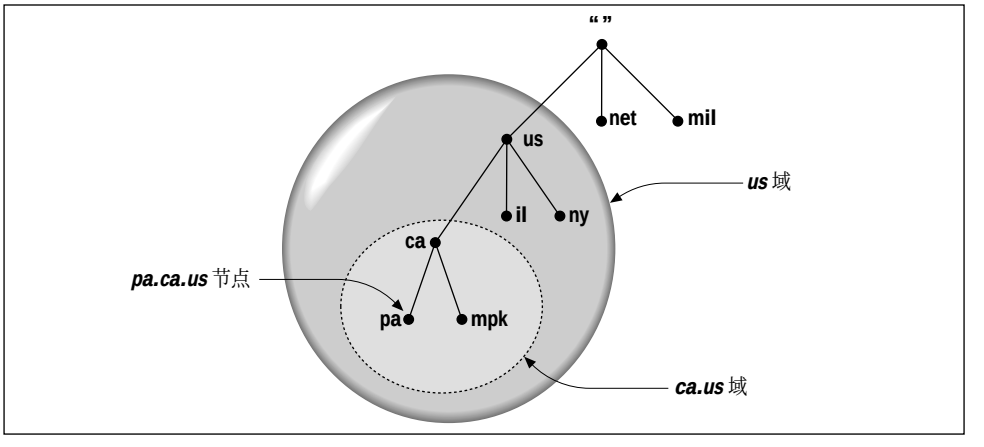


图 2-5　一个节点处于多个域中

简而言之，域就是域名空间中的一棵子树。不过，如果域只是简单地由一些域名和子域组成，那么主机又在哪里呢？域难道不是一组主机吗？

主机在这里，它们用域名来表示。记住，域名只不过是 DNS 数据库中的索引，“ 主机 ”就是那些分别指向各个主机信息的域名。一个域包含所有那些域名在其中的主机。这些主机是逻辑相关的，通常是由于地理或组织上的关系而联系在一起的，而不依赖于网络、地址或硬件类型。也许有十台不同的主机，它们分属不同的网络，甚至处于不同的国家，但却在同一个域中（注 2）。

位于树中叶子位置的域名通常表示单个主机，也可以指向网络地址、硬件信息和邮件路由信息。树内部的域名既可以命名一台主机，也可以指向有关该域的信息。内部节点域名并不一定非此即彼，它们可以同时表示与之相联系的域和网络上的一台

注 2：　注意：不要把 DNS 中的域同 Sun 的 NIS 中的域搞混了。虽然 NIS 域也指向一组主机，而且两种域也都有相似的结构化名字，但两者的概念是截然不同的。NIS 用的是层次化名字，但这种层次也就仅限于在同一NIS域中的主机共享关于主机和用户的某些数据，它们不能通过浏览 NIS 名字空间来查找其他 NIS 域中的数据。NT 域提供了账号管理和安全服务，不过它同 DNS 域也没有什么关系。

特定主机。比如，*hp.com* 既是惠普域的名字，也是运行惠普主Web服务器的主机域名。

使用域名时所获取的信息类型取决于你使用时的上下文。给*hp.com*中的某个人发邮件返回的是邮件路由信息，而远程登录（telnet）这个域名时，则返回的是主机信息（例如在图 2-6 中，*hp.com* 的 IP 地址）。

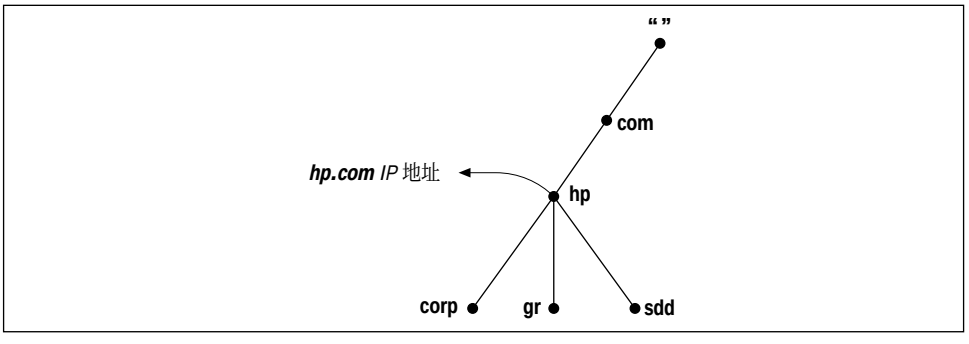


图 2-6　一个同时具有主机和结构化数据的内部节点

一个域可以有好几个它自己的子树，称为子域（注 3）。

要想判断一个域是否是另一个域的子域，很简单，只用比较它们的域名即可。子域的域名是以其父域的域名来结尾的。例如，域 *la.tyrell.com* 一定是 *tyrell.com* 的子域，因为 *la.tyrell.com* 是以 *tyrell.com* 结尾的。同样地，它也是 *com* 的子域。

除了用相对关系来表示域之外，比如说一个域是另一个域的子域，还常常用层次（level）来表示域。在邮件列表和 Usenet 新闻组中，就到处可见顶级域（top-level domain）或二级域（second-level domain）。这些术语是用来表示域在域名空间中的位置的：

- 顶级域是根的孩子。
- 一级域（first-level）是根的孩子（也就是顶级域）。

注 3：　在DNS和BIND的文档中，域和子域这两个词常常可以互换。这里我们用“子域”这个词来表示一种相对概念：如果一个域的根在另一个域中，那么它就是那个域的子域。

- 二级域是一级域的孩子，依此类推。

资源记录

与域名相关的数据是存放在资源记录（resource record）中的，简称RR。资源记录按照网络或软件的类型被划分成不同的类（class）。目前，有internet（任何基于TCP/IP协议的网络）类，有基于Chaosnet协议的网络类，还有使用Hesiod软件的网络类。（Chaosnet是一种具有重大历史意义的老网络。）

其中internet类是最为常见的。（我想大概没有什么人还在使用Chaosnet类，而Hesiod类的使用也主要限于MIT内。）本书中，我们主要讲的是internet类。

即使在同一类（class）中，记录也有好几种类型（type），这是依据存储在域名空间中的数据种类来划分的。不同的类有不同的记录类型，而同一类型又可能为多个类所共用，比如，几乎每个类都定义了“地址”（address）这样一种类型。类中每个记录类型都定义了它自己的记录语法，任何属于这个类和这个类型的资源记录都必须遵从这个语法。（想了解所有internet资源记录类型及其语法的具体内容，请参阅附录一。

如果以上内容看上去太粗略，别担心——待会儿，我们将更加详细地讲述internet类的记录。在第四章中，我们将讲到一些常用的记录，附录一中则有更全面的信息。

Internet上的域名空间

到目前为止，我们谈到域名空间的理论结构以及其中存储的数据类型，甚至对你可能见到的名字类型都在示例（有些是虚构的）中做了讲解。但这对你理解日常在Internet上看到的域名并没有太大的帮助。

DNS本身对域名中的标号并没有什么强制性要求，也没有赋予任何一层标号以特殊的意义。你在管理自己的域名时，可以自主地选定它们的语义。你只要用从A到Z的字母来命名你的子域，就没有人能阻止你（即使他们强烈建议你不要这样）。

不过，现在的Internet域名空间自行加上了一些结构。特别是上层的域，它们的域名遵循特定的惯例（不是规则，实际上你也可以打破这些惯例，而确实也有人这样

做了）。这些惯例可以使域名看上去不那么混乱。要想解释一个域名，了解这些惯例是很有用的。

顶级域名

最开始的顶级域名按组织将 Internet 域名空间分成七个域：

com
: 商业组织，例如惠普（*hp.com*）、Sun 公司（*sun.com*），还有 IBM（*ibm.com*）。

edu
: 教育机构，例如加州大学伯克利分校（*berkeley.edu*）和普渡大学（*purdue.edu*）。

gov
: 政府部门，例如 NASA（*nasa.gov*）和美国国家科学基金会（*nsf.gov*）。

mil
: 军事部门，例如美国陆军（*army.mil*）和美国海军（*navy.mil*）。

net
: 通常是提供网络基础设施的组织，例如 NSFNET（*nsf.net*）和 UUNET（*uu.net*）。但是从 1996 年起，*net* 与 *com* 一样，向任何商业组织开放。

org
: 通常是非盈利性组织，例如 Electronic Frontier Foundation（*eff.org*）。不过，与 *net* 一样，在 1996 年对 *org* 的限制也取消了。

int
: 国际组织，例如 NATO（*nato.int*）。

还有一个顶级域叫做 *arpa*，它原来是在 ARPAnet 从主机表向 DNS 系统转变中使用的。本来，ARPAnet 上的所有主机都在 *arpa* 域下有主机名，所以很容易找到它们。后来，它们逐渐移到了其他按组织功能来划分的顶级域的各个子域当中。不过，在某些情况下，*arpa* 域仍然在使用，稍后我们会讲到。

你可能会注意到，在上述内容中我们似乎有点偏见，所举的例子几乎都是美国的组织。其实这也很好理解，要记得 Internet 是从 ARPAnet 发展起来的，而 ARPAnet 是

一个由美国资助的研究计划。没人能预见到ARPAnet会有今天这样的成就，也没人能想到Internet会跨越国界，无处不在。

现在，原来这些域被称为通用顶级域（generic top-level domain），简称gTLD。2001年初，我们有了更多这样的通用顶级域，诸如*name*、*biz*、*info*和*pro*，这都是为了适应Internet的迅速发展、满足更多对域名空间的需求而出现的。负责管理Internet域名空间系统的ICANN（Internet Corporation for Assigned Names and Numbers）同意增加这些新的gTLD，同时在2000年末增加了*aero*、*coop*和*museum*。要了解ICANN的工作和新的gTLD，请访问*http://www.icann.org/*。

为了适应迅速增长的Internet国际化的需要，Internet域名空间原来的管理者们做出了某种妥协。他们不再坚持所有顶级域名都要采用组织模式，现在也允许采用地理模式来进行建立。新的顶级域名是为各个国家而保留的（不一定现在就要建立起来）。这些域名都遵循现有的国际标准ISO 3166（注4）。ISO 3166中规定用两个字母的简写来代表世界上的每个国家。附录四中列出了目前所使用的顶级域表。

深入一步

在各个顶级域中，惯例以及对惯例的恪守程度是不一样的。ISO 3166标准中的一些顶级域名严格遵循美国原来制定的组织模式。例如，澳大利亚的域名为*au*，有诸如*edu.au*和*com.au*之类的子域。而ISO 3166标准中的另一些域遵循的是*uk*域的做法，它们按照组织形成子域，*co.uk*表示公司，*ac.uk*表示学术机构。不过在大多数情况下，即使是按照地理位置形成的顶级域也会再按组织进行划分。

然而*us*顶级域却不是这样的。它有五十个子域对应于美国的五十个州（注5）。这些子域的名字也是采用美国邮政服务标准，用两个字母缩写代表州的名字。在每个州的域名中，组织也仍然是按地理位置来划分的：大多数子域对应于城市的名称。在城市下面，其子域常常是与各个主机相对应的。

注4： 除英国外。按照ISO 3166以及Internet惯例，英国的顶级域名应该是*gb*（Great Britain）。但是，在英国几乎所有的组织使用的顶级域名都是*uk*（the United Kingdom）。

注5： 其实在*us*域下还有一些子域：一个是华盛顿特区，一个是关岛，等等。

理解域名

既然知道了大多数顶级域名的含义以及它们名字空间的结构，你一定会发现理解域名是件很容易的事情。现在就让我们来试着分析一些域名，练习一下：

lithium.cchem.berkeley.edu

这个域名我们前面已经提到过，*berkeley.edu* 是加州大学伯克利分校的域名。（即使并不知道这个，你也应该能推断出它是美国一所大学的域名，因为这个域名是在 *edu* 这个顶级域中的。）*cchem* 是 *berkeley.edu* 的化学学院子域。最后面的 *lithium* 是这个域中某台主机的名字，如果他们为每台主机都分配域名的话，这可能就是几百个主机名中的一个。

winnie.corp.hp.com

这个例子有点难，但也不是太难。*hp.com* 无疑是惠普公司的域名（实际上在前面我们也已经提到过了）。*corp* 子域无疑是指公司总部，而 *winnie* 则是某人为某台主机所起的名字。

fernwood.mpk.ca.us

这里就需要你运用你对 *us* 域的理解了。很显然，*ca.us* 是加州的域，但 *mpk* 是什么就颇费思量了。如果你不了解旧金山的地理知识，你绝对猜不到它是 Menlo Park 的域名。（这可不是爱迪生住的 Menlo Park，那是在新泽西州。）

daphne.ch.apollo.hp.com

我们在此引入这个例子，只是想让你知道并不是所有的域名都只能由四个标号组成。*apollo.hp.com* 是 *hp.com* 域的一个子域，代表以前的阿波罗计算机公司。（惠普收购阿波罗时，同时也获得了阿波罗的 Internet 域，*apollo.com* 后来就成了 *apollo.hp.com*。）*ch.apollo.hp.com* 指的是阿波罗公司在马萨诸塞州 Chelmsford 的地址。而 *daphne* 是一台在 Chelmsford 的主机。

授权

还记得DNS设计的一个主要目的就是为了分散管理吗？这是通过授权（delegation）来实现的。对域进行授权，很像在工作中委派任务。经理可以将一个很大的项目分成一些很小的任务，并将它们委派给不同的员工负责。

同样地，域管理组织可以将域分成若干子域。这些子域可以授权给其他组织。这就意味着这些被授权的组织要负责维护其子域中的数据。它可以自由地改动数据，甚至将该子域再分成更多的子域，再进行授权。父域只保留指向子域数据源的指针，这样一来它就可以将查询者指向那里。比如说，在斯坦福管理校园网的人就被授予了管理 *stanford.edu* 域的权利，如图 2-7 所示。

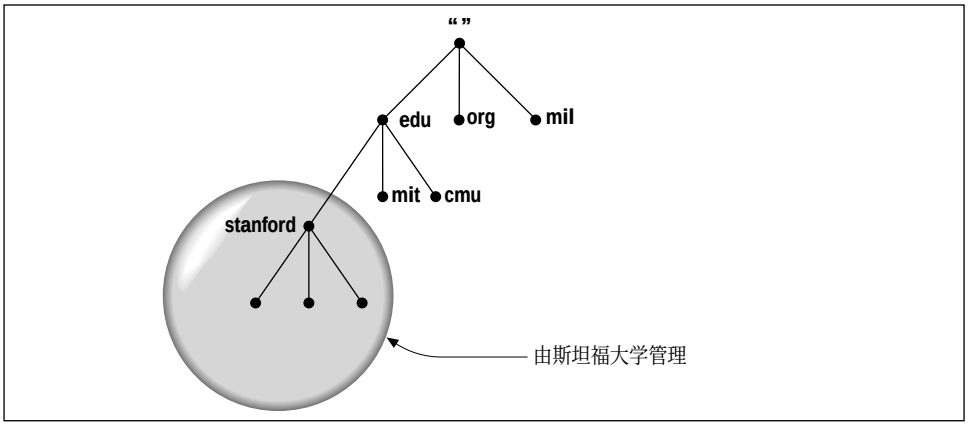


图 2-7　stanford.edu 被授权给斯坦福大学

不是所有的组织都会把它所有的域都授权出去，就像不是所有的经理都会把工作分派出去一样。一个域可能有好几个已经被授权出去的子域，而同时还有一些不属于这些子域的主机。比如说 Acme 公司，它在 Rockaway 有分公司，而总部在 Kalamazoo，它就应该有 *rockaway.acme.com* 子域和 *kalamazoo.acme.com* 子域。另外，它还有分散在全美国的销售点，这些地方的主机放在 *acme.com* 下，比放在任何一个子域下都要合适。

待会儿我们将介绍如何创建和授权子域。而现在最为重要的是要弄明白授权就是指将子域的管理责任授予另一个组织。

名字服务器和区

存储关于域名空间的信息的程序叫做名字服务器（name server）。名字服务器通常含有域名空间中某一部分的完整信息，这一部分我们称为区（zone），区的内容是从

文件或其他名字服务器中加载而来。这时我们就说这个名字服务器对这个区具有权威（authority）。一个名字服务器也可以同时对多个区具有权威。

区和域的区别是很重要的，也是很微妙的。所有的顶级域名，以及许多二级域名和更低级别的域名，比如*berkeley.edu* 和 *hp.com*，通过授权被分成了更小也更好管理的单元。这些单元被称为区（zone）。*edu* 域，如图 2-8 所示，被划分成了许多区，包括 *berkeley.edu* 区、*purdue.edu* 区和 *nwu.edu* 区。在域的顶部，还有一个 *edu* 区。管理*edu*的人将该域分解是十分自然的事，否则他们将不得不亲自管理*berkeley.edu* 子域，而将 *berkeley.edu* 授权给伯克利自己管理也会更有意义一些。那管理 *edu* 的人又做些什么呢？他们要管理 *edu* 区，这个区主要包含 *edu* 的子域的授权信息。

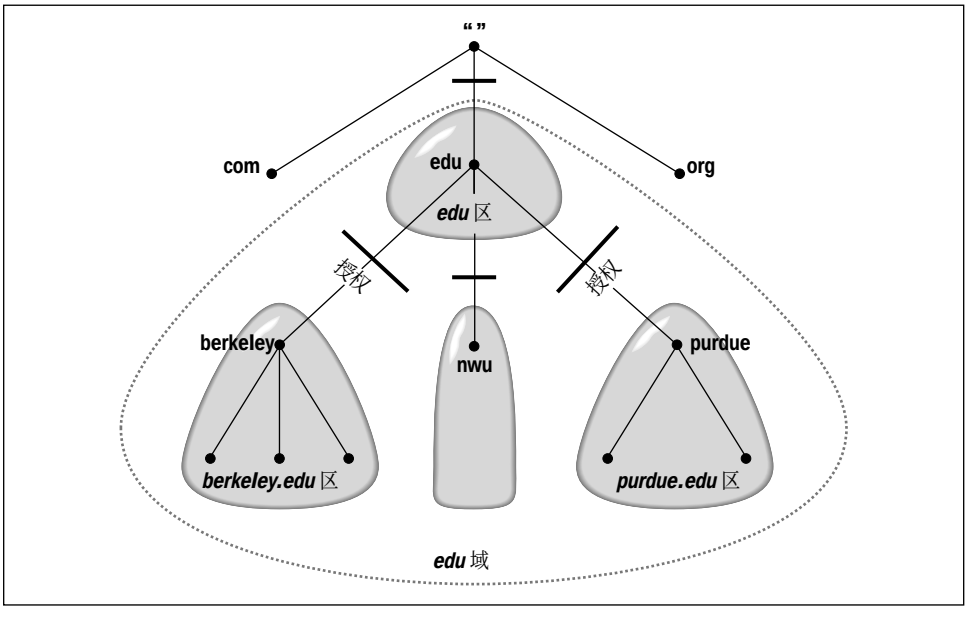


图 2-8　edu 域被分成许多区

berkeley.edu 子域同样也通过授权被分成了多个区，如图 2-9所示，有一些称为 *cc*、*cs*、*ce*、*me* 等等的授权子域。每个子域又被授权给一组名字服务器，其中有些名字服务器还同时具有 *berkeley.edu* 的权威。不过，即使各个区被授权给同一名字服务器，它们之间仍然是相互独立的，各自可以有完全不同的一组权威的名字服务器。

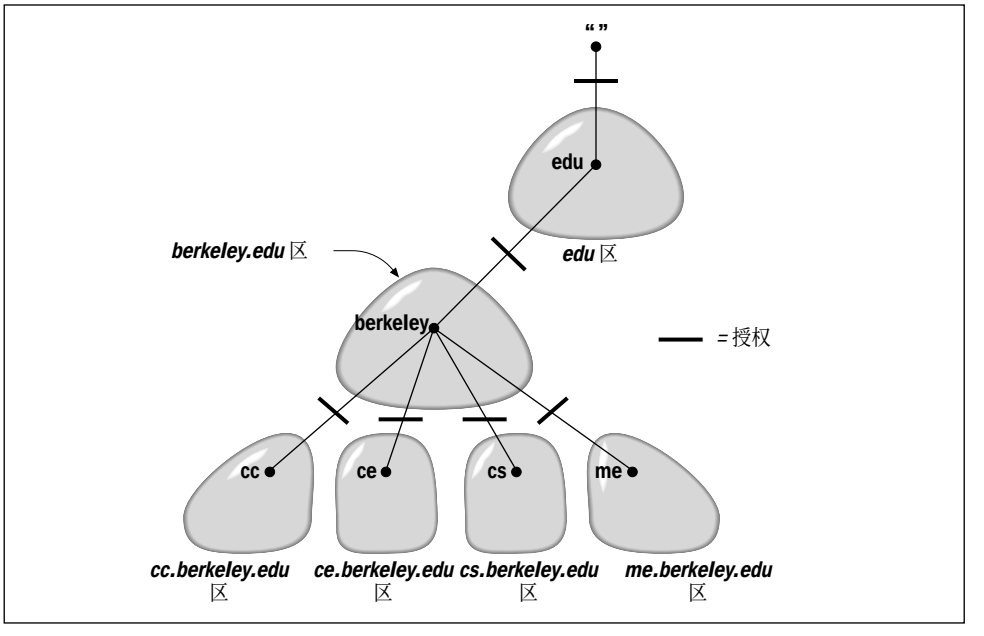


图 2-9 berkeley.edu 域被分成多个区

一个区和一个域可以共享同一个域名，却含有不同的节点。特别地，区不含有任何在已经被授权出去的子域中的节点。例如，顶级域名 *ca*（代表加拿大）有子域 *ab.ca*、*on.ca* 和 *qc.ca*，分别代表 Alberta、Ontario 和 Quebec。这些子域的权威分别被授予给各个省的名字服务器。域 *ca* 不仅包括 *ca* 中所有的数据，还包括 *ab.ca*、*on.ca* 和 *qc.ca* 中的数据。而区 *ca* 只包括 *ca* 中的数据（见图 2-10），这些数据可能主要是指向授权子域的指针，而 *ab.ca*、*on.ca* 和 *qc.ca* 是独立于 *ca* 的区。

如果域中某个子域没有被授权，那么区就包含这个子域中的域名和数据。子域 *ca* 的子域 *bc.ca* 和 *sk.ca*（代表 British Columbia 和 Saskatchewan）就没有被授权出去。（可能是 British Columbia 和 Saskatchewan 的当局还没有准备好管理自己的区，而负责顶级区 *ca* 的组织又想保持名字空间的一致性，就一次为加拿大所有的省都分配了域名。）在这种情况下，区 *ca* 就有一条不规则的底边，包括 *bc.ca* 和 *sk.ca* 而不包括 *ca* 的其他子域，如图 2-11 所示。

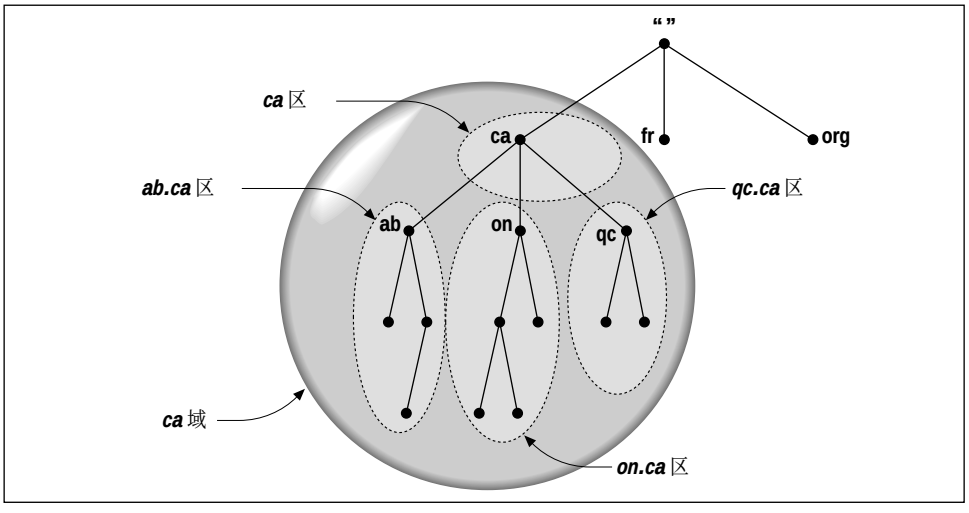


图 2-10　ca 域 ...

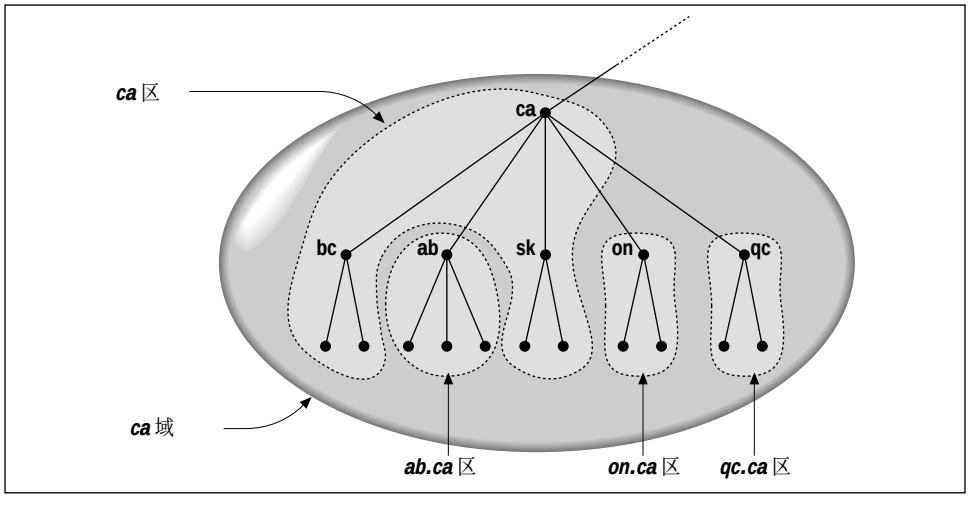


图 2-11　... 和 ca 区的对比

现在就很清楚为什么名字服务器要加载区而非域了：域可能会含有超过名字服务器所需的信息（注 6）。域可能含有已经授权给其他名字服务器的数据，而区是以授权来划分界限的，它决不会含有已被授权出去了的数据。

注 6：　想像一下，如果根名字服务器加载根域而不是根区：它将加载整个的域名空间！

如果你才刚刚开始，你的域可能还没有子域。此时还没有授权，所以这时域和区含有相同的数据。

授权子域

即使你现在还不需要把你的一部分域授权给其他组织，多了解一点如何对子域授权也是很有帮助的。简单地说，授权就是将你的一部分域的责任分配给另一个组织。实际操作就是将你子域的权威分配给其他一些名字服务器。(注意，这里我们说的是多个名字服务器，而不是一个。)

这时你的数据将不再包含已授权子域的数据，而是包括一些指针，这些指针指向相应子域的权威名字服务器。现在如果有一个请求，向你的名字服务器询问该子域中的数据，它就可以返回相应的名字服务器列表。

名字服务器的类型

DNS 规范定义了两种类型的名字服务器：*primary master*（主名字服务器）和 *secondary master*（辅名字服务器）。区的 primary master 名字服务器从位于本机的文件中读取区数据，而区的 secondary master 名字服务器则是从该区其他的权威名字服务器处读取区数据，后者称为前者的 *master*（主）服务器。颇为常见的是，这个 master 名字服务器就是这个区的 primary master，不过也不一定非要这样：一个 secondary master 也能从其他 secondary master 那里加载区数据。当一个 secondary master 启动时，它自动与其 master 服务器通信，如果需要，就从中获取区数据，这被称为“区传送”(zone transfer)。虽然许多人（和许多软件，包括微软的 DNS Manager）仍然使用 secondary 这个词，但现在人们更喜欢用 slave（辅名字服务器）来表示 secondary master 名字服务器（译注 1）。

其实一个区的主和辅名字服务器都是该区的权威。虽然slave这个词听起来有点儿贬低的意思，但实际上它并不是一个二等名字服务器。DNS提供这样两种类型的名字服务器是为了让管理变得简单些。一旦创建了区数据并建立起了主名字服务器，要

译注 1： 本书中 primary master、primary 和 master 的意义相同，文中以后统一为主名字服务器，secondary master、secondary 和 slave 的意义相同，文中以后统一为辅名字服务器。

创建新的名字服务器时，你就不需要傻乎乎地将数据从这台主机拷贝到那台主机了。你只需建立辅名字服务器，它能从该区的主名字服务器获取数据。一旦这些辅名字服务器建立起来，当需要时，它们就会自动传送新数据。

为指定的区建立多个名字服务器的确是个很好的想法，这样辅名字服务器就显得很重要了。出于冗余考虑，你想使用多个名字服务器，多个服务器既能分散负载，还能确保区中每台主机就近都有名字服务器。使用辅名字服务器使得这些要求在管理上变得切实可行。

其实，把某个名字服务器称为主名字服务器或辅名字服务器有点儿不太确切。前面我们提到过，一个名字服务器可以是多个区的权威。因而它也可以是这个区的主名字服务器，同时又是另一个区的辅名字服务器。不过多数名字服务器对于它所加载的大多数区而言，要么是主名字服务器，要么是辅名字服务器。所以，如果我们称某个名字服务器为主名字服务器或辅名字服务器的话，意思是说，对于以它为权威的大多数区来说，它是主名字服务器或辅名字服务器。

区数据文件

主名字服务器从本机中加载数据的文件叫做区数据文件（zone data file）。我们也常常称之为数据文件（data file）或数据库文件（database file）。辅名字服务器也能从数据文件中加载区数据。辅名字服务器常常被配置成为：将主名字服务器传送过来的区数据备份到本机的数据文件当中。如果后来辅名字服务器被“ 杀死 ”，再重启动时，它会首先读备份的数据文件，然后检查它的区数据是否是最新的。这样一来，既避免了传送还未更改的区数据，又为在主名字服务器关闭时提供了数据源。

数据文件包含描述区的资源记录。这些资源记录描述了区中所有的主机，并标明对子域的授权情况。BIND 还允许数据文件中含有一些特殊的指令，用来包括区数据文件中其他数据文件的内容，这非常像 C 语言中使用的 *#include* 语句。

解析器

解析器是访问名字服务器的客户端程序。主机上运行的应用程序如果需要从域名空间中获得信息，就要使用解析器。解析器处理以下任务：

- 向名字服务器提出查询
- 解释响应信息（可能是资源记录，也可能是出错信息）
- 向提出请求的程序返回信息

在BIND中，解析器就是链接到诸如Telnet和FTP这样程序的一组库例程。它甚至不是一个独立的进程。它有收集查询、向名字服务器发送查询并等待应答、如果没有应答就再次发送查询的功能，不过也就只能做这么多了。大部分寻找对查询请求的应答的工作是在服务器一方完成的。DNS规范把这种解析器称为存根解析器（stub resolver）。

某些DNS的实现中有功能更强的解析器，它们能完成一些更为复杂的事情，比如说为已经检索到的信息建立一个缓存（注7）。不过它们远不如BIND中实现的存根解析器的使用那么普遍。

解析

名字服务器很善于从域名空间中检索数据。由于大多数解析器的功能有限，名字服务器也不得不如此。它们不仅能给出自己享有权威的区的数据，还能在域名空间中搜索，找到它们并不享有权威的区的信息。这个过程称为“名字解析”（name resolution），或简单地说成“解析”（resolution）。

名字空间采用逆向树结构，所以只需要一条信息就能找到树中任何一个点，这条消息就是根名字服务器的域名和地址。名字服务器能向根名字服务器发出查询域名空间中任意名字的要求，然后根名字服务器再启动相关的子域名字服务器。

根名字服务器

根名字服务器知道所有顶级区的权威名字服务器在哪儿。（实际上，有些根名字服务器是普通顶级区的权威。）当收到关于某个域名的查询后，根名字服务器能提供该域名所在顶级区的那些权威名字服务器的名字和地址。然后，顶级名字服务器能提供

注7： 例如Rob Austein为TOPS-20设计的CHIVES解析器就能缓存。

该域名所在二级区的权威名字服务器的列表。每个被查询的名字服务器给查询者一个信息，或者是如何进一步找到答案的查询信息，或者这个信息本身就是查询者要查找的答案。

不言而喻，根名字服务器对解析来说是非常重要的。正是因为如此，DNS 提供了许多机制（如缓存，等会儿我们将要讨论）来帮助根名字服务器减轻负载。不过在缺乏其他信息的情况下，解析总是不得不从根名字服务器开始。这就使得根名字服务器对于 DNS 操作而言至关重要；如果 Internet 上所有根名字服务器都长时间不可到达的话，Internet 上的所有解析都会失效。为了防范于未然，Internet 有十三个根名字服务器（到本书写作时），分布在网络的不同部分中。例如，一个在商业 Internet 主干网 PSINet 上；一个在 NASA Science Internet 上；两个在欧洲；还有一个在日本。

这些根服务器是许多查询的汇聚点，所以非常繁忙；即使有十三个之多，每个根名字服务器的流量还是很高。最近一项在根名字服务器管理员中进行的非正式调查显示：某些根服务器每秒收到数千个查询。

尽管根名字服务器上的负载很重，Internet 上的解析工作还是进行得相当好。图 2-12 中表明了一个真实域中真实主机的地址解析过程，包括如何遍历域名空间树的过程。

本地名字服务器向根名字服务器查询 *girigiri.gbrmpa.gov.au* 的地址，根服务器告知它去联系 *au* 名字服务器。本地名字服务器问 *au* 名字服务器同样的问题，被告知 *gov.au* 名字服务器的地址列表。本地名字服务器从列表中选择一个 *gov.au* 名字服务器并向其继续询问同样的问题，*gov.au* 名字服务器就告诉本地名字服务器 *gbrmpa.gov.au* 名字服务器的地址。最终，本地名字服务器向 *gbrmpa.gov.au* 名字服务器询问并获得答案。

递归

你可能已经注意到，在上面这个例子中，各个名字服务器的工作量是有很大差别的。其中四个对所收到的查询只是很简单地从它们已有的答案中选出一个最适合的——主要是指出一些其他的名字服务器。它们不用为找到所需数据而亲自查询。但是，解析器直接查询的名字服务器（本地名字服务器）却要不断地依照指示（referral）进行查询，直到得到结果。

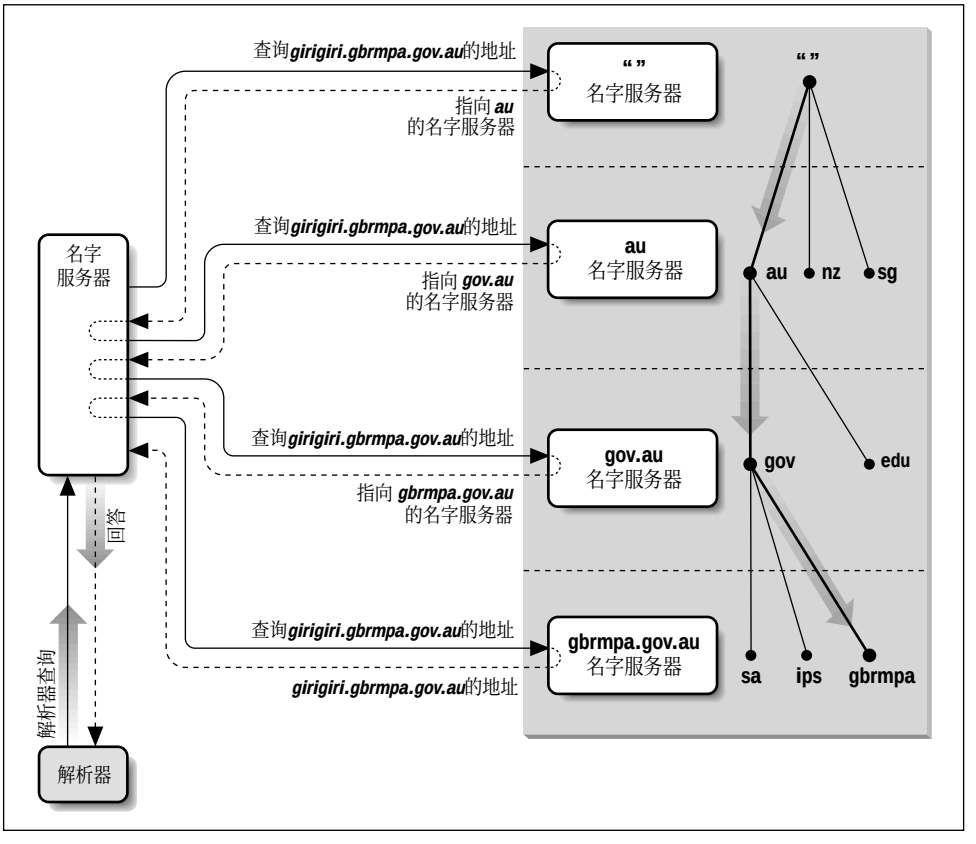


图 2-12 Internet 上域名 girigiri.gbrmpa.gov.au 的解析

为什么本地名字服务器不只是简单地把另一个名字服务器的地址告诉解析器呢？这是由于存根解析器没有跟随指示的智能。而名字服务器又是怎么知道它不能仅仅只回答一个指示呢？这是因为解析器发送的是一个递归（recursive）查询。

有两种查询方式：递归方式（recursive）和反复方式（iterative），后者也称为非递归方式（nonrecursive）。递归查询将大部分的解析负担置于一个名字服务器上。递归或递归解析（recursive resolution）指的是名字服务器在收到递归查询时所使用的解析过程。正如编程中的递归算法，名字服务器只重复一个简单的过程（向远程名字服务器提出查询，再遵照指示）直到收到结果。在下一节，我们将讲述反复（iteration）或反复解析（iterative resolution），它指的是名字服务器在收到反复查询时所使用的解析过程。

在递归方式中，解析器向（本地）名字服务器发送一个关于某个域名信息的递归查询。被查询的名字服务器必须返回所请求的数据或者是出错说明，出错可能包括所请求的类型数据不存在或所给域名不存在（注 8）。（本地）名字服务器不能只将查询者指向另一个名字服务器，因为该查询是递归的。

如果被查询的名字服务器不是所请求的数据的权威，它将不得不向其他名字服务器发出查询以获得答案。它可以向其他名字服务器发送递归查询，从而要求它们找到答案并返回（然后它自己再返回给查询者）。或者，它也可以发送反复查询，很可能又被指示到一些更靠近“所要寻找的域名的名字服务器。目前 DNS 的实现中，采用的是后者，将不断地依照指示进行查询，直到找到结果（注 9）。

当名字服务器收到一个它无法回答的递归查询时，它将向“最有可能知道的”名字服务器进行查询。最有可能知道的名字服务器是指与所要寻找域名最接近的区的权威。例如，如果名字服务器收到一个要求查找域名 *girigiri.gbrmpa.gov.au* 的递归查询，它首先检查自己是否知道谁是 *girigiri.gbrmpa.gov.au* 的权威名字服务器。如果它知道，就将查询发往其中一个名字服务器。如果不知道，它就会检查自己是否知道 *gbrmpa.gov.au* 的名字服务器，然后是 *gov.au*，再就是 *au*。默认值是根区，检查到这儿一定会停止，因为每个名字服务器都知道根名字服务器的域名和地址。

使用最有可能知道的名字服务器可确保解析过程尽可能短。当 *berkeley.edu* 的名字服务器收到查找 *waxwing.ce.berkeley.edu* 的地址的递归查询时，它不会去查询根名字服务器；它只需沿着授权信息，直接找到 *ce.berkeley.edu* 的名字服务器。同样地，一个刚刚查找过 *ce.berkeley.edu* 中域名的名字服务器再查找其中的域名时，不必从根节点开始再解析一次 *ce.berkeley.edu*（或 *berkeley.edu*）；在后面“缓存”一节中我们将讲述这是如何实现的。

收到递归查询的名字服务器发送的查询总是与解析器发给它的一模一样，比如说，要查询 *waxwing.ce.berkeley.edu* 的地址。这个名字服务器绝不会只向其他名字服务

注 8： BIND 8 名字服务器可以被配置成拒绝递归查询；为何要这样以及如何做到，参见第十一章。

注 9： 有一个例外，服务器能被配置成将所有不能解析的查询都提交给一个指定的名字服务器，该指定的服务器被称为“转发器”（forwarder）。关于转发器的使用详情见第十章。

器查询*ce.berkeley.edu*或*berkeley.edu*，虽然这些信息也同样存储在名字空间中。这样的查询会造成一些问题：可能并没有*ce.berkeley.edu*名字服务器（也就是说，*ce.berkeley.edu*可能是*berkeley.edu*区的一部分）。同时，也有可能*edu*或*berkeley.edu*名字服务器知道*waxwing.ce.berkeley.edu*的地址。那么只查询*berkeley.edu*或*ce.berkeley.edu*将会错过这些信息。

反复

与递归解析相对应的反复解析就不要求被查询的（本地）名字服务器做那么多工作。在反复解析中，名字服务器只用将它已知的最合适的答案返回给查询者。它本身不需要再有任何其他查询。被查询的名字服务器在它的本地数据（包括它的缓存，待会儿我们会谈到）中寻找所需数据。如果没有找到答案，它就在本地数据中找出与所要查询的名字服务器最接近的名字服务器的名字和地址，并作为指示返回给查询者，帮助它把解析过程进行下去。注意，这个指示包括本地数据中列出的所有名字服务器，由查询者来选择下一个向谁提出查询。

在权威名字服务器间选择

有的读者也许会问收到递归查询的名字服务器如何在区的权威名字服务器间选择呢？比如，我们曾说过现在有十三个根名字服务器。名字服务器只是简单地向指示中列在第一个的名字服务器查询吗？或者是随机选择？

BIND名字服务器使用一种称为“往返时间”（roundtrip time）或RTT的度量（metric）对同一个区的权威名字服务器进行选择。往返时间是指远程名字服务器响应查询的时间长度。每次BIND名字服务器向远程名字服务器发送查询时，都启动一个内部计时器。当它收到响应时就停止计时，记录下该远程名字服务器过了多长时间才响应。当名字服务器要选择向哪个名字服务器发送查询时，它就选择具有最小RTT的名字服务器。

在BIND名字服务器查询某个名字服务器之前，先给它一个随机的RTT值，不过这个值比任何真实的RTT值都要小。这就保证在根据真实RTT值选择之前，BIND名字服务器会以随机的顺序查询某个区的所有名字服务器。

总的来说，这种简单而有效的算法使得 BIND 名字服务器能迅速“锁定”最近的名字服务器，而且又没有带外（out-of-band）性能测试机制的开销。

小结

总地来说，以上描述构成整个解析过程，如图 2-13 所示。

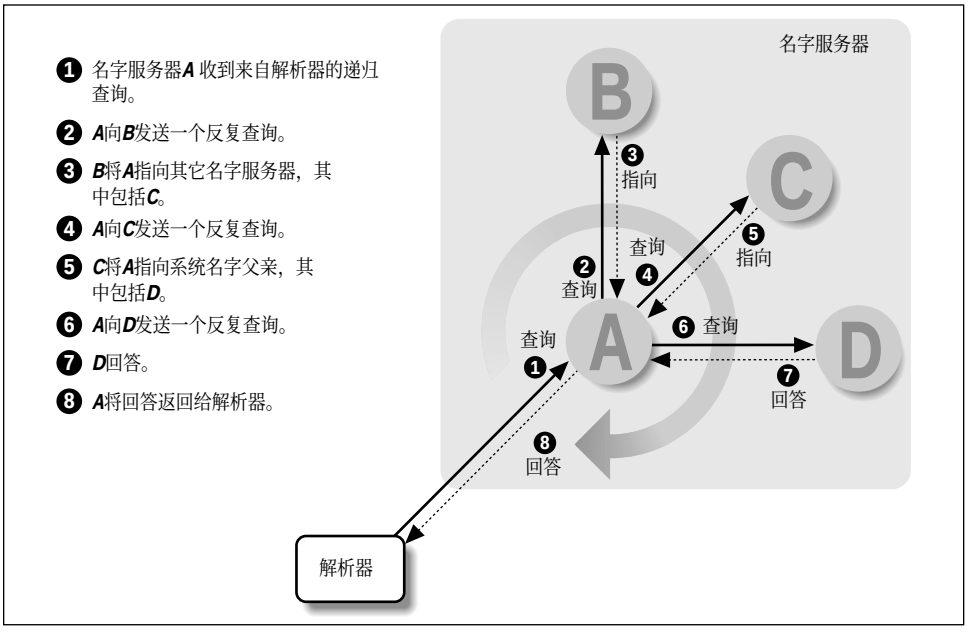


图 2-13　解析过程

解析器向本地名字服务器发送查询，而本地名字服务器为该解析器寻求答案，会向许多其他名字服务器发送反复查询。它所查询的每个名字服务器依次返回在名字空间中更进一层、更靠近所查域名的区的权威名字服务器。最后，本地名字服务器向权威名字服务器发生查询，后者返回答案。本地名字服务器总是使用它收到的每个响应（无论是指示，还是答案）来更新响应名字服务器的 RTT，而 RTT 值将有助于今后决定向哪个名字服务器发送解析域名的查询。

地址到名字的映射

谈到现在，解析过程还有一个主要功能没有讲，那就是地址是如何映射到域名的。地址到名字的映射被用来产生输出（例如，用在日志文件中），这可以方便人的阅读和解释。它还可以用于某些授权检查当中。比如，Unix主机把地址映射为名字用来比较*.rhosts*和*hosts.equiv*文件中的条目。使用主机表进行地址到名字的映射是非常简单的。只需要直接在主机表中顺序查找这个地址。查找结束后，将返回正式的主机名。不过在DNS中，完成地址到名字的映射就没有这么简单了。数据，包括地址，在域名空间中是按照名字来索引的。给定一个域名，查找地址相对而言要容易一些。而根据给定的地址映射到名字似乎要费力得多，因为它要查找树中与每个域名相关的数据。

实际上有一个更好的解决方法，既聪明又有效。既然给定了一个索引数据的域名后，你就能很容易地找到数据，那为什么不创建一部分以地址为标号的域名空间呢？在Internet域名空间中就特地划出一部分名字空间作为此用，这部分名字空间被称为*in-addr.arpa*域。

*in-addr.arpa*域中的节点都是用数字来作标号的，这些数字就是以点分字节（dotted-octet）来表示的IP地址。（用点分隔四个0到255之间的数字称为“点分字节”表示法，这是表示32位IP地址的常用方法。）比如说，*in-addr.arpa*最多有256个子域，分别和IP地址中的第一个字节的每个可能值相对应。这些子域又可以有256个自己的子域，分别和第二个字节每个可能的值相对应。最后，在第四层下面，与最后一个字节相连的资源记录给出了这个IP地址对应的主机或网络的全称域名。这个庞大的域：*in-addr.arpa*，如图2-14所示，足够容纳Internet上所有的IP地址。

记住，在读域名的时候，IP地址应该倒过来显示，因为名字是从叶子往根的方向读的。比如说，如果*winnie.corp.hp.com*的IP地址是15.16.192.152，那么它相应于*in-addr.arpa*子域的名字就是*152.192.16.15.in-addr.arpa*，该IP地址反向映射到域名*winnie.corp.hp.com*。

IP地址是以与名字空间相反的方向来表示的，IP地址的第一个字节在*in-addr.arpa*域的底部。这样一来，IP地址就能从上至下正确地读出了。

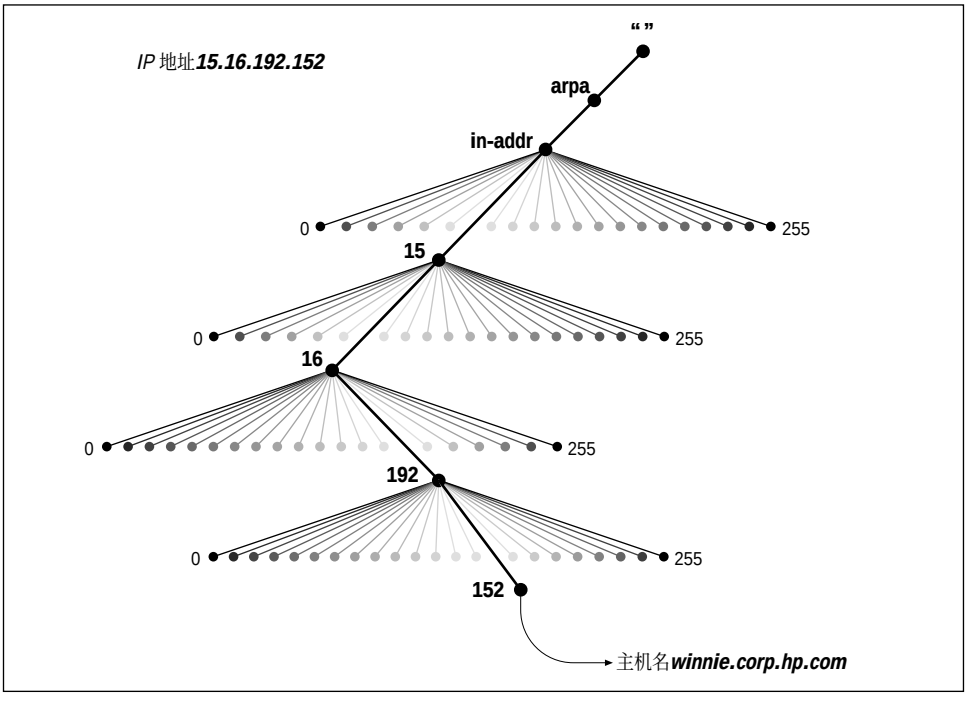


图 2-14　in-addr.arpa 域

如同域名一样，IP 地址也是采用层次结构的。网络号也同域名一样被分发，而且管理员可以将他的地址空间分成若干子网，再进一步把网络号授权下去。IP 地址同域名的区别在于：IP 地址从左向右越来越细化，而域名则刚好相反。图 2-15 说的就是这个意思。

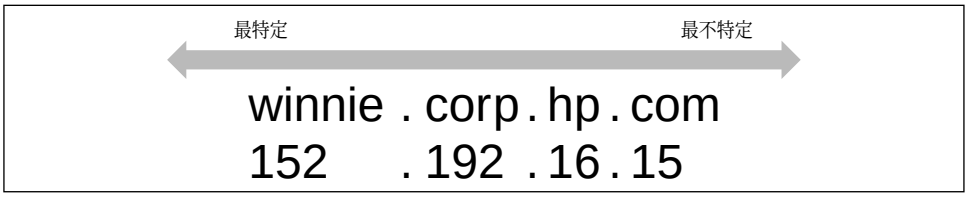


图 2-15　名字和地址的层次

使 IP 地址的第一个字节出现在树中的最高层，将会给管理员根据网络号来对 *in-addr.arpa* 区授权的能力。例如，*15.in-addr.arpa* 区包含的就是所有 IP 地址以 15 开头的主机的反向映射信息，可以把它授权给网络 15.0.0.0 的管理员。如果把 IP 地址

正向表示在 *in-addr.arpa* 区，那么 *15.in-addr.arpa* 表示的就是所有 IP 地址以 15 结尾的主机 —— 要为这个区授权显然不太现实。

逆向查询

很显然，*in-addr.arpa* 域只对由 IP 地址到域名的映射有用。要想在域名空间中搜索一个以任意部分数据（类似于地址的形式，但是除地址以外的某部分数据）为索引的域名也都需要类似于 *in-addr.arpa* 的专门的名字空间，或者只能是费力地逐个查找。

逐个进行查找在某种程度上也是可行的，这被称为“ 逆向查询 ”（inverse query）。逆向查询指的是根据给定的数据查找域名。这样的查询只由收到查询的名字服务器来处理。该服务器在本地数据库中搜索待查询的数据，如果可能，就返回对应的域名。如果找不到，就放弃。它不会试图查询其他名字服务器。

由于任意一个名字服务器都只了解整个域名空间的一部分，所以逆向查询不一定能返回结果。例如，当名字服务器收到一个它全然不知的 IP 地址的逆向查询时，就不能返回任何结果，但是，它也不知道这个 IP 地址是否存在，因为它只保存着 DNS 数据库的部分信息。另外，逆向查询的实现在DNS规范中是一个可选项；BIND 4.9.8 也有实现了逆向查询的代码，但是默认情况下是被注释掉的。BIND 8 和 BIND 9 根本没有包括这些代码，虽然它们还能够识别逆向查询并拼凑一个假的响应（注 10）。其实这也没什么，因为只有在一些很小的软件（比如，老版本的 *nslookup*）中才使用逆向查询。

缓存

对习惯使用主机表查询的人来说，名字解析的整个过程似乎太过麻烦。但在实际使用中，它还是很快的，一个很主要的特色就在于缓存（caching）。

处理递归查询的名字服务器可能需要发送好几个查询才能找到结果。不过在这个过程中，它也了解到域名空间的许多信息。每次它得到一些名字服务器列表的指示，

注 10： 关于这个功能的细节，请看第十二章中的“ 查询被拒绝 ”一节。

它就知道这些名字服务器是哪些区的权威，也知道这些服务器的地址。当解析过程结束时，它最终找到原来查询所请求的数据后，还可以把这些数据保存起来，以备后用。在 BIND 4.9 以及所有的 BIND 8 和 9 中，名字服务器甚至还进行否定缓存（negative caching）：当某个权威名字服务器返回结果，说所查询域名或数据不存在时，本地名字服务器也会暂时将该信息放入缓存。名字服务器将所有这些数据都放入缓存是为了提高以后查询的速度。下次解析器向名字服务器查询某个它所知的域名数据时，解析过程将大大缩短。如果此时名字服务器已将结果（无论是肯定的还是否定的）放在缓存中，它只需要向解析器返回这个结果就行了。即使它没有将结果直接放在缓存当中，它也可能已经获知该域名所在区的权威名字服务器的标识，然后直接向它们查询。

举个例子来说，假设我们的名字服务器已经查找过 *eecs.berkeley.edu* 的地址。在这个过程当中，它将把 *eecs.berkeley.edu* 和 *berkeley.edu* 的名字服务器的名字和地址（还有 *eecs.berkeley.edu* 的 IP 地址）放入缓存。现在解析器向我们的名字服务器查询*baobab.cs.berkeley.edu*的地址，我们的名字服务器将跳过查询根名字服务器这一步。我们的名字服务器认为 *berkeley.edu* 是它所知道的离 *baobab.cs.berkeley.edu* 最近的祖先，于是从 *berkeley.edu* 名字服务器开始查询，见图 2-16。另外，如果我们的名字服务器曾经查找到 *eecs.berkeley.edu* 的地址不存在，下次它再接收到同样的查询时，可以简单地根据缓存做出相应的回答。

缓存除了能加快解析速度外，还能避免访问根名字服务器来回答本地无法回答的查询。这使得我们不用总是依赖根，根也不至于频繁地被访问而负载过重。

生存期

当然，名字服务器不能把数据永远放在缓存中。如果这样，即使权威名字服务器上的数据有了变动，它也绝不会知道。远程名字服务器还会使用这些缓存的数据。因此，包含这些数据的区的管理员要为数据设定一个生存期（time to live），简称TTL。生存期就是名字服务器允许数据在缓存中存放的时间。生存期一过，名字服务器就必须丢弃缓存的数据，并从权威名字服务器上获取新的数据。对于否定缓存数据也是一样；每隔一段时间，名字服务器也要清除否定回答，以防权威名字服务器上已经增加了新的数据。

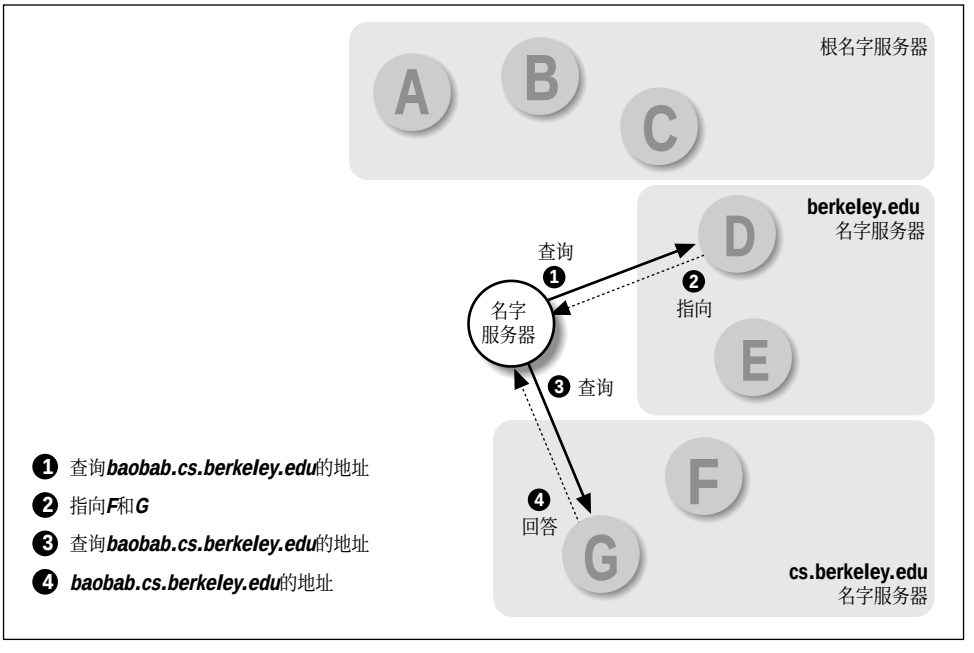


图 2-16 解析 baobab.cs.berkeley.edu

数据生存期的长短主要取决于效率与一致性两者之间的平衡。TTL 越短，区数据在网络中的一致性就越高，因为远程名字服务器将更频繁地查询权威名字服务器以获取新的数据。不过这样一来，也将增加名字服务器的负担，使得对你的区中信息的平均解析时间变长。

TTL 越长，查询你的区中信息的平均解析时间也就越短，因为数据能在缓存中存放更长的时间。缺点就是，如果名字服务器上的数据发生了变动，缓存中的信息将长时间地保持不一致。

讲了这么多理论，你可能已经急不可耐地要试一试了。不过在你建立自己的区和名字服务器之前，还要做点作业，下一章我们就会布置给你。

本章内容：

- 获得BIND
- 选择一个域名

第三章

我该从哪里开始?

最后，小鹿说：“你还记得你叫什么吗？”
她的声音多轻柔甜美啊！
“我真希望我知道！”可怜的爱丽丝想。
她忧伤地回答：“现在我什么也记不起来了。”
“再想想，”小鹿说，“不可能记不起来的。”
爱丽丝想了想，却什么也想不起来。她怯怯地说：
“请你告诉我你叫什么，好吗？这或许能帮我想起来。”
“那你再靠近一点，我来告诉你。”
小鹿说，“我也记不起来了。”

既然你已经了解DNS的理论，就让我们来做些更实际的事情吧。在建立你的区之前，先要获得BIND软件。通常，大部分基于Unix的操作系统都将它作为一个标准部件。不过你常常会想得到一个更新的版本，它具有所有的最新功能和更强的安全性。

一旦获得BIND，你就需要为你的主区（main zone）取一个域名——这可没有听起来那么简单，因为这要求你在Internet名字空间里找到一个合适的位置。一旦你决定好了，就要和你所选择域名的父区的管理员进行联系。

不过，还是一件一件来吧。让我们来先说说从哪里获得BIND。

获得 BIND

如果你计划建立自己的区，并为之运行名字服务器，你首先需要 BIND 软件。即使你想让别人来管理你的区，手边有这个软件也是有好处的。比如，在把数据文件发给你的远程名字服务器的管理员之前，可以用你本地的名字服务器来测试数据的正确性。

大部分商业 Unix 供应商一并提供 BIND 及其他的标准 TCP/IP 网络软件，而且网络软件通常包含在操作系统中，所以你可免费得到 BIND。即使网络软件是单独收费，也很可能你早已经购买了它，因为你知道 DNS 对于网络是必须的，对不对？

但是如果你的 Unix 没有你想要的 BIND 版本，或者你想要最新、最好的版本，你总是能得到 BIND 源代码。多幸运，它是免费。写这本书时的最新（BIND 8.2.3 和 9.1.0 版本）BIND 源代码可以通过匿名 FTP 分别从 ISC（Internet 软件协会）的网站 *ftp.isc.org* 上的 */isc/bind/src/cur/bind-8/bind-src.tar.gz* 和 */isc/bind9/9.1.0/bind-9.1.0.tar.gz* 下载。在大多数常见的 Unix 平台上编译它们是很容易的（注 1）。ISC 在文件 *src/INSTALL* 中列出了 BIND 可被编译的 Unix 风格的操作系统，包括各种版本的 Linux、Digital Unix 和 Solaris 2。此外，还列出了其他一些 BIND 过去支持的 Unix 和非 Unix 风格的（如 MPE，多道程序执行程序）操作系统，不用花费太大力气最新版本的 BIND 就能在上面编译（注 2）。无论你的操作系统是哪一类，我们都强烈建议你阅读 *src/INSTALL* 中的有关部分。附录三中给出了在 RedHat Linux 6.2 上编译 BIND 8.2.3 和 9.1.0 的指令，这个附录相当短。

也许你们当中一些人的操作系统已经带有某个版本的 BIND，但是你在想你是否真的需要最新、最好的 BIND。最新版本的 BIND 都提供哪些新功能呢？下面是一个概述：

安全性补丁

正如已被证明的那样，使用最新版本的 BIND 的最重要原因是：只有最近经过

注 1： 编译 BIND 9 的早期版本(9.1.0 以前的)需要许多技巧，因为这些版本要求 *pthreads*，许多操作系统的 *pthreads* 的实现都不尽相同。BIND 9.1.0 及以后的版本可以以参数 *configure-disable-threads* 编译而得，无需 *pthreads*。

注 2： 据我们所知，实际上，BIND 8.2.3 就能很好地在数种这样的操作系统上编译。

修补的版本才能抵御大多数对名字服务器的攻击，其中一些攻击是广为人知的。BIND 8.2.3 和 BIND 9.1.0 能抵御所有常见的攻击，而 BIND 4.9.8 能抵御其中一些主要的攻击。更早版本的BIND有许多人们熟知的弱点。如果你管理的是一台Internet上的名字服务器，我们强烈建议你使用BIND 8.2.3、BIND 9.1.0，至少要用 BIND 4.9.8，或者任何你读到本书时最新发布的版本。

安全特性

BIND 8和BIND 9支持关于查询、区传送和动态更新的访问控制表。BIND 4.9支持关于查询和区传送的访问控制表，更早版本的BIND根本不支持访问控制表。某些名字服务器，特别是运行在堡垒主机或其他安全性要求很高的主机上，可能需要这些特性。

在第十一章中我们将讲述这些特性。

DNS 更新

BIND 8 和 BIND 9 支持 RFC 2136 中所描述的动态更新标准。这允许被授权的代理（agent）通过发送特殊的更新消息来增加或删除资源记录来更新区数据。BIND 4 服务器不支持动态更新。

我们将在第十章中谈到动态更新。

DNS 通知（NOTIFY）

BIND 8 和 BIND 9 支持区变动通知，它允许一个区的主名字服务器在序列号增加的时候通知该区的辅名字服务器。BIND 4 不支持 NOTIFY。

我们将在第十章中谈到 NOTIFY。

增量区传送

BIND 8.2.3 和 BIND 9 支持增量区传送（incremental zone transfer），它允许辅名字服务器只要求从主服务器获得区变动的部分。这就使得区传送更快、更有效，对于大的、经常变动的区这个特性尤为重要。

配置语法

BIND 8 和 BIND 9 使用的配置语法与 BIND 4 的完全不同的。虽然这些新的配置语法更加灵活、功能也更强大，但是它也需要你从头开始重新学习如何配置 BIND。不过现在有这本书来帮助你了。

我们将在第四章中介绍 BIND 8 和 BIND 9 的配置语法，并将在本书的其余部分中不断地讲到。

附录二中，我们还提供了当前四个最流行的BIND版本（4.9.8、8.1.2、8.2.3和9.1.0）的功能综述。如果你还不确信哪个版本适合你，或者你需要一些特别的BIND特性而又不知道BIND 9是否已经支持，那么就参阅一下附录二吧。

如果读完上述列表且查阅过附录，确信了你需要BIND 8或BIND 9的特性，而且你的操作系统又没有BIND 8和BIND 9名字服务器，那么就下载源代码，然后自己编译生成它。

方便的邮件列表和Usenet新闻组

关于如何将BIND移植到各个版本的Unix上的说明足够写成一本和本书同样厚的书，所以我们只有推荐你去查看BIND用户邮件列表（*bind-users@isc.org*），或是相应的Usenet新闻组（*comp.protocols.dns.bind*）去进一步寻求帮助（注3）。对于BIND 9，有一个单独的邮件列表*bind9-users@isc.org*（注4）。那些阅读并投稿到BIND用户邮件列表的人们会给你的移植工作提供难以估量的帮助。不过在发邮件到列表询问某种移植是否可行之前，一定要检查一下邮件列表的档案库，地址为*http://www.isc.org/ml-archives/bind-users*。同时也要查看一下ISC的BIND网页*http://www.isc.org/products/BIND*，看有没有针对你的操作系统的说明或链接，还可以看看Andras Salamon的DNS资源目录，从中获取一些预编译的BIND软件。这个目录位于*http://www.dns.net/dnsrd/bind.html*，目前有少量预编译二进制代码。

另外一个你可能会感兴趣的邮件列表是*namedroppers*列表。这个列表中的用户都同IETF工作组有关，这个工作组开发对DNS规范的扩展，即DNSEXT。例如，有关对一种新DNS资源记录类型的建议的讨论有可能出现在*namedroppers*邮件列表中，而不是BIND的邮件列表。想了解更多有关DNSEXT章程的信息，请浏览*http://www.ietf.org/html.charters/dnsext-charter.html*。

注3： 想在Internet邮件列表中提问，你要做的只是向该邮件列表的地址发送一封邮件。但是如果你想加入这个列表，你就要给这个邮件列表的维护者发送一封邮件，请求他或她把你的电子邮件地址加入到这个列表中。不要把请求邮件直接发送到这个列表，这被认为是不礼貌的。Internet的习惯是向*list-requeset@domain*发送邮件就能同维护人员联系上，在这里*list@domain*就是邮件列表的地址。所以，比如说你就能通过向*bind-workers-request@isc.org*发送邮件，联系上BIND用户邮件列表的管理员。

注4： 大多数BIND 9的开发者只会阅读*bind9-users*邮件列表。

namedroppers 邮件列表的地址是 *namedroppers@ops.ietf.org*，它通过网关连接到 Internet 新闻组 *comp.protocols.dns.std*。若想加入 *namedroppers* 邮件列表，给 *namedroppers-request@ops.ietf.org* 发送一封正文为“ subscribe namedroppers ”的邮件。

查找 IP 地址

你应该注意到，我们给你的都是一些主机的域名（主机上有许多可通过 FTP 获取的软件），且上面所提到的邮件列表也是以域名的形式出现。这又突显出 DNS 的重要性：你能在 DNS 的帮助下获得多么有价值的软件和建议啊！但不幸的是，这又是一个鸡和蛋的问题。如果没有建立起 DNS，你就不能给包含域名的地址发电子邮件，那你又怎么向列表中的某个人询问如何建立 DNS 呢？

我们虽然能把我们曾提到的主机的 IP 地址告诉你，但是主机的 IP 地址是经常变动的。在这里，我们将谈谈如何暂时借用别人的名字服务器来找到这些信息。只要你的主机连到 Internet 并且安装了 *nslookup* 程序，你就能从 Internet 名字空间中检索信息。例如，查找 *ftp.isc.org* 的 IP 地址，你可以用：

```
%  nslookup ftp.isc.org. 207.69.188.185
```

这行命令指示 *nslookup* 向 IP 地址为 207.69.188.185 的主机上的名字服务器查询 *ftp.isc.org* 的 IP 地址，应该产生如下所示的输出：

```
Server:  ns1.mindspring.com
Address:  207.69.188.185

Name:    pub1.pa.vix.com
Address: 204.152.184.27
Aliases:  ftp.isc.org
```

现在你就可以 FTP 到 *ftp.isc.org* 的 IP 地址 204.152.184.27 了。

我们怎么知道 IP 地址为 207.69.188.185 的主机运行了名字服务器呢？我们的 ISPMindspring，告诉我们的 —— 那是他们的一个名字服务器。你也可以使用你的 ISP 提供的名字服务器。如果他们不提供的话（他们真不该这样！），你可以暂时使用本书中列出的一些名字服务器。你只用它来查找一点儿地址或其他数据，该名字服务器的管理员是不会介意的。不过，如果你总是用你的解析器或其他查询工具使用别人的名字服务器，那么这是非常不礼貌的。

当然，如果你能访问已连接到Internet并配置了DNS的主机，你就能用FTP获取你想要的东西了。

一旦有了可用的BIND版本，就该开始考虑你的域名了。

选择一个域名

选择域名可比听起来要复杂得多，因为必须选择一个名字并找出谁在管理父区（parent zone）。换句话说，你要在Internet域名空间中找出适合你的位置，再查出谁在管理域名空间中这个特别的一角。

挑选域名第一步就是要找出在现有域名空间中你的适当位置。自上而下是最简便的：先找出你所在的顶级域，再找出适合你的顶级域的子域。

注意，要想知道Internet域名空间长得什么样（除了我们已经告诉你的），你就需要能够访问Internet。你并不一定非要有一台已经配置好名字服务的主机，有当然是最好了。如果你没有，可以向其他名字服务器“借”名字服务（就像前面*ftp.isc.org*那个例子中的一样）。

登记员和注册机构

在进一步讨论之前，我们需要定义一些术语：注册机构（registry）、登记员（registrar）和注册（registration）。DNS规范中没有定义这些术语，但是它们已应用于现在Internet名字空间的管理中。

注册机构是一个组织，它负责维护顶级域（实际上是区）的数据文件，这些文件包括对该顶级域的各个子域的授权。在Internet目前的结构下，一个给定顶级域的注册机构不能多于一个。登记员扮演着用户和注册机构之间接口的角色，提供注册和增值服务。他向一个顶级域中每个用户提供注册机构的区数据和其他一些数据（包括联系信息）。

注册是一个过程，通过这个过程，用户告诉登记员将哪个子域授权给哪个名字服务器，同时也向登记员提供联系和计费信息，登记员再将这些变动通知注册机构。

Network Solutions 公司是 *com*、*net*、*org* 和 *edu* 顶级域的惟一注册机构和登记员。现在，还是让我们言归正传。

哪里是我合适的位置呢？

如果你的机构不在美国，你首先要决定你是要一个 *com*、*net* 或 *org* 这样的通用顶级域的子域，还是要一个你自己国家顶级域的子域。通用顶级域并不为美国的机构所独有。如果你的公司是个多国或跨国公司，不适合放在任何一个国家的顶级域中，或者你就是喜欢使用通用顶级域名，而不愿意使用你自己国家的顶级域名，你都可以申请一个通用顶级域名下的域。如果你这样选择，请跳到本章后面的“通用顶级域”一节。

如果你决定使用本国顶级域名下的子域，应该检查一下你们国家的顶级域名是否已经注册；如果已经注册，又是采用什么样的结构。如果你还不太确信你们国家的顶级域的名字是什么，请查阅附录四。

有些国家的顶级域，比如新西兰的 *nz*、澳大利亚的 *au*、以及英国的 *uk*，是按照组织来划分二级域的。它们的二级域的名字，比如 *co* 或 *com* 代表商业实体，反映了组织上的隶属关系。还有一些，比如法国的 *fr* 域和丹麦的 *dk* 域，是按照各个大学和公司来分成了许多子域，例如St. Etienne大学（the University of St. Etienne）的域 *univ-st-etienne.fr*，以及丹麦 Unix 用户组的 *dkuug.dk*。许多顶级域都有其各自的网址描述其结构。如果你不知道你们国家顶级域网址的URL，那么从 *http://www.allwhois.com* 开始，从这里可以链接到那些网址。

如果你的顶级域没有网址解释它是如何组织的，你就得使用像 *nslookup* 这样的工具慢慢摸索，找出你的顶级域的结构。（如果你对我们没有做适当的介绍就直接使用 *nslookup* 感到很不习惯的话，可以先浏览一下第十二章。）例如，下面就是如何使用 *nslookup* 来列出 *au* 域的子域：

```
% nslookup - 207.69.188.185        - 使用位于 207.69.188.185 的名字服务器
Default Server:  ns1.mindspring.com
Address:  207.69.188.185

> set type=ns
> au.                                - 查找 au 区的名字服务器(ns)
Server:  ns1.mindspring.com
Address: 207.69.188.185
```

```
au      nameserver = MUNNARI.OZ.AU
au      nameserver = MULGA.CS.MU.OZ.AU
au      nameserver = NS.UU.NET
au      nameserver = NS.EU.NET
au      nameserver = NS1.BERKELEY.EDU
au      nameserver = NS2.BERKELEY.EDU
au      nameserver = VANGOGH.CS.BERKELEY.EDU
MUNNARI.OZ.AU     internet address = 128.250.1.21
MULGA.CS.MU.OZ.AU       internet address = 128.250.1.22
MULGA.CS.MU.OZ.AU       internet address = 128.250.37.150
NS.UU.NET         internet address = 137.39.1.3
NS.EU.NET         internet address = 192.16.202.11
NS1.BERKELEY.EDU  internet address = 128.32.136.9
NS1.BERKELEY.EDU  internet address = 128.32.206.9
NS2.BERKELEY.EDU        internet address = 128.32.136.12
NS2.BERKELEY.EDU        internet address = 128.32.206.12

> server ns.uu.net.     - 现在查询其中的一个名字服务器 —— 越近越好！
Default Server:  ns.uu.net
Addresses:  137.39.1.3

> ls -t au.  —列出 au 区的数据
             —该区的 NS 记录表明了对子域的授权，并会给出子域的名字。
             —注意，出于安全原因，不是所有的名字服务器都允许列出区的数据。
 [ns.uu.net]
$ORIGIN au.
@                       3D IN NS        mulga.cs.mu.OZ
                        3D IN NS        vangogh.CS.Berkeley.EDU.
                        3D IN NS        ns1.Berkeley.EDU.
                        3D IN NS        ns2.Berkeley.EDU.
                        3D IN NS        ns.UU.NET.
                        3D IN NS        ns.eu.NET.
                        3D IN NS        munnari.OZ
ORG                     1D IN NS        mulga.cs.mu.OZ
                        1D IN NS        rip.psg.COM.
                        1D IN NS        munnari.OZ
                        1D IN NS        yalumba.connect.COM
info                    1D IN NS        ns.telstra.net.
                        1D IN NS        ns1.telstra.net.
                        1D IN NS        munnari.oz
                        1D IN NS        svc01.apnic.net.
otc                     4H IN NS        ns2.telstra.com
                        4H IN NS        munnari.oz
                        4H IN NS        ns.telstra.com
OZ                      1D IN NS        mx.nsi.NASA.GOV.
                        1D IN NS        munnari.OZ
                        1D IN NS        mulga.cs.mu.OZ
                        1D IN NS        dmssyd.syd.dms.CSIRO
                        1D IN NS        ns.UU.NET.
csiro                   1D IN NS        steps.its.csiro
                        1D IN NS        munnari.OZ
                        1D IN NS        manta.vic.cmis.csiro
                        1D IN NS        dmssyd.nsw.cmis.csiro
```

```
                        1D IN NS        zoiks.per.its.csiro
COM                     1D IN NS        mx.nsi.NASA.GOV.
                        1D IN NS        yalumba.connect.COM
                        1D IN NS        munnari.OZ
                        1D IN NS        mulga.cs.mu.OZ
                        1D IN NS        ns.ripe.NET.
> ^D
```

我们使用的基本技巧简单明了：查询顶级域的名字服务器列表（因为只有它们才有相应区的完整信息），然后查询这些名字服务器中的一个，列出被授权子域的名字服务器。

如果你不能分辨你所属子域的名字，可以查找相应区的联系信息，然后给技术联系人发邮件，礼貌地询问，以获得建议。同样地，如果你觉得自己属于某个子域，但不太确信，也可以向管理该子域的人询问，确认一下。

要想知道向谁询问某个子域的信息，要查相应区的 SOA（start of authority，权威起始）记录。在每个区的SOA 记录中，都有一个字段存放该区技术联系人的电子邮件地址（注 5）。（SOA 记录中的其他一些字段提供了该区的主要信息 —— 过一会儿我们将详细讨论。）你同样也可以用 *nslookup* 来查看区的 SOA 记录。

例如，如果你对 *csiro* 子域感兴趣，通过查找 *csiro.au* 的 SOA 记录，就能查出是谁在管理它：

```
%   nslookup - 207.69.188.185
Default Server:  ns1.mindspring.com
Address:  207.69.188.185

>   set type=soa
>  csiro.au.          - 查询 csiro.au 的 SOA 数据
Server:  ns1.mindspring.com
Address: 207.69.188.185

csiro.au
        origin = steps.its.csiro.au
        mail addr = hostmaster.csiro.au
        serial = 2000041301
        refresh = 10800 (3H)
        retry   = 3600 (1H)
        expire  = 3600000 (5w6d16h)
```

注 5： 子域和区有相同的域名，不过，SOA 实际上是属于区而不是域的。区的联系电子邮件地址上的人很可能并不能管理整个子域（因为子域下可能还有一些已被授权的子域），不过他或她一定知道这个子域是干嘛的。

```
                minimum ttl = 86400 (1D)
```

其中，*mail addr* 字段是 *csiro.au* 的联系人的 Internet 地址。要将这个地址转换成 Internet 电子邮件地址格式，需要将地址中的第一个“．”变成“@”符号。于是 *hostmaster.csiro.au* 就变成了 *hostmaster@csiro.au*（注 6）。

使用 whois

whois 服务同样也能帮助你辨认出给定域的内容。不幸的是，有许多 *whois* 服务器（大多数顶级域的管理员都运行着一个），而且它们之间不像名字服务器那样相互通信。所以，使用 *whois* 的第一步是找到正确的 *whois* 服务器。

查找正确的 *whois* 服务器最简单的方法就是从 *http://www.allwhois.com* 开始（见图 3-1）。前面我们曾提到过这个站点上有每个国家代码顶级域网址的列表，上面还有带 *whois* URL 的顶级域列表，这些网页有查询 *whois* 服务器的基于 HTML 的接口。

图 3-1　allwhois.com 站点

注 6：　这种 Internet 邮件地址格式是两种早期 DNS 记录 MB 和 MG 的遗迹：MB（mail box）和 MG（mail group），这些 DNS 记录将 Internet 邮件地址以及邮件组（邮件列表）看成是相应域的子域。MB 和 MG 从来没有被取消过，而 SOA 记录中使用它们过去所规定的地址格式，可能是出于感情上的原因。

将滚动条拉到“ Australia(au) ”，点击“ Jump to Whois ”进入一个页面，通过它就能直接进入 *csiro.au*，如图 3-2 所示。

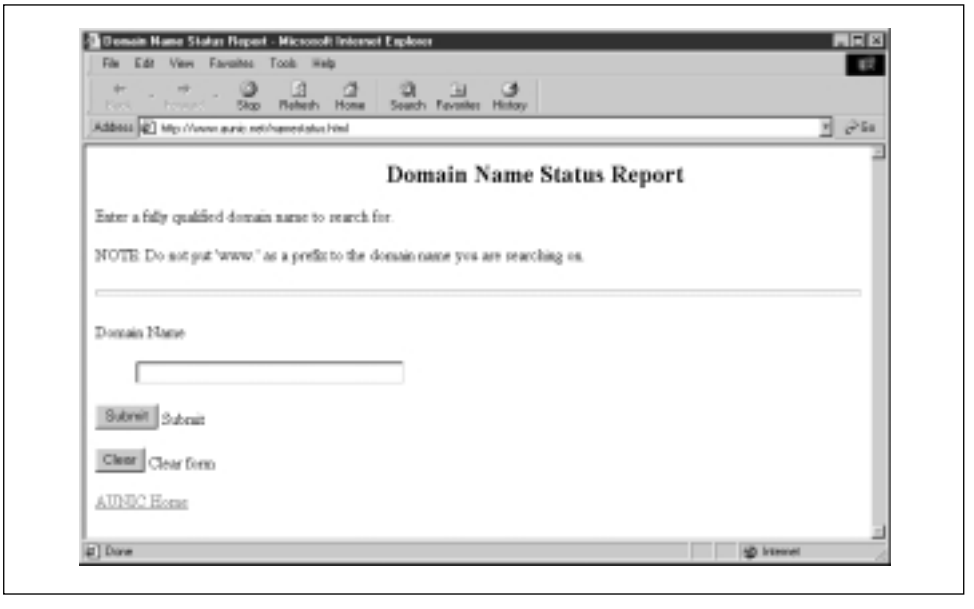


图 3-2　au 的 whois 服务器的 Web 接口

点击“ Submit ”就可以检索到如图 3-3 所示的信息。

或许由 WebMagic 在网上提供的统一 *whois* 查找服务更有趣。它们的网址是 *http://www.webmagic.com/whois/index.html*，它可以让你选择包含你所查找子域的顶级域（有时是二级域），然后就可以和对应的 *whois* 服务器联系了。

显然，如果你要查找一个美国之外的域的联系人，这两个网址都是非常有用的。

一旦你找到了正确的网址或联系人，很可能就找到了登记员。在美国以外，大多数域都只有一个登记员。也有少数，比如丹麦的 *dk* 和英国的 *co.uk* 和 *org.uk*，就有好几个登记员。不过按照上述步骤你一样可以找到它们。

在美国

我们首先讲了国际域的情况，但是如果你是在美国怎么办呢？

图 3-3　从 au whois 服务器获得的有关 csiro.au 的信息

如果你是在美国，那么你属于哪个域将主要取决于你的机构是干什么的，你希望你的域名是什么样的，以及你愿意付多少钱。如果你的机构适合于下述条件中的一个，你就应该加入 *us* 顶级域：

- K-12（幼儿园到十二年级）的学校
- 社区大学和技术性业余学校
- 州和本地政府机构

即使你不是上述任何一种，如果你想要一个能指示出你位置的域名，比如 *acme.*

boulder.co.us，那么你就可以在*us*顶级域中注册。*us*域对三级域以下的子域的授权大都以“位置”命名（通常是城市或郡）；二级域对应于相应的美国邮政服务的两个字母的州名缩写（回想一下我们在第二章中“Internet域名空间”一节所讲到的）。所以，比如说，如果你所需的只是一个子域来容纳你在科罗拉多州Colorado Springs地下室里的两台互联的主机，你就可以注册*toms-basement.colorado-springs.co.us*。

最后，是花费的问题。通常注册*us*顶级域的子域比在*com*、*net*或*org*下注册要便宜，有时甚至是免费的。

如果想了解更详细的有关*us*域结构和管理规则的信息，请查阅美国NIC的网址*http://www.nic.us*。

当然，在美国的人还可以申请一个通用顶级域，比如*com*、*net*或*org*下的子域。只要你不申请已存在的域名，你想要什么样的都可以。本章后面一部分中我们将谈到在通用顶级域下如何注册。

us域

让我们通过一个例子来告诉你如何搜索*us*域的名字空间，以获得一个理想域名的方法。比如说，你要帮助你儿子在科罗拉多Boulder的幼儿园注册一个域名。

使用你在一台位于CU的主机上的账户（从大学时开始的），你能检查是否存在一个Boulder域。（如果你没有那儿的账户，但是能连接到Internet，你仍然可以使用*nslookup*查询一个已知的名字服务器。）

```
%  nslookup
Default Server:  boulder.colorado.edu
Address: 128.138.238.18, 128.138.240.1

>   set type=ns
> co.us.                    - 查找co.us的名字服务器
Default Server:  boulder.colorado.edu
Address:  128.138.238.18, 128.138.240.1

co.us   nameserver = VENERA.ISI.EDU
co.us   nameserver = NS.ISI.EDU
co.us   nameserver = RS0.INTERNIC.NET
co.us   nameserver = NS.UU.NET
co.us   nameserver = ADMII.ARL.MIL
co.us   nameserver = EXCALIBUR.USC.EDU
```

这里列出了所有*co.us*的名字服务器的名字。现在，将服务器改变为某个*co.us*的名字服务器，比如说*venera.isi.edu*，再检查是否有子域（你现在还没有退出*nslookup*）：

```
> server venera.isi.edu.      - 将服务器改为 venera.isi.edu
Default Server:  venera.isi.edu
Address:  128.9.0.32

> ls -t co.us.          - 列出 co.us 区的数据以查找 NS 记录
[venera.isi.edu]
$ORIGIN co.us.
@                       1W IN NS        NS.ISI.EDU.
                        1W IN NS        RS0.INTERNIC.NET.
                        1W IN NS        NS.UU.NET.
                        1W IN NS        ADMII.ARL.MIL.
                        1W IN NS        EXCALIBUR.USC.EDU.
                        1W IN NS        VENERA.ISI.EDU.
officemate1.monument    1W IN NS        ns1.direct.ca.
                        1W IN NS        ns2.direct.ca.
la-junta                1D IN NS        ns2.cw.net.
                        1D IN NS        usdns.beltane.com.
                        1D IN NS        usdns2.beltane.com.
morrison                1W IN NS        NS1.WESTNET.NET.
                        1W IN NS        NS.UTAH.EDU.
littleton               1W IN NS        NS1.WESTNET.NET.
                        1W IN NS        NS.UTAH.EDU.
mus                     1W IN NS        NS1.WESTNET.NET.
                        1W IN NS        NS.UTAH.EDU.
ci.palmer-lake          1W IN NS        DNS1.REGISTEREDSITE.COM.
                        1W IN NS        DNS2.REGISTEREDSITE.COM.
co.adams                1W IN NS        ns1.rockymtn.net.
                        1W IN NS        ns2.rockymtn.net.
[...]
```

啊！这里有这么多东西！有名叫*la-junta*、*morrison*、*littleton*、*mus*的子域，还有许多其他的。甚至还有一个子域是Boulder的（一点也不奇怪，就叫*boulder*）：

```
boulder                 1W IN NS        NS1.WESTNET.NET.
                        1W IN NS        NS.UTAH.EDU.
```

你如何找到与*boulder.co.us*管理员联系的方法呢？你可以试着用*whois*，不过*boulder.co.us*不是个顶级国家域或是通用顶级域的子域，你很可能什么也找不到。还好，美国的NIC提供了每个*us*三级子域的联系人的电子邮件地址，地址为*http://www.isi.edu/in-notes/us-domain-delegatd.txt*。如果在那也找不到你想要的，你还可以用*nslookup*来查找*boulder.co.us*区的SOA记录，就像找到*csiro.au*的联系人一样。不过阅读发往SOA记录中地址的邮件的人也许并不处理注册（因为区的技术性功能和

管理功能可能是分开的），不过这些人应该知道谁会受理注册，并告诉你如何同他们联系。

下面就是如何用 *nslookup* 找出 *boulder.co.us* 的 SOA 记录：

```
%  nslookup
Default Server:  boulder.colorado.edu
Address:  128.138.238.18, 128.138.240.1

>  set type=soa
>  boulder.co.us.      - 查找 boulder.co.us 的 SOA 记录
Default Server:  boulder.colorado.edu
Address:  128.138.238.18, 128.138.240.1

boulder.co.us
        origin = ns1.westnet.net
        mail addr = cgarner.westnet.net
        serial = 200004101
        refresh = 21600 (6H)
        retry   = 1200 (20M)
        expire  = 3600000 (5w6d16h)
        minimum ttl = 432000 (5D)
```

同在*csiro.au*的例子中一样，使用之前要将*mail addr*字段中的第一个“ . ”换成“ @ ”。这样，*cgarner.westnet.net* 就变成了 *cgarner@westnet.net*。

要获得*boulder.co.us*的一个子域的授权，你要从*http://www.nic.us/cgi-bin/template.pl* 下载一份注册表模板，再发给联系人。

不过如果你发现你所在的地方还没有创建子域，那么就仔细读一读 *http://www.nic.us/register/locality.html* 的 *us* 域的授权政策，填好 *http://www.nic.us/cgi-bin/template.pl* 的注册表。

通用顶级域

正如我们前面提到的，你可能有很多理由需要申请一个通用顶级域（如，*com*、*edu* 或 *org*）的子域：你在一家多国或跨国公司工作，你认为以“ com ”结尾的域人们会更熟悉，或者仅仅是因为这样要好听一些。让我们一起来看一个在通用顶级域下选择域名的小例子。

假设你是明尼苏达州霍普金斯的一个点子库的网络管理员。你刚刚通过一个商业ISP获得了与Internet的连接。你的公司以前连UUCP这样的链路都没有，所以你现在肯定还没有注册到Internet名字空间。

由于你是在美国，你可以选择加入*us*域或是某个通用顶级域。不过，你们的点子库世界闻名，所以*us*域并非最佳选择。选择*com*下的子域会更好一些。

你们的点子库名叫The Gizmonic Institute，所以你觉得*gizmonics.com*是个合适的域名。现在你就要检查一下*gizmonics.com*这个名字是否已经有人使用了，可以用你在UMN上的账户：

```
% nslookup
Default Server:  ns.unet.umn.edu
Address:  128.101.101.101

> set type=any
> gizmonics.com.    - 查找 gizmonics.com 的任意记录
Server:  ns.unet.umn.edu
Address:  128.101.101.101

gizmonics.com   nameserver = NS2.SFO.WENET.NET
gizmonics.com   nameserver = NS1.SFO.WENET.NET
```

喔！似乎已经有人使用*gizmonics.com*这个域名了（真难以置信!）（注7）。*gizmonic-institute.com*是有点长，但还算直观：

```
% nslookup
Default Server:  ns.unet.umn.edu
Address:  128.101.101.101

> set type=any
> gizmonic-institute.com.        - 查找 gizmonic-institute.com 的任意记录
Server:  ns.unet.umn.edu
Address:  128.101.101.101

*** ns.unet.umn.edu can't find gizmonic-institute.com.: Non-existent host/domain
```

还好，*gizmonic-institute.com*还是空着的，所以接下来你就该进行下一步了：挑选登记员。

注7： 实际上，*gizmonics.com*归Joel Modgson所有，他预先设想了The Gizmonic Institute和Mystery Science Theater 3000。

选择登记员

选择登记员？欢迎进入崭新的竞争世界！在1999年春之前，有一家公司Network Solutions Inc.，既是*com*、*net*、*org*和*edu*的注册机构又是登记员。要注册任何一个通用顶级域的子域，你都必须到Network Solutions公司来。

在1999年6月，管理域名空间的组织ICANN（我们在上一章中提到过的），在*com*、*net*和*org*的登记员功能中引入了竞争。现在有许多*com*、*net*和*org*的登记员可供选择，其名单参见*http://www.internic.net/regist.html*。

我们无法告诉你如何选择登记员，但是看看它们提供的价格和其他服务将会对你有帮助。比如，看你是否能得到令人满意的注册服务。

检查你的网络是否已经注册

在继续以前，你应该先检查一下你的IP网络是否已经注册。有的登记员不会将子域授权给未注册网络上的名字服务器，而网络注册机构（待会儿我们会谈到它）也不会将对应于未注册网络的*in-addr.arpa*区授权。

一个IP网络定义了一段IP地址。例如，网络15/8是由所有从15.0.0.0到15.255.255.255的IP地址组成的。网络199.10.25/24是从199.10.25.0开始，到199.10.25.255结束的。

InterNIC曾经是所有IP网络的官方来源，他们为每个连接到Internet的网络分配IP地址，并保证互不冲突。现在，InterNIC原来的角色大部分已经被Internet服务提供商（ISP）所取代，他们从自己的网络中分配一段地址空间给用户使用。如果你的网络地址是来自于ISP，那么你的上一级网络（如，ISP的网络地址空间）很可能已经注册过了。也许，你还想检查一下你的ISP的网络是否真的被注册过。如果你的ISP没有注册他们的网络，除了可以批评他们外，你自己不能（也不可能）做些什么。如果你确信他们已经注册过，你就可以跳过这一节的其余部分，继续往前看。

选读：关于 CIDR

以前，当我们撰写本书第一版的时候，Internet 的 32 位地址空间被分成了三大类：A 类、B 类和 C 类。A 类网络，用 IP 地址的第一个字节（前 8 位）标识网络，而剩下的位被管理网络的机构分配给网络上不同的主机。大多数拥有 A 类网络的机构还将其网络分成子网络（subnetwork）或子网（subnet），给寻址机制又添加了一个层次。B 类网络用两个字节来标识网络、两个字节标识主机。C 类网络用三个字节来标识网络、另一个字节来标识主机。

遗憾地是，这种大 / 中 / 小的网络系统并不适合每个人。许多机构用 C 类网络（最多可以容纳 254 台主机）太小，用 B 类网络（最多可容纳 65534 台主机）又太大。不管怎么样，许多这样的机构被分配了 B 类网络。结果，很快 B 类网络就变得不够用了。

为了解决这个问题，给各个机构创建大小正合适的网络，Classless Inter-Domain Routing（无类域间路由），简称为 CIDR（读作 cider）就应运而生了。顾名思义，CIDR 没有采用老的 A 类、B 类和 C 类这样的规定。它不是用一个、两个或三个字节来作为网络标识符的，分配者可以用IP地址中任意数目的邻近位作为网络标识符。举个例子说，如果一个机构需要大约四倍于 B 类网络大小的地址空间，分配者就可以用 14 位来作为网络标识符，剩下 18 位（B 类网络大小的四倍）的地址空间供使用。

CIDR 的出现很自然使原来的分类方法过时 —— 虽然在人们的闲谈中还是会经常使用。现在，要指定某个 CIDR 网络，就要指出把哪个高位值分配给了机构，用点分字节表示法表示，还要指出用几位来标识网络。两者之间用“ / ”隔开。15/8 是老的 A 类网络大小的网络，以 00001111 开头。老的 B 类网络大小的网络 128.32.0.0 现在写做 128.32/16。而网络 192.168.0.128/25 是由从 192.168.0.128 到 192.168.0.255 的 128 个 IP 地址组成的。

然而，如果你的网络是由 InterNIC 分配的，像过去那样，或者你就是一个 ISP，你就应该检查一下你的网络是否已经注册。你到哪里去查你的网络是否注册了呢？哎呀!当然是注册网络的组织呀。这个组织叫做网络注册机构（network registry）（还能叫什么呢？），它们分布于世界各地，受理网络注册。在西半球是 ARIN，the

American Registry of Internet Numbers（*http://www.arin.net/*），负责分配IP地址空间以及注册网络。在亚太地区是APNIC，the Asia Pacific Network Information Center（*http://www.apnic.net/*）。在欧洲是RIPE Network Coordination Centre（*http://www.ripe.net/*）。每个注册机构也可以将一个地区的注册权分派出去，例如，ARIN将墨西哥和巴西的注册权授权给各自国家。一定要向管理你国家的注册机构查询。

如果你不太确信你的网络是否已经注册，最好是用各个注册机构提供的whois服务来查找你的网络。这里是每个注册机构的*whois*网页的URL：

ARIN

http://www.arin.net/whois/index.html

APNIC

http://whois.apnic.net

RIPE

http://www.ripe.net/cgi-bin/whois

如果你发现你的网络还没有注册，在建立你的*in-addr.arpa*区以前一定要注册。每个注册机构注册网络的程序都不同，不过几乎都要交钱（不幸的是，从你手中交到他们手中）。

你可能会发现你的网络已经分配给了你的ISP。如果是这样，你就不用单独再向网络注册机构注册了。

一旦你所有连接到Internet的主机都在已注册的网络上时，你就该注册你的区了。

注册你的区

不同的登记员有不同的注册策略和程序，不过有一点是相同的，那就是他们都是通过网站在线受理注册的。你若按本章前面讲的那样已经选择登记员，那就知道该到哪个网站进行申请了。

所有登记员都需要的基本信息是你的名字服务器的域名和地址，以及有关向你寄送账单或收取费用的信用卡的信息。如果你没有连接到Internet，那么请给他们可以作

为你名字服务器的 Internet 主机的地址。有的登记员还要求你的区要有已经能运行起来的名字服务器。(那些不要求名字服务器已经转起来的登记员可能也会要求你预计这些名字服务器什么时候能够运行起来。)如果你的登记员要求你的区要有已经能运行的名字服务器，就请跳到第四章。然后再与你的登记员联系，提供这些必须的信息。

大多数登记员还会要求一些关于你组织的信息，包括你的区的管理和技术支持联系人（可以是同一个人）。如果你的联系人还没有在登记员的 *whois* 数据库中注册，你还得提供一些注册用的信息。其中包括他们的名字、邮寄地址、电话号码以及电子邮件地址。如果他们已经在 *whois* 中注册过了，就只需要在注册时给出他们的 *whois* “名字编号”（一个惟一的字母数字 ID）。

还有一个注册新区时应该提到的问题，那就是费用。大多数登记员是商业企业，对注册域名收取费用。Network Solutions 是最早的 *com*、*net* 和 *org* 登记员，对在通用顶级域下注册的子域每年收取 35 美元。(如果你已经有一个 *com*、*net* 和 *org* 下的子域，而最近却没有收到来自 Network Solutions 的账单，你最好用 *whois* 检查一下你的联系信息，确信他们有你当前的地址和电话号码。)

如果你是直接连接到 Internet 上的，你应该使与你的 IP 网络对应的 *in-addr.arpa* 区也授权给你。例如，你公司分配的是网络 192.201.44/24，你也应该管理 *44.201.192.in-addr.arpa* 区。这将使你能控制你网络上主机的 IP 地址到名字的映射。第四章也解释了如何建立你的 *in-addr.arpa* 区。

在前面一节中，我们要你找到几个问题的答案：你的网络是否是某个 ISP 网络的一部分？你的网络或者说你所在的ISP的网络是否已经注册？在哪个网络中注册的？要想使你的 *in-addr.arpa* 区授权给你，你就需要这些答案。

如果你的网络是一个大网络的一部分，而它已经注册给某个ISP 了，你应该和这个 ISP 联系，让它把 *in-addr.arpa* 区中相应的子域授权给你。每个 ISP 对建立 *in-addr.arpa* 授权都有不同的程序。你的ISP的网页是获取这方面信息的好地方。如果你不能从那儿得到信息，试着查找与你ISP 的网络相对应的 *in-addr.arpa* 域的 SOA 记录。例如，如果你的网络是UUNet的153.35/16网络的一部分，你可以查找 *35.153.in-addr.arpa* 的 SOA 记录，找到这个区的技术支持联系人的电子邮件地址。

如果你的网络是直接向某个地区性注册机构注册的，与他们联系，让他们帮你把你的 *in-addr.arpa* 区注册。每个注册机构的授权程序信息都能从它们的网站上获得。

既然已经注册了你的区，现在最好用点儿时间把你的东西整理一下。下一章，我们将介绍如何建立起你自己的名字服务器。

第四章

建立 BIND

本章内容：

- 我们的区
- 建立区数据
- 建立 BIND 配置文件
- 缩写
- 主机名检查（BIND 4.9.4 及后续版本）
- 工具
- 运行主名字服务器
- 运行辅名字服务器
- 增加更多的区
- 接下来是什么？

“ 它看上去还不错，” 当她看完之后说，“ 不过，
就是有点儿难以理解。”（你瞧，她不愿承认她根本
就不懂，即便是向她自己承认。）“ 不知道为什么，
我好像知道些什么 —— 但是又说不出来是什么。”

如果你是认真地阅读本书的每一章，现在你可能就有点儿急着要运行一个名字服务器了。这一章就是教你如何运行一个名字服务器的。让我们来建立一些名字服务器吧。你们中还有一些人可能是看了目录后直接跳到本章来的。如果是这样，那可要小心一些，因为我们可能会用到前几章的概念，希望你能理解它们的意思。

有很多因素会影响到如何建立你的名字服务器。其中最重要的因素就是你是以何种方式接入 Internet 的：完全接入（比如，直接 FTP 到 *ftp.uu.net*），有限接入（受一个安全防火墙的限制），或者是根本就没有接入。在本章中，我们假设你是完全接入的。至于其他一些情况，我们将在第十一章中进行讨论。

在本章中，我们将为一些虚构的区建立两个名字服务器，你可以照着这个例子建立你自己的区。我们将详细讲述有关的话题，保证你的头两个名字服务器能够运行起来。在接下来的几章中，我们会补漏拾遗，并进行深入地探讨。如果你已经建立起你自己的名字服务器，就只需要浏览本章，熟悉一下我们使用的术语，或是检查一下你在建立你的服务器时有没有漏掉些什么。

我们的区

我们虚构的是一个大学的区，电影大学（Movie University），它研究电影工业的方方面面，并且探索发行电影的新方法。其中最有前途的一个计划就是研究利用网络作为发行媒体。在访问过我们的登记员网站以后，我们决定使用 *movie.edu* 这个域名。最近我们又获得了连接到 Internet 的许可。

电影大学现在有两个以太网，还计划着再建一个到两个。现有的以太网网络号分别是 192.249.249/24 和 192.253.253/24。我们主机表的一部分包括如下内容：

```
127.0.0.1       localhost

# 这是我们的杀手机器

192.249.249.2   robocop.movie.edu robocop
192.249.249.3   terminator.movie.edu terminator bigt
192.249.249.4   diehard.movie.edu diehard dh

# 这些机器老化严重，不久将被替换

192.253.253.2   misery.movie.edu misery
192.253.253.3   shining.movie.edu shining
192.253.253.4   carrie.movie.edu carrie

# 蠕虫洞是一种虚构的场景，它能迅速地在长距离上传输空间游客，
# 被认为是不稳定的。蠕虫洞和路由器之间的惟一区别是，
# 路由器传输包不会那么迅速——特别是我们的路由器。

192.249.249.1   wormhole.movie.edu wormhole wh wh249
192.253.253.1   wormhole.movie.edu wormhole wh wh253
```

网络如图 4-1 所示。

建立区数据

我们建立电影大学名字服务器的第一步是把主机表中的数据转换为相应的DNS区数据。主机表数据的DNS版本有多个文件。一个文件将所有主机名映射到地址，其他一些文件则将地址映射回主机名。名字到地址的查找有时称为正向映射（forward mapping），而地址到名字的查找称为反向映射（reverse mapping）。每个网络都有包含它自己的反向映射数据的文件。

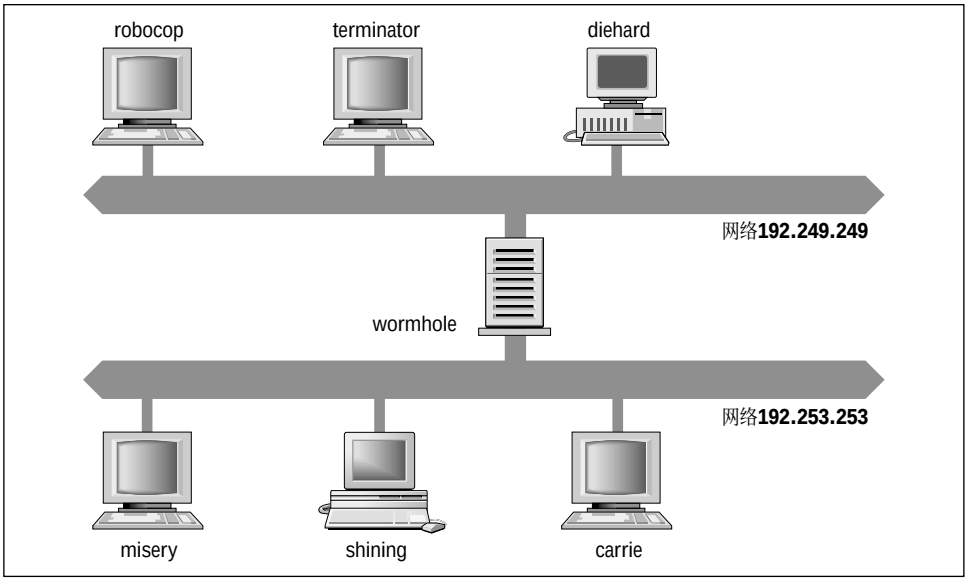


图 4-1 电影大学网

本书中约定：将主机名映射到地址的文件称为 *db.DOMAIN*。对于 *movie.edu* 来说，这个文件就称为 *db.movie.edu*。将地址映射到主机名的文件叫做 *db.ADDR*，这里 ADDR 是去掉尾部 0 的网络号或指定的网络掩码。在我们这个例子中，这些文件就是 *db.192.249.249* 和 *db.192.253.253*，每个对应一个网络。*db* 是数据库（database）的简称。我们将把这些 *db.DOMAIN* 和 *db.ADDR* 文件统称为区数据文件（zone data file）。还有其他一些区数据文件：*db.cache* 和 *db.127.0.0*，这些文件是系统开销（overhead）。每个名字服务器都必须有这些文件，这些文件对每个名字服务器来说或多或少都是相同的。

名字服务器需要一个配置文件来把所有这些区数据文件绑定在一起，对于 BIND 4 来说这个文件通常就是 */etc/named.boot*。对于 BIND 8 或 9 来说，这个文件通常是 */etc/named.conf*。区数据文件的格式对所有 DNS 实现来说都是一样的：它被称为主文件格式（master file format）。而另一方面，配置文件的格式对于不同的名字服务器实现来说是不同的，在这里我们指的是 BIND。

区数据文件

区数据文件中的大部分条目被称为 DNS 资源记录（resource record）。DNS 查找是

不区分大小写的，所以你可以在区数据文件中输入大写、小写或者混合着写。我们倾向于全部使用小写。虽然查找与大小写无关，但是大写是被系统保留的。如果你为 *Tootsie.movie.edu* 在区数据中添加了一个记录，即使人们查找 *tootsie.movie.edu* 也能找到这个记录，只不过域名中是大写的“ T ”。

资源记录必须从一行的第一列开始。我们在本书中所显示的示例文件中的资源记录确实是从第一列开始的，只是因为本书的排版格式而看起来缩进去了。在DNS RFC中，表示资源记录的那些例子都是按照特定顺序排列的。大多数人都选择使用那样的顺序，我们也是如此，但是并不一定非要按照这种顺序。排列区数据文件中资源记录的顺序如下：

SOA 记录
: 指示该区的权威

NS 记录
: 列出该区的一个名字服务器

其他记录
: 有关该区中主机的数据

本章中讲到的其他记录包括：

A
: 名字到地址的映射

PTR
: 地址到名字的映射

CNAME
: 规范名字（相对于别名而言）

毫无疑问，你们当中那些对主文件格式已经有些了解的人在查看我们的数据时会说“ 用其他方式的话，看起来会更短些... ”我们没有在区数据中使用缩写或者简写，至少在开始时没有，这样你就能了解每个资源记录的完整语法。一旦你理解了这些完整的版本，我们就会回过头来“ 压缩 ”这些文件。

注释

区数据文件如果包含有注释和空行的话，会更容易理解一些。注释是以分号开头的，到行尾处结束。正如你可能猜到的那样，名字服务器会忽略注释和空行。

设定区默认的 TTL 值

在开始写区数据文件之前，必须先搞清楚正在使用的是哪个版本的 BIND。因为设定区默认的生存期的方法在 BIND 8.2 中有所变化。在 BIND 8.2 以前，SOA 记录中最后一个字段设置区默认的TTL 值。就在BIND 8.2出来之前，公布了RFC 2308，将 SOA 记录中最后一个字段的含义改为了“ 否定缓存 TTL ”（negative caching TTL）。它的意思是一个远程名字服务器能将区的否定响应（negative response）缓存多长时间，否定响应就是报告某个域名或某个域名的某个类型的数据不存在。

那么在 BIND 8.2 及后续版本中该如何设定 TTL 值呢？使用新的 $TTL 控制语句。$TTL 指定了在文件中该语句后（但在其他$TTL 语句之前）所有记录的生存期，并且没有显式的 TTL。

名字服务器在查询响应中提供这个 TTL 值，允许其他服务器将数据在缓存中存放 TTL 所指定的时间。如果你的数据不是经常变动或变动不大，可以考虑将TTL 默认值设为几天。1周大概是使之有意义的最大值了。像1小时这样短的值也可以用，但是我们通常不建议使用小于 1 小时的值，这是考虑到由此引起的 DNS 流量。

因为我们使用的是新版本的BIND，所以需要用$TTL语句为我们的区设定一个默认的 TTL。3 小时对我们来说正好，所以我们区数据文件的第一条语句就是：

```
$TTL 3h
```

如果你运行的是比 BIND 8.2 更早的名字服务器，不要添加 $TTL 语句 —— 名字服务器不会认得它，而是把它作为语法错误对待。

SOA 记录

每个这样的文件中接下来的条目（在 BIND 8.2 以前的版本中是第一个条目）就是 SOA（start of authority）资源记录。SOA 记录表示对该区数据而言，这个名字服

务器就是最好的信息来源。根据这个 SOA 记录，我们的名字服务器就享有对区 *movie.edu* 的权威。每个 *db.DOMAIN* 和 *db.ADDR* 文件都要有 SOA 记录。每个区数据文件中允许有一个也只允许有一个 SOA 记录。

我们把下面的 SOA 记录添加到 *db.movie.edu* 文件中：

```
movie.edu. IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                            1         ; 序列号
                            3h        ; 3 小时后刷新
                            1h        ; 1 小时后重试
                            1w        ; 1 周后期满
                            1h )      ; 否定缓存 TTL 为 1 小时
```

名字 *movie.edu.* 必须从文件的第一列开始。要确保这个名字是以“ . ”结尾的，正如我们这里所写的，否则产生的结果就会让你感到非常惊讶。(我们会在本章后面解释其原因。)

IN 表示 Internet。这是一个数据类 —— 还有其他一些类，但是都没有被广泛使用。我们的例子只使用 IN 类。类字段是可选的。如果没有指明类，名字服务器就根据指示它读取该文件的配置文件中的语句来决定类。在本章中我们也会看到这种情况的。

SOA 后面的第一个名字（*ternimator.movie.edu.*）是 *movie.edu* 区的主名字服务器的名字。第二个名字（*al.robocop.movie.edu.*）是管理该区的人的电子邮件地址（如果你把第一个“ . ”换成“ @ ”的话）。你经常会看到将 *root*、*postmaster* 或 *hostmaster* 作为电子邮件地址。名字服务器不会使用这个邮件地址 —— 它只对人有意义。如果你对该区有任何疑问，可以给列出的电子邮件地址发送邮件消息。BIND 4.9 及其后续版本还为此提供了另外一种资源记录类型：RP（responsible person，负责人）。将会在第七章中讨论 RP 记录。

圆括号可以使 SOA 记录跨越多行。SOA 记录的圆括号中的大部分字段是供辅名字服务器使用的，我们会在本章后面介绍辅名字服务器时再讨论。现在只需假定这些值都是合理的。

我们在 *db.192.249.249* 和 *db.192.253.253* 文件的开头都添加了类似的 SOA 记录。在这些文件中，我们将 SOA 记录中的第一个名字从 *movie.edu.* 改为相应的 *in-addr.arpa* 区的名字，分别为 *249.249.192.in-addr.arpa.* 和 *253.253.192.in-addr.arpa.*。

NS 记录

我们在每个文件中添加的下一个条目是NS（name server，名字服务器）资源记录。为我们区的每个权威名字服务器都添加一个 NS 记录。下面是 *db.movie.edu* 文件中的 NS 记录：

```
movie.edu.  IN NS  terminator.movie.edu.
movie.edu.  IN NS  wormhole.movie.edu.
```

这些记录表明区 *movie.edu* 有两个名字服务器。这些名字服务器运行在主机 *terminator. movie.edu* 和 *wormhole.movie.edu* 上。因为像 *wormhole.movie.edu* 这样的多宿主主机和网络连接得非常好，因此对于名字服务器来说是极好的选择。它可以被多个网络上的主机直接访问，如果它同时也作为路由器，由于它总是被严密监视着，所以不会经常停机。我们将会在第八章中更详细地讨论将你的名字服务器放置在哪里。

同 SOA 记录一样，我们也将 NS 记录添加到 *db.192.249.249* 和 *db.192.253.253* 文件中。

地址和别名记录

接下来，创建名字到地址的映射。我们在 *db.movie.edu* 文件中添加下列资源记录：

```
;
; 主机地址
;
localhost.movie.edu.  IN A     127.0.0.1
robocop.movie.edu.    IN A     192.249.249.2
terminator.movie.edu. IN A     192.249.249.3
diehard.movie.edu.    IN A     192.249.249.4
misery.movie.edu.     IN A     192.253.253.2
shining.movie.edu.    IN A     192.253.253.3
carrie.movie.edu.     IN A     192.253.253.4
;
; 多宿主主机
;
wormhole.movie.edu.   IN A     192.249.249.1
wormhole.movie.edu.   IN A     192.253.253.1
;
; 别名
;
bigt.movie.edu.       IN CNAME terminator.movie.edu.
```

```
dh.movie.edu.          IN CNAME diehard.movie.edu.
wh.movie.edu.          IN CNAME wormhole.movie.edu.
wh249.movie.edu.       IN A     192.249.249.1
wh253.movie.edu.       IN A     192.253.253.1
```

估计你不会对前面两个代码块感到有什么好奇怪的。A 代表地址（address），每个资源记录都把一个名字映射成一个地址。*wormhole.movie.edu* 是一个路由器，它的名字对应有两个地址，于是就有两个资源记录。同主机表查找不同，对于一个名字，一次DNS查找可以返回多个地址。查找*wormhole.movie.edu*就会返回两个地址。如果查询者和名字服务器处于同一网络，为了获得更好的性能，一些名字服务器会把离查询者最近的地址放在回答的最前面。这个特性被称为“ 地址排序 ”（address sorting），我们会在第十章中加以说明。如果没有提供地址排序功能，这些地址就会轮流作为回答返回，所以下一次回答和前一次回答中地址的排列顺序都是不同的。这个“ 循环反复 ”的（round robin）特性首先是在 BIND 4.9 中实现的。

第三个代码块是主机表别名。对于前三个别名，我们创建了 CNAME（canonical name，规范名）资源记录。不过，我们为其他两个别名创建了地址记录，等会儿我们会详细说明的。CNAME记录将别名映射到它的规范名。名字服务器处理CNAME记录和主机表中处理别名的方式不同。当名字服务器查找一个名字而找到了一个 CNAME 记录时，它会用规范名来替换这个名字，然后再查找这个新的名字。譬如当名字服务器查找 *wh.movie.edu* 时，它找到了一个指向 *wormhole.movie.edu* 的 CNAME 记录。接着就会开始查找 *wormhole.movie.edu*，并返回两个地址。

关于像 *bigt.movie.edu* 这样的别名有一件事情你是要牢记的：他们永远不能出现在资源记录的右边。换句话说：在资源记录的数据部分总是要使用规范名（比如，*terminator.movie.edu*）。注意，我们刚才创建的 NS 记录使用的就是规范名。

最后两个条目解决了一个特殊问题。假设你有一个*wormhole.movie.edu*那样的路由器，而你想检查其中的一个接口。一个常用的排错技巧就是 *ping*（译注1）这个接口看是否会有响应。如果你 *ping* 了 *wormhole.movie.edu* 这个名字，当查找这个名字时，名字服务器会返回两个地址。*ping* 会使用返回的第一个地址。但哪一个地址会在前面呢？

在用主机表时，我们可以选择使用 *wh249.movie.edu* 或是 *wh253.movie.edu*，这两个

注译 1： Unix 或 Windows 操作系统的一个系统命令。

名字各自对应该主机的一个地址。为了在 DNS 中提供相同的功能，我们没有把 *wh249.movie.edu* 和 *wh253.movie.edu* 设置为别名（CNAME 记录）。这会导致查找别名时将返回 *wormhole.movie.edu* 的两个地址。于是我们使用了地址记录。现在要检查 *wormhole.movie.edu* 上的 192.253.253.1 接口是否正常，我们就可以 *ping* 一下 *wh253.movie.edu*，而它是指向惟一地址的。对 *wh249.movie.edu* 来说也一样。

把这作为一条通用规则来说就是：如果一台主机是多宿主的（具有不止一个网络接口），让每个接口地址对应一个惟一的别名，再为这个别名创建一个地址（A）记录。为每个对所有地址都通用的别名创建一个 CNAME 记录。

现在就不要把 *wh249.movie.edu* 和 *wh253.movie.edu* 这样的名字告诉你的用户了。这些名字只是用于进行系统管理的。如果用户知道可以使用 *wh253.movie.edu* 这样的名字，而当这些名字在某些地方（比如，在 *.rhosts* 文件中）无法使用时，他们就会感到很困惑。这是因为这些地方需要通过地址反向查找规范名，如 *wormhole.movie.edu*。

因为我们对于别名 *wh249.movie.edu* 和 *wh253.movie.edu* 使用的是地址（A）记录，你可能会问："是否可以在所有情况下都使用地址记录而不用 CNAME 记录呢？"对于大部分应用程序来说，使用地址记录而不是 CNAME 记录不会有什么麻烦，因为它们只想找到 IP 地址。但是有一个应用程序 *sendmail*（译注2）就不是这样的。*sendmail* 通常用规范名替换邮件首部中所使用的别名；只有当邮件首部中的名字有相关的 CNAME 数据时才进行这种规范化。如果你的别名没有对应的 CNAME 记录，你的 *sendmail* 系统就必须知道你的主机所有可能为外界所知的别名，这就要求你对 *sendmail* 系统进行一些额外的配置。

除了 *sendmail* 的问题之外，用户还困惑为什么要在 *.rhosts* 文件使用规范名。正常情况下，查找一个有 CNAME 数据的名字（CNAME 记录）会找到规范名，然而查找有地址数据的名字（地址记录）时却不会。在这种情况下，用户应该通过查找 IP 地址来得到规范名，就像 *rlogind* 那样，不过这样的用户不该在我们管理的系统中出现。

译注 2： 互联网上最有名而且使用最广泛的电子邮件系统，可参看 O'Reilly 的《sendmail》一书，本书影印版已由中国电力出版社出版。

PTR 记录

下面我们要创建地址到名字的映射。*db.192.249.49*这个文件将网络192.249.249/24里的地址映射到主机名。这种映射使用的DNS资源记录是PTR（pointer 指针）记录。网络上每个网络接口都有一个这样的记录。（回忆一下在DNS中像查找名字那样查找地址。地址被反转过来，然后在后面加上 *in-addr.arpa*。）

下面是我们为网络 192.249.249/24 添加的 PTR 记录：

```
1.249.249.192.in-addr.arpa.  IN PTR wormhole.movie.edu.
2.249.249.192.in-addr.arpa.  IN PTR robocop.movie.edu.
3.249.249.192.in-addr.arpa.  IN PTR terminator.movie.edu.
4.249.249.192.in-addr.arpa.  IN PTR diehard.movie.edu.
```

关于这些数据有好几处需要注意的地方。首先，地址只能指向一个名字：规范名。因此192.249.249.1映射到 *wormhole.movie.edu*而不是 *wh249.movie.edu*。你可以创建两个 PTR 记录，一个指向 *wormhole.movie.edu*，一个指向 *wh249.movie.edu*，不过大部分系统都不想看到一个地址对应多个名字。其次，即使 *wormhole.movie.edu* 有两个地址，在这里你也只能看到其中一个。因为这个文件只显示与网络 192.249.249/24 的直接连接，在这里 *wormhole.movie.edu* 只有一个这样的连接。

对于网络 192.253.253/24 我们也创建了类似的数据。

完整的区数据文件

到目前为止，我们已解释了区数据文件中的各种资源记录，现在我们将让你了解一下它们都在一起时看起来是怎么样的。再重申一下，这些资源记录的实际顺序并不重要。

下面是文件 *db.movie.edu* 的内容：

```
 $TTL 3h
movie.edu. IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                              1         ; 序列号
                              3h        ; 3小时后刷新
                              1h        ; 1小时后重试
                              1w        ; 1周后期满
                              1h )      ; 否定缓存TTL为1小时
;
; 名字服务器
```

```
;
movie.edu.  IN NS  terminator.movie.edu.
movie.edu.  IN NS  wormhole.movie.edu.

;
; 对应规范名字的地址
;
localhost.movie.edu.  IN A     127.0.0.1
robocop.movie.edu.    IN A     192.249.249.2
terminator.movie.edu. IN A     192.249.249.3
diehard.movie.edu.    IN A     192.249.249.4
misery.movie.edu.     IN A     192.253.253.2
shining.movie.edu.    IN A     192.253.253.3
carrie.movie.edu.     IN A     192.253.253.4
wormhole.movie.edu.   IN A     192.249.249.1
wormhole.movie.edu.   IN A     192.253.253.1

;
; 别名
;
bigt.movie.edu.       IN CNAME terminator.movie.edu.
dh.movie.edu.         IN CNAME diehard.movie.edu.
wh.movie.edu.         IN CNAME wormhole.movie.edu.

;
; 接口专用的名字
;
wh249.movie.edu.      IN A     192.249.249.1
wh253.movie.edu.      IN A     192.253.253.1
```

下面是文件 *db.192.249.249* 的内容：

```
$TTL 3h
249.249.192.in-addr.arpa. IN SOA terminator.movie.edu. al.robocop.movie.edu.(
                          1        ; 序列号
                          3h       ; 3小时后刷新
                          1h       ; 1小时后重试
                          1w       ; 1周后期满
                          1h )     ; 否定缓存TTL为1小时

;
; 名字服务器
;
249.249.192.in-addr.arpa.  IN NS  terminator.movie.edu.
249.249.192.in-addr.arpa.  IN NS  wormhole.movie.edu.

;
; 指向规范名字的地址
;
1.249.249.192.in-addr.arpa.  IN PTR wormhole.movie.edu.
2.249.249.192.in-addr.arpa.  IN PTR robocop.movie.edu.
```

```
3.249.249.192.in-addr.arpa.  IN PTR terminator.movie.edu.
4.249.249.192.in-addr.arpa.  IN PTR diehard.movie.edu.
```

而下面是文件 *db.192.253.253* 的内容：

```
$TTL 3h
253.253.192.in-addr.arpa. IN SOA terminator.movie.edu. al.robocop.movie.edu.(
                          1         ; 序列号
                          3h        ; 3小时后刷新
                          1h        ; 1小时后重试
                          1w        ; 1周后期满
                          1h )      ; 否定缓存TTL为1小时

;
; 名字服务器
;
253.253.192.in-addr.arpa.  IN NS  terminator.movie.edu.
253.253.192.in-addr.arpa.  IN NS  wormhole.movie.edu.

;
; 指向规范名字的地址
;
1.253.253.192.in-addr.arpa.  IN PTR wormhole.movie.edu.
2.253.253.192.in-addr.arpa.  IN PTR misery.movie.edu.
3.253.253.192.in-addr.arpa.  IN PTR shining.movie.edu.
4.253.253.192.in-addr.arpa.  IN PTR carrie.movie.edu.
```

回送地址

名字服务器还需要一个额外的有关回送网络（loopback network）的*db.ADDR*文件，回送地址（loopback address）是一个特殊的地址，主机用它来将数据流导向自己。回送网络（通常）是 127.0.0/24，而主机号（通常）是 127.0.0.1。因此这个文件名就是 *db.127.0.0*。毫无疑问，它看起来和其他 *db.ADDR* 文件一样。

下面是文件 *db.127.0.0* 的内容：

```
$TTL 3h
0.0.127.in-addr.arpa. IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                           1         ; 序列号
                           3h        ; 3小时后刷新
                           1h        ; 1小时后重试
                           1w        ; 1周后期满
                           1h )      ; 否定缓存TTL为1小时

0.0.127.in-addr.arpa.  IN NS  terminator.movie.edu.
0.0.127.in-addr.arpa.  IN NS  wormhole.movie.edu.
```

```
1.0.0.127.in-addr.arpa.  IN PTR localhost.
```

为什么名字服务器需要这个小文件呢？让我们先来想一下。没有人负责网络127.0.0/24，但是系统却用它来作为回送地址。既然没有人直接负责，那么使用它的每个人都要自己负责管理。你可能会忽略这个文件，而你的名字服务器仍能继续工作。但是如果查找127.0.0.1就会失败，因为所联系的根名字服务器本身并没有配置为将127.0.0.1映射到某个名字。你应该自己提供这个映射，那么就没有问题了。

根线索（root hint）数据

除了你的本地信息以外，名字服务器还需要知道负责根区的名字服务器在何处。这个信息只能从Internet主机*ftp.rs.internic.net*（198.41.0.6）那里查到。使用匿名FTP从上面的*domain*子目录里下载*named.root*这个文件。（*named.root*就是我们前面称为*db.cache*的那个文件。在你下载后把它改名为*db.cache*就可以了。）

```
;     本文件保存了根名字服务器的信息，这些信息将被用来初始化 Internet 域名服务器
;     的缓存（例如，在 BIND 域名服务器的配置文件的语句 "cache   .   <file>" 中引用本
;     文件）。
;
;     本文件是由 InterNIC 注册服务提供的，
;     可通过匿名 FTP 从下述地址获得：
;             文件            /domain/named.root
;             服务器          FTP.RS.INTERNIC.NET
;        或者，通过 Gopher 在 RS.INTERNIC.NET
;             菜单            InterNIC Registration Services (NSI)
;                 子菜单      InterNIC Registration Archives
;             文件            named.root
;
;     上次更新：     Aug 22, 1997
;     根区的相关版本号：   1997082200
;
;
; 从前的 NS.INTERNIC.NET
;
.                        3600000  IN  NS    A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET.      3600000      A     198.41.0.4
;
; 从前的 NS1.ISI.EDU
;
.                        3600000      NS    B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET.      3600000      A     128.9.0.107
;
; 从前的 C.PSI.NET
;
.                        3600000      NS    C.ROOT-SERVERS.NET.
```

```
C.ROOT-SERVERS.NET.      3600000      A     192.33.4.12
;
; 从前的TERP.UMD.EDU
;
.                        3600000      NS    D.ROOT-SERVERS.NET.
D.ROOT-SERVERS.NET.      3600000      A     128.8.10.90
;
; 从前的NS.NASA.GOV
;
.                        3600000      NS    E.ROOT-SERVERS.NET.
E.ROOT-SERVERS.NET.      3600000      A     192.203.230.10
;
; 从前的NS.ISC.ORG
;
.                        3600000      NS    F.ROOT-SERVERS.NET.
F.ROOT-SERVERS.NET.      3600000      A     192.5.5.241
;
; 从前的NS.NIC.DDN.MIL
;
.                        3600000      NS    G.ROOT-SERVERS.NET.
G.ROOT-SERVERS.NET.      3600000      A     192.112.36.4
;
; 从前的AOS.ARL.ARMY.MIL
;
.                        3600000      NS    H.ROOT-SERVERS.NET.
H.ROOT-SERVERS.NET.      3600000      A     128.63.2.53
;
; 从前的NIC.NORDU.NET
;
.                        3600000      NS    I.ROOT-SERVERS.NET.
I.ROOT-SERVERS.NET.      3600000      A     192.36.148.17
;
; 暂时位于NSI (InterNIC)
;
.                        3600000      NS    J.ROOT-SERVERS.NET.
J.ROOT-SERVERS.NET.      3600000      A     198.41.0.10
;
; 位于LINX, 由RIPE NCC管理
;
.                        3600000      NS    K.ROOT-SERVERS.NET.
K.ROOT-SERVERS.NET.      3600000      A     193.0.14.129
;
; 暂时位于ISI (IANA)
;
.                        3600000      NS    L.ROOT-SERVERS.NET.
L.ROOT-SERVERS.NET.      3600000      A     198.32.64.12
;
; 位于日本, 由WIDE管理
;
.                        3600000      NS    M.ROOT-SERVERS.NET.
M.ROOT-SERVERS.NET.      3600000      A     202.12.27.33
; 文件结束
```

域名“．”指向根区。因为根区的名字服务器总是会随着时间而改变，不要以为上述列表总是正确的。去下载一个新的 *named.root* 文件吧！

这个文件是怎么保持最新的呢？作为网络管理员，这是你的责任。一些老版本的 BIND 会定期更新这个文件。但是这个特性被取消了：很明显，它并没有作者所希望的那样好。有时候，更改后的*db.cache*文件会发送到*bind-users*或者*namedroppers* 邮件列表。如果你订阅了其中某个列表，你就会听说这些变动。

能不能在这个文件中放一些除了根名字服务器数据以外的数据呢？可以，但是这没有用。起初，名字服务器在它的缓存中安装这些数据。虽然“ cache file ”这个名字没有变，但是这个文件的用法（慢慢地）改变了。名字服务器把该文件中的数据保存在内存中一个特殊的地方，作为根线索。即使它们的 TTL 减少到 0 也不会像对待缓存数据那样丢弃它们。名字服务器使用这些线索数据去查询根名字服务器以得到当前的根名字服务器列表，然后把它保存在缓存里。当在缓存中的根名字服务器列表超时了，名字服务器就使用这些线索再得到一份新的列表。

当名字服务器已经有一个根名字服务器列表时，为什么还要向根线索文件中的名字服务器查询这个列表呢 —— 也许它本身就是一个根名字服务器？这是因为虽然这个文件可能已经过时了，但几乎可以肯定的是，其中某些名字服务器是知道当前根名字服务器列表的。

3600000代表什么？它是这个文件中每个记录的显式生存期。在该文件的老版本中，这个数字是99999999。因为这个文件的内容最初是放在缓存中的，名字服务器需要知道在多长时间内这些数据是能用的。99999999 秒意味着非常长的一个时间 —— 根名字服务器的数据在服务器运行期间都有效。而现在因为名字服务器把这些数据保存在特殊的地方，即使超时了也不会丢弃，TTL 就不必要了。但是设置时间为 3600000 秒也没有什么坏处，同时这也是一个你可以告诉给下一个名字服务器管理员的有趣的 BIND 典故。

建立 BIND 配置文件

现在区数据文件都已经创建完毕了，还要指示名字服务器去读取这些文件。对于 BIND 来说，指示服务器读取其区数据文件的机制就是配置文件。到目前为止，我

们已经讨论了那些数据和格式遵循DNS规范的文件。不过配置文件是针对BIND的，在 DNS RFC 中并没有定义。

从版本 4 到版本 8，BIND配置文件的语法变化非常大。谢天谢地，BIND 8 和 BIND 9 之间没有任何变化。我们先讲 BIND 4 的语法，然后再讲与之对应的 BIND 8 和 9 的语法。此外，你还需要查看 *named*（注 1）的联机手册以找到你所要用到的。如果你已经有了一个 BIND 4 的配置文件，可以通过运行 *named-bootconf* 程序把它转换成版本 8 或 9 的配置文件，这个程序是随 BIND 源码一起发布的。在 BIND 8 中，这个程序在 *src/bin/named-bootconf* 下；BIND 9 中则在 *contrib/named-bootconf* 下。

在 BIND 4 中，配置文件中的注释和区数据文件中的一样 —— 以分号开始，到行尾结束：

```
; 这是注释
```

在 BIND 8 和 9 中，你可以使用三种风格的注释：C 语言风格、C ＋＋风格和 shell 风格：

```
/* 这是 C 语言风格的注释 */
// 这是 C++ 风格的注释
# 这是 shell 风格的注释
```

不要在 BIND 8 或 9 配置文件中使用 BIND 4 的注释方式，那是没有用的。分号是用来结束一条配置语句而不是表明一行注释的开始。

一般来说，配置文件包括了一行用来说明区数据文件所在的目录。在读取区数据文件之前，名字服务器把当前目录转到该目录去。这就允许指定的文件名是相对于当前目录的，而不用写完整的路径名。下面就是 BIND 4 的目录行：

```
directory /var/named
```

这里是 BIND 8 或 9 的目录行：

```
options {
```

注 1： *named* 的发音是“ name-dee ”，表示“ name server daemon ”。BIND 的发音是与“ kind ”的发音押韵的。一些具有创造力的人注意到这些名字很相似，他们故意错读成“ bin-dee ”和“ named ”（类似“ tamed ”）。

```
        directory "/var/named";
        // 在这里放置额外的选项
};
```

注意：在配置文件中只能有一条*options*语句，所以本书中后面提到的任何其他可选项都必须和*directory*选项加在一起。

在一个主服务器上，每个要读取的区数据文件在配置文件中都对应有一行。对于BIND 4来说，这一行有三个字段：*primary*（要从第一列开始）、区的域名和文件名：

```
primary  movie.edu                db.movie.edu
primary  249.249.192.in-addr.arpa db.192.249.249
primary  253.253.192.in-addr.arpa db.192.253.253
primary  0.0.127.in-addr.arpa     db.127.0.0
```

对BIND 8或9而言，行以关键字*zone*开始，后面是域名和类（*in*代表Internet）。*master*类型则和BIND 4中的*primary*一样。最后一个字段是文件名：

```
zone "movie.edu" in {
      type master;
      file "db.movie";
};
```

在本章前面我们提到过如果在资源记录中省略了类字段，名字服务器就根据配置文件来确定类。*zone*语句中的*in*将类设定为Internet类。*in*还是BIND 8或9中*zone*语句的默认值，所以对类为Internet的区，完全可以不要这个字段。因为BIND 4语法没有地方指明一个区的类，所以BIND 4的默认值也是*in*。

下面是BIND 4配置文件中读取根线索文件的行：

```
cache  .  db.cache
```

下面是BIND 8或9配置文件中与之等价的行（注2）：

```
zone "." in {
```

注2：实际上BIND 9有一个内置的线索区，所以在*named.conf*文件中不需要线索区的*zone*语句。不过有一句也没有什么关系，而在配置文件中看不到*zone*语句总有些别扭，所以我们总是写一句。

```
        type hint;
        file "db.cache";
};
```

如前所述，这个文件不是用于一般的缓存数据的。它只含有根名字服务器线索。

默认地，BIND 4 希望配置文件名为 */etc/named.boot*，但是也可以用命令行选项来更改。BIND 8 和 9 希望配置文件名为 */etc/named.conf*，而不是 */etc/named.boot*。我们的例子中的区数据文件在目录 */var/named* 中。实际上使用哪个目录并不重要。不过如果根文件系统空间不是很充裕，那么就不要把这个目录放在根文件系统中，而且要确保在名字服务器启动之前该目录所在的文件系统已经被安装（mounted）。下面是一个完整的 BIND 4 */etc/named.boot* 文件：

```
; BIND 配置文件

directory /var/named

primary  movie.edu                db.movie.edu
primary  249.249.192.in-addr.arpa db.192.249.249
primary  253.253.192.in-addr.arpa db.192.253.253
primary  0.0.127.in-addr.arpa     db.127.0.0
cache    .                        db.cache
```

下面是完整的 BIND 8 或 9 的 */etc/named.conf* 文件：

```
// BIND 配置文件

options {
        directory "/var/named";
        // 在这里放置额外的选项
};

zone "movie.edu" in {
        type master;
        file "db.movie.edu";
};

zone "249.249.192.in-addr.arpa" in {
        type master;
        file "db.192.249.249";
};

zone "253.253.192.in-addr.arpa" in {
        type master;
        file "db.192.253.253";
};
```

```
zone "0.0.127.in-addr.arpa" in {
        type master;
        file "db.127.0.0";
};

zone "." in {
        type hint;
        file "db.cache";
};
```

缩写

到目前为止，我们已经创建了主名字服务器所需要的所有文件。现在让我们回过头来重新看一遍这些区数据文件，其中有一些缩写我们还没有使用。除非你首先阅读并且理解了那些完整的写法，否则这些缩写看起来就会像密码一样。现在你已经知道了这些完整的写法，也看了 BIND 配置文件，下面我们就会告诉你一些缩写。

附加域名

primary 指令（BIND 4）或者 *zone* 语句（BIND 8 和 9）的第二个字段指定了一个域名。这个域名对最常使用的缩写来说是关键。这个域名是区数据文件中所有数据的起点（origin）。这个起点会被附加到所有区数据文件中不以“ . ”结尾的名字后面。这个起点在每个区数据文件中都可能不同，因为每个文件描述的是不同的区。

由于起点会被附加到名字后面，那么就没有必要在文件 *db.movie.edu* 中如下输入 *robocop.movie.edu* 的地址：

```
robocop.movie.edu.    IN A     192.249.249.2
```

我们可以这样输入：

```
robocop    IN A     192.249.249.2
```

在文件 *db.192.249.249* 中我们输入：

```
2.249.249.192.in-addr.arpa.  IN PTR robocop.movie.edu.
```

既然 *249.249.192.in-addr.arpa* 是起点，我们可以输入：

```
2  IN PTR robocop.movie.edu.
```

还记得前面我们提醒过你不要在使用全称域名时忘了结尾的“ . ”吗？假设你忘记了结尾的 “ . ”。一个看起来像这样的条目：

```
robocop.movie.edu    IN A     192.249.249.2
```

就会变成 *robocop.movie.edu.movie.edu*，而这根本不是你的本意。

符号 @

如果一个域名和起点相同，那么这个名字就可以写成“ @ ”。这在区数据文件的SOA记录中最常见。可以如下输入 SOA 记录：

```
@ IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                              1         ; 序列号
                              3h        ; 3 小时后刷新
                              1h        ; 1 小时后重试
                              1w        ; 1 周后期满
                              1h )      ; 否定缓存 TTL 为 1 小时
```

重复最后一个名字

如果一个资源记录名（从第一列开始）是一个空格或者制表符，那么就沿用上一个记录的名字。如果一个名字有多个资源记录，你就可以使用这个方法。下面这个例子就是一个名字有两个地址记录：

```
wormhole   IN A     192.249.249.1
           IN A     192.253.253.1
```

在第二个地址记录中，暗示*wormhole*就是其对应的名字。即使对于不同类型的资源记录，也可以使用这种捷径。

缩写后的区数据文件

现在我们已经告诉你了一些缩写，我们将使用这些缩写来再写一遍区数据文件。

下面是文件 *db.movie.edu* 的内容：

```
$TTL 3h
;
; Origin added to names not ending
; in a dot: movie.edu
; 起点：movie.edu 将被自动添加到那些没有以点结尾的名字后
;

@ IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                          1         ; 序列号
                          3h        ; 3 小时后刷新
                          1h        ; 1 小时后重试
                          1w        ; 1 周后期满
                          1h )      ; 否定缓存 TTL 为 1 小时

;
; 名字服务器（默认为名字 '@'）
;
          IN NS  terminator.movie.edu.
          IN NS  wormhole.movie.edu.

;
; 规范名字的地址
;
localhost  IN A     127.0.0.1
robocop    IN A     192.249.249.2
terminator IN A     192.249.249.3
diehard    IN A     192.249.249.4
misery     IN A     192.253.253.2
shining    IN A     192.253.253.3
carrie     IN A     192.253.253.4

wormhole   IN A     192.249.249.1
           IN A     192.253.253.1

;
; 别名
;
bigt       IN CNAME terminator
dh         IN CNAME diehard
wh         IN CNAME wormhole

;
; 接口专用的名字
;
wh249      IN A     192.249.249.1
wh253      IN A     192.253.253.1
```

下面是文件 *db.192.249.249* 的内容：

```
$TTL 3h
;
```

```
; 起点：249.249.192.in-addr.arpa 将自动被添加到那些没有以点结尾的名字后
;

@ IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                          1         ; 序列号
                          3h        ; 3 小时后刷新
                          1h        ; 1 小时后重试
                          1w        ; 1 周后期满
                          1h )      ; 否定缓存 TTL 为 1 小时

;
; 名字服务器（默认为名字 '@'）
;
   IN NS  terminator.movie.edu.
   IN NS  wormhole.movie.edu.

;
；指向规范名字的地址
;
1  IN PTR wormhole.movie.edu.
2  IN PTR robocop.movie.edu.
3  IN PTR terminator.movie.edu.
4  IN PTR diehard.movie.edu.
```

下面是文件 *db.192.253.253* 的内容：

```
$TTL 3h
;
; 起点：253.253.192.in-addr.arpa 将被自动添加到那些没有以点结尾的名字后
;

@ IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                          1         ; 序列号
                          3h        ; 3 小时后刷新
                          1h        ; 1 小时后重试
                          1w        ; 1 周后期满
                          1h )      ; 否定缓存 TTL 为 1 小时

;
; 名字服务器（默认为名字 '@'）
;
   IN NS  terminator.movie.edu.
   IN NS  wormhole.movie.edu.

;
; 指向规范名字的地址
;
1  IN PTR wormhole.movie.edu.
2  IN PTR misery.movie.edu.
3  IN PTR shining.movie.edu.
4  IN PTR carrie.movie.edu.
```

下面是文件 *db.127.0.0* 的内容：

```
$TTL 3h
@ IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                            1         ; 序列号
                            3h        ; 3小时后刷新
                            1h        ; 1小时后重试
                            1w        ; 1周后期满
                            1h )      ; 否定缓存TTL为1小时

   IN NS  terminator.movie.edu.
   IN NS  wormhole.movie.edu.

1  IN PTR localhost.
```

看看新的 *db.movie.edu* 文件，你可能注意到了，我们已经从 SOA 和 NS 记录中删除了 *movie.edu*，如下所示：

```
@ IN SOA terminator al.robocop (
                            1         ; 序列号
                            3h        ; 3小时后刷新
                            1h        ; 1小时后重试
                            1w        ; 1周后期满
                            1h )       ; 否定缓存TTL为1小时
  IN NS  terminator
  IN NS  wormhole
```

在其他区数据文件中你不可以这样做，因为起点不一样。在文件 *db.movie.edu* 中，我们写的全都是全称域名，这样 NS 和 SOA 记录对于所有区数据文件来说就完全一样了。

主机名检查（BIND 4.9.4 及后续版本）

如果你的名字服务器比 BIND 4.9.4 还要老，或者是 BIND 9 到 9.1.0（注 3），那么直接跳到下一节吧。

如果你的名字服务器是BIND 4.9.4或者更新的版本, 你必须加倍留心你的主机是如何命名的。从 4.9.4 版开始，BIND 开始检查主机名是否遵循 RFC 952。如果一个主机名不遵循 RFC 952，BIND 会认为这是个语法错误。

注 3： 从 BIND 9 到 BIND 9.1.0 都没有实现名字检查，但是在今后的 BIND 9 版本中可能会实现，所以你仍然应该阅读本小节。

不过别慌，这种检查只适用于被认为是主机名的名字。记住，资源记录有一个名字字段和一个数据字段，例如：

```
<name>      <class>  <type>  <data>
terminator  IN       A       192.249.249.3
```

主机名出现在 A（地址）和 MX（在第五章中提及）记录的名字字段中。主机名也出现在 SOA 和 NS 记录的数据字段中。CNAME 不一定非要遵循主机命名规则，因为它可以指向非主机名的名字。

让我们来看看主机命名规则。每个标识中允许包含字母和数字。下面都是有效的主机名：

```
ID4            IN A 192.249.249.10
postmanring2x  IN A 192.249.249.11
```

主机名中间出现连字符“ - ”也是允许的：

```
fx-gateway     IN A 192.249.249.12
```

警告：主机名中不允许使用下划线“ - ”。

除主机名以外的名字可以使用任何可印刷的 ASCII 字符。

如果资源记录的数据字段需要一个邮件地址（就像在 SOA 记录中一样），第一个标识可以包括任何可印刷的字符，因为它并不是一个主机名，但是其余的标识必须遵循上面描述的主机名语法。例如，一个邮件地址使用下面的语法：

```
<ASCII-characters>.<hostname-characters>
```

比如说，如果你的邮件地址是*key_grip@movie.edu*，在 SOA 记录中你仍然可以使用它，即使其中含有下划线。记住，在邮件地址中你要把第一个“ . ”换成“ @ ”，就像这样：

```
movie.edu. IN SOA terminator.movie.edu. key_grip.movie.edu. (
                              1         ; 序列号
                              3h        ; 3 小时后刷新
                              1h        ; 1 小时后重试
                              1w        ; 1 周后期满
```

```
                                1h )      ; 否定缓存TTL为1小时
```

从语法比较自由的BIND版本升级到比较保守的版本时，这一层额外的检查可能会导致出现很大的问题，特别是那些含有下划线的主机名。如果你打算以后再更改这些名字（你还是会改变的，对吧？），这个特性可以被调整为警告信息而不是错误信息，或者很简单地将不合法的名字都忽略掉。下面BIND 4的配置文件语句把产生的错误变成警告信息：

```
check-names primary warn
```

下面是对应的BIND 8或BIND 9的配置行：

```
options {
        check-names master warn;
};
```

我们会简单地解释一下这些被*syslog*记录下来的警告信息。下面是BIND 4中用来忽略这些错误信息的配置文件语句：

```
check-names primary ignore
```

下面是等价的BIND8或BIND 9语句：

```
options {
        check-names master ignore;
};
```

如果这些不遵循规定的名字来自于一个你所备份的区（而你无权管理），那么就在配置文件中加上一条相似的配置语句，不过使用*secondary*而不是*primary*：

```
check-names secondary ignore
```

对于BIND 8或9而言，使用的是*slave*而不是*secondary*：

```
options {
        check-names slave ignore;
};
```

如果这样的名字来自于对某个查询的回答而不是区数据传送，就用下面的语句：

```
check-names response ignore
```

对于 BIND 8：

```
options {
        check-names response ignore;
};
```

这里是 BIND 4.9.4 的默认设置：

```
check-names primary fail
check-names secondary warn
check-names response ignore
```

这里是 BIND 8 的默认设置：

```
options {
        check-names master fail;
        check-names slave warn;
        check-names response ignore;
};
```

对于 BIND 8 而言，这样的名字检查可以限定于以区为基础，这样就可以替换（override）某个区的 *options* 语句中的名字检查行为：

```
zone "movie.edu" in {
        type master;
        file "db.movie.edu";
        check-names fail;
};
```

注意：*options* 语句有三个字段（*check-name master fail*），但是 zone 语句只有两个字段（*check-names fail*）。这是因为 *zone* 语句已经指定了上下文（在 *zone* 语句中指定的区）。

工具

如果有一个工具能够把你的主机表转换成主文件格式（master file format）是不是很方便？确实是有这种东西，用 Perl 写的 *h2n* —— 一个主机表到主文件的转换器。你可以先用 *h2n* 来创建你的区数据文件，然后再手工维护你的数据。或者，你可以反复使用 *h2n*。正如你看到的那样，主机表的格式确实比主文件格式更容易理解和正确修改。所以你可以维护 */etc/hosts* 文件，然后每次修改后再运行 *h2n* 来更新你的区数据文件。

如果你打算使用 *h2n*，最好一开始就用它，因为它使用的是 */etc/hosts*，而不是你手工编制的区数据来产生新的区数据文件。我们可以像下面这样，通过使用 *h2n* 来生成这一章中的示例区数据文件，从而省去许多功夫：

```
% h2n -d movie.edu -s terminator -s robocop \
  -n 192.249.249 -n 192.253.253 \
  -u al.robocop.movie.edu
```

（要生成 BIND 8 或 9 的配置文件，使用 *-v 8* 选项。）

-d 和 *-n* 选项指定的是正向映射区的域名和网络号。你会注意到，那些区数据文件的名字就是由这些选项而来的。选项 *-s* 列出NS记录中使用的区的权威名字服务器。*-u*（user）是 SOA 记录中的电子邮件地址。在讨论了 DNS 是如何影响电子邮件的之后，我们将在第七章中更详细地讨论 *h2n*。

运行主名字服务器

现在你已经创建了你的区数据文件，也就已经准备好可以开始运行几个名字服务器了。你将需要建立两个名字服务器：一个主名字服务器和一个辅名字服务器。不过，在启动一个名字服务器之前，必须确认 *syslog* daemon（守护进程）已经在运行了。如果名字服务器读配置文件和区数据文件时遇到错误，就会把这个信息送到 *syslog* daemon。如果这个错误很严重，名字服务器就会退出。

启动名字服务器

在这里，我们假定你管理的机器上已经有 BIND 名字服务器和已安装好的支持工具 *nslookup*。检查一下 *named* 联机手册，看看名字服务器可执行程序所在的目录，并且确认该可执行程序也在你的系统里。在BSD系统中，名字服务器从 */etc* 开始运行，但是也可能在 */usr/sbin*。其他可以查找 *named* 的地方有 */usr/etc/in.named* 和 */usr/sbin/in.named*。以下描述是假设名字服务器在 */usr/sbin* 中。

你必须成为超级用户（root）才能启动名字服务器。名字服务器在一个保留端口上监听（listen）查询，所以需要超级用户的特权。你第一次运行名字服务器时，从命令行启动它，检查它是否工作正常。然后，我们会教你如何在系统启动时自动运行名字服务器。

下面的命令将启动名字服务器。我们在主机*terminator.movie.edu*上运行下列命令：

```
#  /usr/sbin/named
```

这个命令假设你的配置文件是*/etc/named.boot* (BIND 4) 或*/etc/named.conf*（BIND 8或9）。你可以把你的配置文件放在别的地方，但是必须使用*-c*命令行选项来告诉名字服务器它在哪里：

```
#  /usr/sbin/named -c conf-file
```

检查 Syslog 错误

启动你的名字服务器后的第一件事情就是检查*syslog*文件中的错误信息。如果你对于*syslog*还不熟悉，可以查看*syslog.conf*联机手册中关于*syslog*配置文件的说明，或者查看*syslogd*联机手册中关于*syslog* daemon的说明。名字服务器以*named*的名义利用*daemon*把这些信息记录到日志中。你可以通过在*/etc/syslog.conf*中查找*daemon*来找出*syslog*的信息记录在哪里：

```
%  grep daemon /etc/syslog.conf
*.err;kern.debug;daemon,auth.notice /var/adm/messages
```

在这台主机上，名字服务器的*syslog*信息被记录在*/var/adm/messages*中，而且*syslog*只保存级别为LOG_NOTICE或更高的信息。一些有用的信息是以LOG_INFO级别被发送到*syslog*的，而你可能想看看这些信息。第七章中我们将更加详细地介绍*syslog*信息，在读完之后，你就能够决定是否要改变记录日志信息的级别（log level）了。

当名字服务器启动时，它会记录一条启动信息：

```
%  grep named /var/adm/messages
Jan 10 20:48:32 terminator named[3221]: starting.
```

这条启动信息不是错误信息，但是与之一起的其他信息可能是错误信息。（如果你的服务器使用*restarted*而不是*starting*，那也没有问题。这条信息是在BIND 4.9.3中改变的。）最常见的错误是区数据文件或者配置文件中的语法错误。例如，如果你在地址记录中忘了写资源记录类型：

```
robocop  IN  192.249.249.2
```

那你就会看到下面的 *syslog* 错误信息：

```
Jan 10 20:48:32 terminator named[3221]: Line 24: Unknown type:
              192.249.249.2
Jan 10 20:48:32 terminator named[3221]: db.movie.edu Line 24:
              Database error near (192.249.249.2)
Jan 10 20:48:32 terminator named[3221]: master zone "movie.edu" (IN) rejected due
to errors (serial 1)
```

或者如果你在 */etc/named.conf* 中把字“ zone ”写错了：

```
zne "movie.edu" in {
```

你将会看到下面的错误信息：

```
Mar 22 20:14:21 terminator named[1477]: /etc/named.conf:10:
                syntax error near `zne'
```

如果 BIND 4.9.4 及后续版本发现一个不符合 RFC 952 的名字，你就会看到下面这样的 *syslog* 错误信息：

```
Jul 24 20:56:26 terminator named[1496]: owner name "ID_4.movie.edu IN"
                                (primary) is invalid - rejecting
Jul 24 20:56:26 terminator named[1496]: db.movie.edu:33: owner name error
Jul 24 20:56:26 terminator named[1496]: db.movie.edu:33: Database error near (A)
Jul 24 20:56:26 terminator named[1496]: master zone "movie.edu" (IN) rejected due
to errors (serial 1)
```

如果有一个语法错误，就检查一下 *syslog* 错误信息中指明的行号，看看你是否能找出错误所在。你已经知道区数据文件大概看起来是个什么样子了，应该能够找出大部分简单的语法错误了。否则你就必须仔细读一读附录一，看看所有资源记录的详细语法。如果你能够改正这样的语法错误，就改正它，然后用 *ndc*（name daemon controller）重新加载名字服务器：

```
#  ndc reload
```

它将重新读取区数据文件（注 4）。在第七章中你会看到更多有关用 *ndc* 来控制名字服务器的内容。

注 4： 对 BIND 9 名字服务器，要使用 *rndc*，不过我们还没有告诉你如何配置它。如果想知道如何配置 *rndc*，直接跳到第七章。不过 *rndc* 不需要太多配置也能工作。

用 nslookup 测试你的设置

如果已经正确地建立了你的本地区，并且连接到了Internet，你就应该能查找本地和远程的域名了。现在我们将通过*nslookup*把这些查找一步一步地呈现给你。本书有专门的一整章（第十二章）是讲这个话题的，在这里我们仍将详细地谈一谈*nslookup*，因为将会用它做一些基本的名字服务器检测。

设置本地域名

开始运行*nslookup*前你需要设置主机的本地域名。这样你就可以直接查找*carrie*这样的名字了，而没有必要输入*carrie.movie.edu*，因为系统会帮你加上域名*movie.edu*。

有两种方法设置本地域名：*hostname(1)*或者*/etc/resolv.conf*。有些人会说实际上大部分情况中都是在*/etc/resolv.conf*中设置本地域名，但是你可以随便用那一种。在这本书中我们假定本地域名是来自于*hostname(1)*的。

创建称为*/etc/resolv.conf*的文件，其中包括下面这样一行，要从第一列开始（用你的本地域名替换掉*movie.edu*）：

```
domain movie.edu
```

或者将*hostname(1)*设置为一个域名。在主机*terminator*上，我们把*hostname(1)*设置为*terminator.movie.edu*。不要在这个名字结尾加上“.”。

查找一个本地域名

*nslookup*能用来查找任何类型的资源记录，也能用来向任意的名字服务器发送查询请求。默认地，它用*resolv.conf*中指定的第一个名字服务器来查找A（地址）记录。（如果*resolv.conf*中没有指定名字服务器，解析器就默认地向本地名字服务器提出查询。）要用*nslookup*来查找某个主机的地址，只需运行*nslookup*，以主机的域名作为惟一的参数。查找本地域名几乎立刻就能返回结果。

我们运行*nslookup*来查找*carrie*：

```
%   nslookup carrie
Server: terminator.movie.edu
```

```
Address: 192.249.249.3

Name:    carrie.movie.edu
Address: 192.253.253.4
```

如果能够成功查找一个本地域名, 就说明对你的正向映射区而言你的本地名字服务器的配置是正确的。如果查找失败，你将看到的如下所示：

```
*** terminator.movie.edu can't find carrie: Non-existent domain
```

这意味着要么你的区数据中没有 *carrie*，这时要检查一下你的区数据文件；要么是你没有在*hostname(1)*中设置你的本地域名；要么是出现了一些名字服务器错误（但是你应该能够通过查看 *syslog* 消息找到这些错误）。

查找一个本地地址

当给了 *nslookup* 一个地址让它去查找时，它知道要进行 PTR 查询而不是地址查询。我们运行 *nslookup* 来根据 *carrie* 的地址进行查找：

```
%   nslookup 192.253.253.4
Server: terminator.movie.edu
Address: 192.249.249.3

Name:    carrie.movie.edu
Address: 192.253.253.4
```

如果查找地址成功的话，你的本地名字服务器对 *in-addr.arpa*（反向映射）区的配置就是正确的。如果查找失败了，你会看到同你查找名字时一样的错误信息。

查找一个远程域名

下一步就是用本地名字服务器来查找远程域名, 比如*ftp.uu.net*或是其他一些你所知道的 Internet 上的系统。这个命令可不会像刚才那个反应得那么快。如果 *nslookup* 无法从你的名字服务器得到响应，在放弃之前，它将等待的时间会超过 1 分钟：

```
%   nslookup ftp.uu.net.
Server: terminator.movie.edu
Address: 192.249.249.3

Name:      ftp.uu.net
Addresses: 192.48.96.9
```

如果这也能顺利进行，就说明你的名字服务器知道根名字服务器的位置，并且知道如何与它们联系，找到你自己区以外的其他域名的信息。如果失败了，要么是因为你忘了配置根线索文件（*syslog*日志中会显示一条信息），或者网络在某个地方坏掉了，使你无法访问要查找的远程区的名字服务器。换个远程域名试试。

如果前面这些查找都成功了，恭喜你！你已经建立起了一个主名字服务器，并且也已经运行起来了。此时你就可以开始配置你的辅名字服务器了。

再来一次测试

虽然你已经检测过了，不过还是再来一次吧。试着让某个远程名字服务器来查找你的区中的域名。只有当你的父区的名字服务器已经将你的区授权给你刚建立起来的名字服务器后，这样的查找才会成功。如果你的父区要求你必须运行起两个名字服务器才能将你的区授权给你，直接跳到下一节。

要想使*nslookup*用一个远程名字服务器来查询你的本地域名，就要把本地主机的域名作为第一个参数，把远程名字服务器的域名作为第二个参数。这次如果失败，在*nslookup*给出一个出错信息之前，也会等待比一分钟还要长的时间。例如，让*gatekeeper.dec.com*来查找*carrie.movie.edu*：

```
%   nslookup carrie gatekeeper.dec.com.
Server: gatekeeper.dec.com.
Address: 204.123.2.2

Name:    carrie.movie.edu
Address: 192.253.253.4
```

如果前两个查找都成功了，但是用远程名字服务器查找本地名字失败，可能是因为你的区还没有向你的父名字服务器注册。在刚开始，这可能不是什么问题，因为你区中的主机能查找你的区内、区外的其他主机的域名。你还能发电子邮件并FTP到本地或远程系统。不过对有的系统而言，如果不能将你的主机地址映射回域名，就不允许你进行FTP连接。但是不注册很快就会有问题了。你的区之外的主机不能查找你区内的域名，那么你可以给远程区中的朋友发邮件，但是你却收不到他们的回信。要解决这个问题，就要与负责你父区的管理员联系，让他们帮你检查一下你的区的授权。

编辑启动文件

一旦确定了你的名字服务器能正常工作，并且从此就能使用它们了，你就需要在你系统的启动文件中把它配置成自动启动，以及将 *hostname(1)* 设置为一个域名。查看一下你的系统供应商是否早已经设置：在系统启动时名字服务器也会启动。你也许要从启动行中删除那些注释字符，否则启动文件可能会测试 */etc/named.conf* 或 */etc/named.boot* 是否存在。要查找自动启动行，可以使用：

```
%  grep named /etc/*rc*
```

或者，如果你使用 System V 风格的 *rc* 文件，就用：

```
%  grep named /etc/rc.d/*/S*
```

如果你什么都没找到，那么在相应的启动文件中，将如下的语句添加到用 *ifconfig* 初始化你的网络接口的语句后面：

```
if test -x /etc/named -a -f /etc/named.conf
then
        echo "Starting named"
        /etc/named
fi
```

你可能想在默认路由安装以后或路由守护进程（*routed* 或 *gated*）启动以后，看这些服务是否需要名字服务器或使用 */etc/hosts* 就可以了，再启动名字服务器。

找到初始化主机名的启动文件。把 *hostname(1)* 改成一个域名。例如我们把：

```
hostname terminator
```

改成：

```
hostname terminator.movie.edu
```

运行辅名字服务器

你需要再建立一个名字服务器以增强 DNS 系统的健壮性。你可以（可能最终也会）为你的区设置多于两个的权威名字服务器。两个名字服务器是最低的要求了 —— 如果你只有一个名字服务器而它又停机了，那么就没有人能查找域名了。第二个名字

服务器能分担第一个名字服务器的负荷，或者在第一个名字服务器停机时承担全部的负荷。你还可以再建立一个主服务器，不过我们并不推荐这么做。我们建议你另建一个辅名字服务器。如果以后你觉得你能应付管理多个主名字服务器所造成的额外工作时，你还是能够把辅名字服务器变成主名字服务器的。

名字服务器是怎么知道它是某个区的主名字服务器还是辅名字服务器呢？*named.conf* 文件会告诉名字服务器它是一个区的主名字服务器还是辅名字服务器。NS记录并没有告诉我们哪个服务器是区的主名字服务器，而哪些又是辅名字服务器——它只说明这些名字服务器都是谁。（总地来说，DNS并不关心这两者的区别，只要实际名字解析正常，辅服务器和主服务器都一样。）

那么主名字服务器和辅名字服务器又有什么区别呢？最关键的区别在于它们是从哪里获取数据。主名字服务器是从区数据文件中读取数据的，而辅名字服务器是通过网络从其他的名字服务器装载数据的。这个过程称为“区传送”（zone transfer）。

辅名字服务器并不一定非要从主名字服务器装载区数据，它还可以从别的辅服务器装载。

使用辅名字服务器的一大优点就是对一个区来说你只用维护一套区数据文件，也就是主名字服务器上的文件。你不用担心各个名字服务器之间的同步问题，辅名字服务器会为你做这些事情。要告诫你的是辅名字服务器不会实时同步——它通过“轮询”（poll）方式来了解它的数据是否是最新的。这个轮询间隔（polling interval）就是SOA记录中一个我们还没有说明过的数字。（BIND 8和9支持一种机制，能加速区数据的分发，我们待会儿再谈这个。）

辅名字服务器不需要从网络上获得所有的区数据文件，开销文件（overhead files）*db.cache* 和 *db.127.0.0* 与主名字服务器上的一样，所以在辅名字服务器上保存了这些文件的一份副本。这意味着辅名字服务器也是*0.0.127.in-addr.arpa*的主名字服务器。当然你也可以把它设置为*0.0.127.in-addr.arpa*的辅名字服务器，只是区数据没有变化——它同样也可以是主名字服务器。

建立

要建立你的辅名字服务器，先要在辅名字服务器主机上为区数据文件创建一个目录

（比如，*/var/named*），然后再将 */etc/named.conf*、*db.cache* 和 *db.127.0.0* 这三个文件拷贝过来：

```
#   rcp /etc/named.conf   host:/etc
#  rcp db.cache db.127.0.0  host:db-file-directory
```

你必须修改辅名字服务器主机上的 */etc/named.conf* 文件。对于 BIND 4 来说，除了 *0.0.127.in-addr.arpa* 之外，要把所有出现的 *primary* 改成 *secondary*。在这些行中的文件名前加上你刚刚建立好的主名字服务器的 IP 地址。例如，如果原来的 BIND 4 配置文件有一行是这样的：

```
primary  movie.edu      db.movie.edu
```

那么将它修改成这样：

```
secondary  movie.edu      192.249.249.3 db.movie.edu
```

如果原来的 BIND 8 或 9 的配置文件有一行是：

```
zone "movie.edu" in {
      type master;
      file "db.movie.edu";
};
```

那么把 *master* 改为 *slave*，然后添加一个带主服务器 IP 地址的 *masters* 行：

```
zone "movie.edu" in {
      type slave;
      file "bak.movie.edu";
      masters { 192.249.249.3; };
};
```

这就告诉名字服务器它是区 *movie.edu* 的辅名字服务器，并且它应该要跟踪名字服务器 195.249.249.3 上保存的该区的版本。辅名字服务器将会在本地文件 *bak.movie.edu* 中保留该区的一个备份。

对于电影大学来说，我们在主机 *wormhole.movie.edu* 上建立了辅名字服务器。回忆一下 *terminator.movie.edu*（主名字服务器）上的配置文件，是这样的：

```
directory /var/named

primary  movie.edu                db.movie.edu
```

```
primary  249.249.192.in-addr.arpa db.192.249.249
primary  253.253.192.in-addr.arpa db.192.253.253
primary  0.0.127.in-addr.arpa     db.127.0.0
cache    .                        db.cache
```

我们把 */etc/named.conf*、*db.cache* 和 *db.127.0.0* 拷贝到 *wormhole.movie.edu* 上，然后像上面说的那样编辑这些配置文件。现在，*wormhole.movie.edu* 上的 BIND 4 配置文件如下所示：

```
directory /var/named

secondary  movie.edu                192.249.249.3 bak.movie.edu
secondary  249.249.192.in-addr.arpa 192.249.249.3 bak.192.249.249
secondary  253.253.192.in-addr.arpa 192.249.249.3 bak.192.253.253
primary    0.0.127.in-addr.arpa     db.127.0.0
cache      .                        db.cache
```

与之等价的 BIND 8 或 9 的配置文件则是这样的：

```
options {
        directory "/var/named";
};

zone "movie.edu" in {
        type slave;
        file "bak.movie.edu";
        masters { 192.249.249.3; };
};

zone "249.249.192.in-addr.arpa" in {
        type slave;
        file "bak.192.249.249";
        masters { 192.249.249.3; };
};

zone "253.253.192.in-addr.arpa" in {
        type slave;
        file "bak.192.253.253";
        masters { 192.249.249.3; };
};

zone "0.0.127.in-addr.arpa" in {
        type master;
        file "db.127.0.0";
};

zone "." in {
        type hint;
        file "db.cache";
```

```
};
```

这会使 *wormhole.movie.edu* 上的名字服务器通过网络从 192.249.249.3（*terminator.movie.edu*）那里加载 *movie.edu*、*249.249.192.in-addr.arpa* 和 *253.253.192.in-addr.arpa*。它也会在*/var/named*中保存这些文件的一份拷贝。你可能会发现把这些备份文件隔离在一个子目录中会很方便。我们在命名时加了一个*bak*这样独特的前缀，因为在极少见的情况下，我们不得不手工删除所有的备份文件。能够一眼就认出它们是备份的区数据文件也是有帮助的，因为这样我们就不会误删除它们了。在后面我们将更多地讨论到备份文件。

现在启动辅名字服务器。就像启动主名字服务器时那样检查*syslog*文件中的错误信息。和在主名字服务器上一样，启动辅名字服务器的命令也是：

```
#    /usr/sbin/named
```

在辅名字服务器上你还要做一项检查，看看名字服务器是否创建了这些备份文件，而在主名字服务器上是不用这样做的。启动了我们在*wormhole.movie.edu*上的名字服务器后不久，就能在*/var/named*目录中看到*bak.movie.edu*、*bak.192. 249.249*和*bak.192.253.253*了。这就意味着 slave 名字服务器已经成功地从主名字服务器那里加载了区，并且保存了一份备份拷贝。

要完全建立起你的辅名字服务器，就要试着也查找一下刚才主服务器启动后查找过的那些域名。这一次*nslookup*必须运行在辅名字服务器所运行的主机上，以保证查询的就是辅名字服务器。如果你的辅名字服务器工作得很好的话，也在你的系统启动文件中添加相应的行，使得当系统启动时就会自动运行辅名字服务器，并且将*hostname(1)*设置成一个域名。

备份文件

辅名字服务器并没有被要求保存一份区数据文件的备份。如果有备份拷贝的话，辅服务器在启动时读取它，过后再检查主服务器看那里是否已经有了更新的拷贝，而不是立即从主名字服务器那里加载新的区的拷贝。如果主名字服务器有了更新的拷贝，辅名字服务器就会取来并且保存在备份文件中。

为什么要保存一份备份的拷贝呢？设想一下在辅名字服务器启动时主名字服务器停

机了，那么在主名字服务器启动之前，辅名字服务器就无法进行区传送，也就无法作为这个区的名字服务器而进行工作了。有这份备份的拷贝，辅名字服务器就有了区数据，虽然可能有点过时。因为辅名字服务器并不依赖于主名字服务器总是要运行着，所以这个设置更健壮。

如果不打算要这些备份文件的话，就在 BIND 4 配置文件中的 *secondary* 那一行删掉最后的文件名。在 BIND 8 或 9 中，删除 *file* 这一行。不过，我们建议你将所有的辅名字服务器都配置成保存备份文件。保存备份区数据文件的额外代价非常小，而如果你非常需要备份文件时却没有，代价就太高了。

SOA 值

还记得 SOA 记录吗？

```
movie.edu. IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                              1         ; 序列号
                              3h        ; 3 小时后刷新
                              1h        ; 1 小时后重试
                              1w        ; 1 周后期满
                              1h )      ; 否定缓存 TTL 为 1 小时
```

我们还没有解释过这些圆括号间的值都是用来干嘛的。

序列号（serial number）作用于这个区中所有的数据。我们选择 1 作为我们的初始序列号码是从逻辑上的意义出发的，但是很多人发现如果用日期作为序列号更有用处，例如 1997102301。这个格式是 YYYYMMDDNN，YYYY 代表年份，MM 代表月份，DD 代表日子，而 NN 代表是这一天中第几次修改。这个域只有在这种排序（order）下才起作用，其他任何排序都不行，因为除此之外没有任何排序能总是随着时间的推移而增加。无论你选择哪种方式，在每次更新了你的区数据后不要忘了增加序列号的值是很重要的。

当辅名字服务器与主名字服务器联系，希望得到区数据时，它首先会请求相关数据的序列号。如果辅名字服务器上该区的序列号比主名字服务器的小，那么辅名字服务器的区数据就过时了。这时辅名字服务器就从主名字服务器那里加载一份新的区数据拷贝。如果辅名字服务器启动时没有可读取的备份文件，它总是会从主服务器那里加载区数据。正如你所猜想的那样，当你修改了主名字服务器的区数据文件后，你必须添加序列号。我们将在第七章中讨论更新区数据文件的问题。

下面四个字段规定了各种时间间隔，默认是以秒为单位：

refresh（刷新）

刷新间隔告诉某个区的辅名字服务器相隔多久检查该区的数据是否是最新的。需要注意的是，这个功能会使系统增加负载，辅名字服务器将每隔一段刷新间隔就对每个区进行一次 SOA 查询。我们所使用的值是 3 小时，看起来有点激进。在等待他们的新工作站能够使用时，大多数用户能够容忍半个工作日的延迟，让区数据这样的东西在网上广播。如果你的服务器仅白天提供服务，可考虑将这个值增加为 8 个小时。如果你的区数据并不经常变化，或者你的辅名字服务器分布很广（就像根名字服务器那样），可以考虑将这个值设得更长一点，比如说 24 小时。

retry（重试）

如果辅名字服务器超过刷新间隔时间后无法访问主名字服务器（可能停机了），那么它就开始每隔一段重试间隔就试着连接一次。重试间隔通常比刷新间隔短，但也不一定非要这样。

expire（过期或期满）

如果在期满时间内辅名字服务器还是无法与主名字服务器联系上，辅名字服务器就使这个区失效。使一个区失效就意味着辅名字服务器将停止关于该区的回答，因为这些区数据太旧了，没有用了。实质上，这个字段是在说：有些时候，数据太旧了，不给数据也比给过期的坏数据要好。将期满时间以周为单位是比较常见的 —— 如果你总是很难访问到你的更新数据源，可以设置得更长一点（1 个月）。期满时间总是比重试和刷新间隔要长得多；如果期满时间比刷新间隔还要小的话，你的辅名字服务器将会在试着加载新数据之前就使该区失效。

否定缓存 TTL（生存期）

TTL 就是生存期（time to live）的意思。这个值对来自这个区的权威名字服务器的否定响应都适用。

注意：对于 BIND 8.2 以前的 BIND，SOA 记录中最后一个字段既是该区默认的生存期，也是否定缓存生存期。

你们中有人可能读过本书前几版，会注意到SOA记录的数字字段中所使用的格式有

所变化。BIND 曾经对上述四个字段只能理解以秒为单位的值。(因此，整整一代管理员都知道一周有 608400 秒。）现在除了最早的 BIND 名字服务器（BIND 4.8.3)，其他所有的 BIND 都允许你在 TTL 控制语句的参数中为这四个字段指定其他的单位，就像本章前面所讲的那样。例如，你可以用 *3h*、*180m*、甚至 *2h60m* 来指定 3 小时的刷新间隔。还可以用 *d* 代表天，用 *w* 代表周。

你的 SOA 记录使用什么样的值完全取决于你的需要。通常，时间越长，名字服务器的负担就越轻，而变动传播的时间就越长；时间越短，名字服务器的负担就越重，而变动传播的时间就越短。本书中我们使用的值对大多数系统来说都是适用的。RFC 1537 建议顶级名字服务器采用下列值：

```
刷新            24 小时
重试             2 小时
期满            30 天
默认 TTL         4 天
```

还有一个实现特性你该注意一下。老版本（4.8.3 以前）的 BIND 辅名字服务器在区加载时停止回答查询。因此，BIND 改成了分散区加载，以减少不可用时间。所以即使你设置了一个很小的刷新间隔，你的辅名字服务器可能也不会像你要求的那样频繁检查。BIND 会尝试着一次加载一定数量的区数据，然后等上 15 分钟，再处理下一批。

现在我们已经告诉了你所有有关辅名字服务器是如何通过轮询来保证它的数据是最新的，不过 BIND 8 和 9 改变了区数据传播的方式！轮询特性还在，但是 BIND 8 和 9 增加了当区的数据改变后进行通知的功能。如果你的主名字服务器和辅名字服务器都是 BIND 8 或 9 的，那么主名字服务器会在区数据改变 15 分钟之内通知辅名字服务器 :该加载该区的一份新的拷贝了。这个通知会缩短辅名字服务器的刷新间隔，试着立即加载区。我们会在第十章中详细讨论这个问题。

多个主服务器

还有其他方法能使你的辅名字服务器配置变得更加健壮吗？有，你可以配置最多十个主名字服务器的 IP 地址。在 BIND 4 的配置文件中，只需在第一个 IP 地址后、备份文件名之前依次添加就可以了。在 BIND 8 或 9 的配置文件中，在第一个 IP 地址后面依次添加并且用分号分隔：

```
masters { 192.249.249.3; 192.249.249.4; };
```

辅名字服务器会按照列出的顺序依次尝试每个IP地址对应的主服务器，一直到收到回答为止。在BIND 8.1.2上，如果第一个响应的主服务器的序列号比较大，那么辅名字服务器总是从它那里进行区传送。只有在前面的主名字服务器都没有响应的情况下，辅名字服务器才会查询下一个主服务器。不过从BIND 8.2以后，辅名字服务器实际上会查询列出的所有主名字服务器，从具有最高序列号的服务器那里传送区数据。如果有好几个有主服务器具有相同的最高序列号，辅名字服务器就从其中列在最前面的那个服务器那里传送区。

这个特性的本意在于，如果运行该区的主名字服务器的主机是多宿主的话，你就能列出它所有的IP地址。不过，因为对于所联系的主机是主名字服务器还是辅名字服务器并没有做任何的检查，所以你还可以列出所有运行该区辅服务器的主机的IP地址，如果这对你的设置有意义的话。这样一来，如果第一个主服务器停机或不可到达的话，你的辅名字服务器就可以从别的主名字服务器那里传送区。

增加更多的区

现在你的名字服务器已经运行起来了，你可能想支持更多的区。需要做些什么呢？确实没有什么特别的。你所要做的只是在配置文件中增加更多的 *primary* 或者 *secondary* 语句（BIND 4）或 *zone* 语句（BIND 8和9）。你甚至可以在你的主服务器上增加 *secondary* 语句，在辅服务器上增加 *primary* 语句。（你可能早已经注意到了，你的辅名字服务器是 *0.0.127.in-addr-arpa* 的主名字服务器。）

到目前为止，重复一下我们在本书前面所讲的某些东西是很有用的。把某个名字服务器称为主名字服务器或辅名字服务器有点傻。名字服务器可以是 —— 通常也是多个区的权威。一个名字服务器可以是某个区的主名字服务器，同时又是另一个区的辅名字服务器。不过，大部分名字服务器要么是它们加载的大部分区的主名字服务器，要么是它们加载的大部分区的辅名字服务器。所以如果我们把某个名字服务器称为主名字服务器或是辅名字服务器，我们指的是对它所加载的大部分区而言它是主名字服务器或是辅名字服务器。

接下来是什么？

在这一章中，我们告诉你了如何通过把*/etc/hosts*转换为等价的名字服务器数据来创建名字服务器区数据文件，还有如何建立主名字服务器和辅名字服务器。不过，要完成你的本地区的设置还有更多工作要做：需要为电子邮件系统修改你的区数据，还要配置你区中其他的主机来使用你的名字服务器，可能还要启动更多的名字服务器。这些主题都将在下面几章中谈到。

第五章

DNS 和电子邮件

本章内容：

- MX 记录
- 邮件交换器到底是什么?
- MX 算法

到这个时候，爱丽丝开始觉得相当困乏了，它继续近乎梦呓地自言自语着：“猫吃蝙蝠吗？猫吃蝙蝠吗？”有时又成了“蝙蝠吃猫吗？”你看，反正她什么问题都答不上来，怎么说也就无所谓了。

我打赌，在看过了那么长的第四章之后，你已经昏昏欲睡了。谢天谢地，作为系统管理员和邮件管理员的你应该对这一章所要讲的东西非常感兴趣：DNS是如何影响电子邮件的。而且即使你对此并不感兴趣，它至少也要比上一章短得多。

DNS与主机表相比的一个优势就在于它支持高级邮件路由。当邮件收发器（mailer）只能用*HOSTS.TXT*（以及它的派生：*/etc/hosts*）来工作时，所能做的最多也就是试着把邮件直接发送到主机的IP地址。如果失败，要么就延迟发送邮件，过一会儿再重试，要么就将邮件退回给发送者。

DNS提供一种机制，能为邮件的发送指定备份主机，它还允许一台主机为别的主机承担邮件处理任务。例如，这就使得没有运行邮件收发器的无盘主机能让它的服务器为它处理发来的邮件。

同主机表不同，DNS允许用任意名字来代表电子邮件目的地。你可以（Internet上大多数机构也是这样做的）把你的主（main）正向映射区的域名作为电子邮件目的地。或者你也可以给你的区添加一些域名，它们只作为电子邮件目的地，而不代表

任何主机。一个逻辑上的电子邮件目的地也可以代表几个邮件服务器。如果用主机表的话，邮件目的地必须是主机，而不能是其他的什么了。

综上所述，在管理员配置网络上的电子邮件时，这些特性给予他们更多的灵活性。

MX 记录

DNS 用一种资源记录类型来实现增强的邮件路由，那就是 MX 记录。MX 记录的功能最早被分成两个记录来实现，MD（mail destination，邮件目的地）和 MF（mail forwarder，邮件转发器）记录。MD 指定了某个发送到给定域名的邮件的最终目的地。MF 指定了一个当最终的目的地不可到达时，用来继续转发邮件的主机。

DNS 在 ARPAnet 上的早期经验显示将这些功能分开来实现效果并不太好。邮件收发器要想决定将邮件发往何处，需要把 MD 和 MF 都连到同一个域名上（如果它们都存在的话）——单独使用其中任意一个都不能使邮件正常地被发送。但是若只显式地查找一种或另一种类型（MD 或 MF），名字服务器将只会把该种记录类型放到缓存中。所以邮件收发器要么就得做两次查询，一个是查询 MD 记录，一个是查询 MF 记录；要么就不能再使用缓存中的回答。这就意味着管理邮件的开销比管理其他服务的开销要大得多，而这最终被认为是不可接收的。

将这两个记录结合成一个记录类型 :MX, 就解决了这个问题。现在邮件收发器只需要某一特定域名（目的地）的所有 MX 记录就能做出邮件路由决定了。只要 TTL 选择得合适，使用缓存中的 MX 记录也很好。

MX 记录为一个域名指定了一个邮件交换器（mail exchanger），它是一台主机，负责处理或转发该域名的邮件（比如，通过一个防火墙）。“ 处理 ” 邮件是指将邮件递送到它写给的个人，或是通过网关送到其他的邮件传送装置，如 X.400 或 Microsoft Exchange。“ 转发 ” 是指将邮件送到最终的目的地，或是通过 Internet 的 SMTP（Simple Mail Transfer Protocol，简单邮件传输协议）送到另一个离目的地更 “ 近 ” 一点的邮件交换器。有时转发邮件也需要让邮件排队等待一段时间。

为了避免邮件路由循环，在 MX 记录中除了邮件交换器的域名之外，另外还有一个参数：优先级值。优先级值是一个无符号的 16 位整数（在 0 和 65535 之间），用来指示邮件交换器的优先级。例如，MX 记录：

```
peets.mpk.ca.us.    IN    MX    10 relay.hp.com.
```

指定 *relay.hp.com* 是 *peets.mpk.ca.us* 的邮件交换器，它的优先级值是 10。

总地来说，一个目的地的所有邮件交换器的优先级值决定了邮件收发器使用它们的先后次序。优先级值本身是多少并不重要，它与其他邮件交换器的优先级值的相对关系才是重要的：它是比该目的地的其他邮件交换器的值高还是低？除非还涉及到其他的记录：

```
plange.puntacana.dr.  IN  MX  1 listo.puntacana.dr.
plange.puntacana.dr.  IN  MX  2 hep.puntacana.dr.
```

与下面这两个记录是完全一样的：

```
plange.puntacana.dr.  IN  MX  50  listo.puntacana.dr.
plange.puntacana.dr.  IN  MX  100 hep.puntacana.dr.
```

邮件收发器总是最先试着向优先级值最小的邮件交换器发送邮件。刚开始的时候，这看上去似乎有点违反直觉：优先级最高的邮件交换器的优先级值最小。不过这是因为优先级值是一个无符号数，你可以将“最好的”邮件交换器的优先级值设为 0。

如果向优先级最高的邮件交换器发送失败，邮件收发器会试着向优先级低一点的邮件交换器（它们的优先级值要大一点）发送。也就是说，邮件收发器会试着先向优先级高的邮件交换器发送，再试着向优先级低的邮件交换器发送。多个邮件交换器可以有相同的优先级值。邮件收发器可以自己选择先向哪个发送（注 1）。邮件收发器必须在试着向所有具有某个给定优先级值的邮件交换器发送之后，才能试着向优先级值更大一级的邮件交换器发送。

例如，*oreilly.com* 的 MX 记录是这样的：

```
oreilly.com.    IN    MX    0  ora.oreilly.com.
oreilly.com.    IN    MX    10 ruby.oreilly.com.
oreilly.com.    IN    MX    10 opal.oreilly.com.
```

总地来说，这些 MX 记录会指示邮件收发器，如果要向 *oreilly.com* 发送邮件，将试着以这样的顺序进行：

注 1： 最新版本的 *sendmail*，版本 8，会在具有相同优先级值的邮件交换器之间随机选择。

1. 最先向 *ora.oreilly.com*
2. 再向 *ruby.oreilly.com* 或 *opal.oreilly.com*
3. 剩下的一个优先级值为 10 的邮件交换器（步骤 2 中没用的那一个）

当然，一旦邮件收发器成功地把邮件发送到 *oreilly.com* 的任意一个邮件交换器，它就停止了。如果邮件收发器成功地把写往 *oreilly.com* 的邮件发送到 *ora.oreilly.com*，它就不需要再试 *ruby.oreilly.com* 或 *opal.oreilly.com* 了。

注意，*oreilly.com* 并不是某台主机，它是 O'Reilly 公司的主正向映射区的域名。O'Reilly 公司用这个域名作为每个在那里工作的人的电子邮件目的地。对于联系人来说，记住单一的电子邮件目的地 *oreilly.com* 比记住有各个雇员电子邮件账号的主机 —— *ruby.oreilly.com?* 或 *amber.oreilly.com?* 容易得多。

当然这需要 *ora.oreilly.com* 上的邮件收发器清楚 O'Reilly 公司的每个用户的电子邮件账号都在哪台主机上。这通常是通过在 *ora.oreilly.com* 上维护一个主别名文件（master aliases file）来完成的，它会将电子邮件从 *ora.oreilly.com* 转发给其最终的目的地。

如果一个目的地没有 MX 记录但有一个或多个 A 记录该怎么办呢？邮件收发器会简单地不把邮件发送给目的地吗？实际上，你可以把最近的 *sendmail* 版本编译成这样工作。不过大多数厂商把它们的 *sendmail* 编译得更宽容一些：如果没有 MX 记录存在，但有一个或多个 A 记录，那么它们至少会试着向这些地址传送。非常规编译的版本 8 的 *sendmail* 会试图向没有 MX 记录的邮件目的地的地址发送。如果你不太清楚你的邮件服务器是否会向只有地址记录的目的地发送邮件，那就查一下你厂商的说明。虽然几乎所有的邮件收发器都会向只有地址记录而没有 MX 记录的目的地发送邮件，但是最好还是使每个合法的邮件地址都有至少一个 MX 记录。当 *sendmail* 有邮件要发送时，它总是会先查找目的地的 MX 记录。如果目的地没有 MX 记录，名字服务器 —— 通常是你的一个权威名字服务器 —— 仍然会回答这个查询，然后 *sendmail* 会继续查找 A 记录。这需要额外的时间，减缓邮件的传送，还会增加你的区的权威名字服务器的负载。如果你能为每个目的地都添加一个 MX 记录，它指向的域名映射到的地址同地址查询时返回的地址相同，那么 *sendmail* 就只用发送惟一的一次查询，然后邮件收发器的本地名字服务器会将 MX 记录放在缓存中以供今后之用。

邮件交换器到底是什么?

对你们中的大多数人而言，邮件交换器可能还是一个新概念，那就让我们再详细地谈谈它吧。在这里有一个简单的比喻可能会有助于你理解：把邮件交换器想像成一个一个的机场，这样当你的亲戚们要坐飞机来看你时，你可以建议他们在哪个机场降落，就如同建立 MX 记录来指示邮件收发器往哪里发送邮件一样。

假设你住在加州的Los Gatos, 你的亲戚们要飞来看你, 离你最近的机场是San Jose，第二近的是 San Francisco，而第三近的是 Oakland。（我们将忽略机票价格，交通状况等因素。）看出它们的相似之处了吗？如下描述它：

```
los-gatos.ca.us.    IN    MX    1 san-jose.ca.us.
los-gatos.ca.us.    IN    MX    2 san-francisco.ca.us.
los-gatos.ca.us.    IN    MX    3 oakland.ca.us.
```

这个 MX 列表就是按照发送目的地的顺序排列的，它告诉邮件收发器（你的亲戚们），要到达某个给定的电子邮件目的地（你的家），要把消息发送到（飞）哪里。优先级值是告诉它们发送到这个目的地的合适度 —— 你可以把它想像成到最终目的地（无论你选择的什么地方）的逻辑距离，或者是把它看成是按照这些邮件交换器到最终目的地的远近程度排序的“ top 10 ”式的排行榜。

根据这个列表，你可以说:“ 先试着飞到 San Jose 去，如果你无法到达那里的话，再按照那个顺序，试一下 San Francisco 和 Oakland。” 它还说明，如果你到了 San Francisco，你应该再飞到 San Jose 去。如果你先到了 Oakland，那么你应试着再飞到 San Jose，或者至少再飞到 San Francisco。

那么怎么样才是一个好的邮件交换器呢？好比具备什么样的条件才会是一个好的机场一样。

大小

> 你不会为了到Los Gatos而选择到Reid-Hillview小型机场的航班，因为这个机场不具备处理大型飞机和许多人的能力。（让一架大型喷气式飞机降落在 280 号州际公路上，也比降落在 Reid-Hillview 机场要好。）同样地，你也不会想用一个瘦弱的、动力不足的主机来作为邮件交换器的，它无法承受这样的负载。

正常工作时间

你很清楚在冬天途经 Denver 国际机场是什么样的，对吗？那你也该很清楚用一台经常不能正常工作、很难访问到的主机作为邮件交换器会是什么样的。

连接

如果你的亲戚是从很远的地方飞来的，你应该确信他们能直接飞到你给他们所列出的表中的至少一个机场。如果他们从 Helsinki 飞来，你就不能他们，他们只能选择 San Jose 和 Oakland。同样地，你也要保证，任何可能给你发邮件的人都能访问你主机的邮件交换器中的至少一个。

管理

当降落或途经时，机场的管理好坏将关系到你的安全，而且还关系到使用是否方便。选择邮件交换器时，请考虑一下这些因素。你邮件的保密性、普通操作的发送速度，以及当你关机时邮件做何处理，这些都取决于管理你的邮件交换器的管理员。

记住机场这个比喻，我们待会儿还要用到的。

MX 算法

刚才讲的都是关于MX记录和邮件交换器的基本概念，不过你还应该了解更多一点。为了避免邮件路由回路，当邮件收发器决定向哪儿发送邮件时，还需要使用一些比我们刚才讲的稍微复杂一点的算法（注 2）。

想像一下如果邮件收发器不检查路由回路，会发生什么。让我们来假设一下，你从你的工作站给 *nuts@oreilly.com* 发邮件，顺便谈谈本书的质量。不幸的是，*ora.oreilly.com* 现在停机了。不过没问题！回想一下 *oreilly.com* 的 MX 记录：

```
oreilly.com.    IN    MX    0  ora.oreilly.com.
oreilly.com.    IN    MX    10 ruby.oreilly.com.
oreilly.com.    IN    MX    10 opal.oreilly.com.
```

你的邮件收发器退一步，把你的消息发往 *ruby.oreilly.com*，它现在正常工作。*ruby.oreilly.com* 的邮件收发器就试着将邮件转发到 *ora.oreilly.com*，但是不能够，

注 2： 这个算法是基于 RFC 974 的，它描述了 Internet 邮件路由如何工作。

因为*ora.oreilly.com*已经停机了。现在怎么办？除非*ruby.oreilly.com*检查它所做事情的合法性，否则它将试着将消息转发给*opal.oreilly.com*，或者甚至发给它自己。当然，这对将邮件发送出去没有丝毫的帮助。如果*ruby.oreilly.com*将消息发给它自己，就产生了一个邮件路由回路。如果*ruby.oreilly.com*将消息发给*opal.oreilly.com*，而*opal.oreilly.com*可能将邮件发还给*ruby.oreilly.com*或是它自己，这样还是会产生邮件路由回路。

为了避免发生这样的情况，邮件收发器在决定向哪儿发送消息之前将放弃某些MX记录。邮件收发器将MX记录按优先级值排序，在这个列表中查找它自己所在主机的规范域名。如果本地主机就是一个邮件交换器，邮件收发器将从列表中删除代表它本身的那个MX记录，以及所有优先级值相等或更大（也就是优先级更低的邮件交换器）的那些MX记录。这就使得邮件收发器不会向它自己或是离最终目的地“更远”的邮件收发器发送消息。

让我们用机场那个比喻再来想想这个问题。这次，把你自己想像成一个飞机乘客（一个邮件），你想到达科罗拉多的Greeley。你不能直接飞到Greeley，但你能飞到Fort Collins或是Denver（两个相邻的最好的邮件交换器）。由于Fort Collins离Greeley更近一些，你选择飞到Fort Collins。

现在，一旦你已经飞到Fort Collins了，就没有理由要飞到Denver了，它离你的目的地更远（一个优先级更低的邮件交换器）。（而从Fort Collins飞到Fort Collins也同样可笑。）所以现在，能使你到达目的地的惟一可行的航班选择是从Fort Collins飞到Greeley。你要取消到优先级低一些的目的地的航班，这样才能避免那些浪费旅行时间的频繁循环飞行。

一个警告：大多数邮件收发器只在MX记录列表中查找它们本地主机的规范域名，它们不会查找别名（也就是CNAME记录中左边的名字）。除非你在MX记录中总是使用规范名字，否则不能保证邮件收发器能在MX列表中找到它自己，你还是有产生邮件回路的可能性。

如果你列出的一个邮件交换器用的是别名，而它又愚蠢到想给它自己发送邮件，它会检查到这个回路，并返回一个错误：

```
554 MX list for movie.edu points back to relay.isp.com
554 <root@movie.edu>... Local configuration error
```

在较新版本的*sendmail*中，上面这个出错信息取代了那个奇怪的出错信息“ I refuse to talk to myself ”。有一个原则：在 MX 记录中总是使用邮件交换器的规范名字。

还有一个警告：作为邮件交换器的主机必须要有地址记录。邮件收发器需要找到你命名的每一个邮件交换器的地址，否则它就不能试着向那里发送。

再回到我们*oreilly.com*的例子中，当*ruby.oreilly.com*从你的工作站收到消息时，它的邮件收发器就检查 MX 记录列表：

```
oreilly.com.    IN    MX    0  ora.oreilly.com.
oreilly.com.    IN    MX    10 ruby.oreilly.com.
oreilly.com.    IN    MX    10 opal.oreilly.com.
```

*ruby.oreilly.com*发现它自己本地主机的域名在列表中，优先级值是10，它将放弃所有优先级值等于或大于 10 的 MX 记录（下面用黑体标出的记录）：

```
oreilly.com.    IN    MX    0  ora.oreilly.com.
oreilly.com.    IN    MX    10 ruby.oreilly.com.
oreilly.com.    IN    MX    10 opal.oreilly.com.
```

只剩下：

```
oreilly.com.    IN    MX    0 ora.oreilly.com.
```

由于*ora.oreilly.com*现在停机，*ruby.oreilly.com*将暂缓发送，将消息放在队列当中。

如果邮件收发器发现它自己的优先级就是最高的（优先级值最小），会怎么样呢？会不得不放弃整个MX列表吗？有的邮件收发器会试着直接把邮件发送到目的地主机的IP地址，这是最后的尝试。而对大多数邮件收发器而言，这是个错误。这会使DNS认为邮件收发器应该为邮件目的地处理（而不是转发）邮件，但是邮件收发器并没有被配置成这样。或者它会通知管理员使用了不正确的优先级值，对 MX 记录进行了错误的排序。

比如说，管理*acme.com*的人添加了一个邮件交换器记录，指示将写给*acme.com*的邮件发到他的 ISP 的邮件收发器上：

```
acme.com.    IN    MX    10 mail.isp.net.
```

许多邮件收发器都要被配置成能识别需要它代为处理邮件的其他主机的别名和名字。

除非 *mail.isp.net* 上的邮件收发器被配置成将写给 *acme.com* 的邮件认为是自己本地的邮件，否则它会认为要求它转接（relay）这封邮件，并试着将这封邮件转发给离最终目的地更近一些的邮件交换器（注3）。当它在MX记录中查找*acme.com*时，它将发现自己就是优先级最高的邮件交换器，于是将邮件退回给发送者。并且还要返回一个我们很熟悉的出错信息：

```
554 MX list for acme.com points back to mail.isp.com
554 <root@acme.com>... Local configuration error
```

许多版本的 *sendmail* 使用 *w* 类或者 *w* 文件类（fileclass）来作为“本地”目的地列表。根据你的*sendmail.cf*文件，在其中增加一个别名，如下所示添加一行非常容易：

```
Cw acme.com
```

如果你的邮件收发器使用其他的邮件传输器（如，UUCP）来把邮件传送到以它为邮件交换器的主机，可能就需要更多的相关配置。

你可能已经注意到了，我们总是将优先级值赋成10的倍数。因为这样使你可以在两个MX记录中间插入记录，而不用改变其他记录的权值，所以很方便，不过除此之外，也就没有什么特别的了。我们也可以以1或100为增量单位，效果也是一样的。

注3： 当然，除非 *mail.isp.net* 的邮件收发器被配置成不转接未知域名的邮件。在这种情况下，它只会简单地拒绝该邮件。

本章内容：

- 解析器
- 解析器配置示例
- 把损失与不便降低到最小
- 与供应商有关的选项

第六章

配置主机

这个在河堤上举行的聚会看上去实在太奇怪了——拖着长长羽毛的鸟儿、皮毛紧紧贴在身上的动物，而且一切都湿漉漉的，显得那么乖戾而又令人感觉不舒服。

如果你或是你机构中的其他人已经为你的区建立起了名字服务器，那么你就可以配置你网络上的主机来使用这些名字服务器，主要是配置这些主机的解析器。为了使用DNS，你还应该检查一下*hosts.equiv*和*.rhosts*这样的文件，以进行必要的修改；你可能需要把这些文件中的一些主机名转换为域名。而且你可能还想添加一些别名，既为了方便用户，也将向DNS的转换所带来的影响降到最小。

本章就是讲述这些话题的，而且还将描述如何在各种常见的Unix版本以及微软的Windows 95、NT和2000中配置解析器。

解析器

在前面的第二章中，我们曾介绍过解析器，但是并没有过多地讨论它。你也许还记得，解析器是DNS客户端的一部分。它负责将应用程序对主机信息的请求翻译成对名字服务器的一个查询，同时还负责将对查询的响应翻译成对应用程序的回答。

到目前为止，我们还没有配置解析器，因为时机还不成熟。在第四章中，我们建立起自己的名字服务器，解析器默认的行为就能满足我们的要求。但是如果我们需要解析器比默认的做得更多，或者和默认的行为不一样，我们就需要配置解析器。

首先要声明一件事情：我们在下面几节中要说到的是没有其他任何命名服务的vanilla BIND 8.2.3解析器。并不是所有的解析器都是这样的；一些供应商仍然封装基于DNS代码早期版本的解析器，而有些供应商还实现了特殊的解析器功能，能允许你修改解析器算法。只要有必要，我们就会指出BIND 8.2.3解析器和其他早期解析器的不同之处，特别是和4.8.3和4.9版解析器的不同，当我们最后一次更新本书时，这些版本还在被很多的供应商使用。在本章的后面我们还会谈到各种不同供应商进行的扩展。

那么实际上解析器能允许你配置些什么呢？大多数解析器至少允许你配置解析器行为的三个方面：本地域名、搜索列表和解析器查询的名字服务器。许多Unix供应商还允许你通过DNS的非标准扩展来配置解析器的其他行为。有时这些扩展还需要同其他软件协同工作，比如Sun公司的NIS（Network Information Server，网络信息服务器）；有时它们只是供应商附加的功能（注1）。

几乎所有的解析器配置都是在*/etc/reslov.conf*这个文件中（也可能是*/usr/etc/resolv.conf*或你的主机上其他与之类似的文件，查看一下*resolver*手册，通常是在第四部分或第五部分，去查一下）。在*reslov.conf*中有五个主要的指令可供你使用：*domain*、*search*、*nameserver*、*sortlist*和*options*。这些指令控制着解析器的行为。某些版本的Unix上还有其他一些由供应商提供的指令，我们将在本章的最后讨论它们。

本地域名

本地域名是解析器所在地的域名。在大多数情况下，它是运行解析器的主机所在区的域名。例如，在主机*terminator.movie.edu*上的解析器很可能就以*movie.edu*作为它的本地域名。

注1： NIS过去被称为“Yellow Pages”或“YP”（黄页），但是由于英国电话公司对Yellow Pages这个名字拥有专利，所以后来改称为NIS。

解析器用本地域名来解释非全称域名。例如，当你在*.rhosts*文件中添加了这样一个条目：

```
relay bernie
```

那么 *relay* 就被假定是你的本地域名。这比允许 Internet 上每台主机域名以 *relay* 开头的主机上的 *bernie* 用户访问更有实际意义。其他的授权文件（比如，*hosts.equiv* 和 *hosts.lpd*）也是这样工作的。

通常，本地域名是根据主机的 *hostname*（主机名）决定的，本地域名是主机名中第一个“ . ”后面的所有字符。如果这个名字中没有“ . ”，那么就假定本地域名是根域。所以 *hostname asylum.sf.ca.us* 就意味着本地域名是 *sf.ca.us*，而 *hostnaem dogbert* 就意味着本地域名是根域 —— 考虑到几乎没有主机只用单标号的域名，所以这很有可能是个错误（注 2）。

你也可以在 *resolv.conf* 中用 *domain* 指令来设置本地域名。如果你在 *domain* 指令中指定了本地域名，那么它将会覆盖从 *hostname* 派生而来的本地域名。

domain 指令的语法非常简单，不过因为解析器不会报错，所以你必须保证使用正确。关键字 *domain* 从第一列开始，后面跟着空白字符（一个或多个空格或者制表符），然后是本地域名的名字。本地域名结尾不要有“ . ”，而要像下面这样：

```
domain colospgs.co.us
```

旧版本的 BIND 解析器（4.8.3 之前的 BIND），行尾的空白字符都是不允许的，这会导致你本地域名的名字有一个或多个空白字符，这可不是你想要的。这里还有另外一种方法来设置本地域名 —— 通过 LOCALDOMAIN 这个环境变量。LOCALDOMAIN 很方便，它使你可以根据不同的用户来设置不同的本地域名。例如，在你公司的计算中心你可能想有一个大规模并行计算机，而全球的雇员都可以访问到。每个雇员都可能在公司的不同子域中工作。通过 LOCALDOMAIN，每个雇员就可以在自己的 shell 启动文件中设置其合适的本地域名。

而你应该使用哪种方法呢？是 *hostname*、*domain* 指令，还是 LOCALDOMAIN？我们倾向于使用 *hostname*，主要是因为 Berkeley 就是这样做的，而且它需要较少的显

注 2： 实际上有一些单标号的域名，它们指向地址，例如 *cc*。

式配置，看起来更清楚一些。而且，一些Berkeley软件，特别是使用*ruserok()*库调用来认证用户的软件，只有*hostname*被设置为完整的域名后，才允许在*hosts.equiv*那样的文件中使用较短的*hostname*。

如果你运行的软件不能容忍长的*hostname*，那么你可以使用*domain*指令。*hostname*这个命令仍然会返回一个短的名字，而解析器则使用*reslov.conf*中规定的域。你甚至可以在拥有很多用户的主机上使用LOCALDOMAIN。

搜索列表

不管是从*hostname*还是从*resolv.conf*得到的本地域名也同时决定了默认的搜索列表（search list）。搜索列表是设计用来使用户减少输入量，从而使使用变得更容易。它的思路就是，在命令行输入的名字可能还不完整，也就是说，可能还不是全称域名时，就试着在一个或多个域中搜索它。

大部分使用域名作为参数的Unix网络命令，如*telnet*、*ftp*、*rlogin*和*rsh*，都将搜索列表应用于这些参数。

从BIND 4.8.3到BIND 4.9，如何产生默认搜索列表和如何应用搜索列表都有所改动。如果你的解析器是较早版本的，你仍然需要了解4.8.3的行为，但是如果你用的是较新的版本，包括BIND 8.2.3（注3）在内，你需要了解4.9版解析器所做的改进。

无论是哪种BIND解析器，用户都可以在域名结尾加一个“.”表明这是一个全称域名（注4）。

```
%  telnet ftp.ora.com.
```

注3： 虽然ISC在BIND 8中增加了大量新的服务器功能，但是它的解析器几乎还是和BIND 4.9的一样。

注4： 注意，我们说过解析器能处理结尾的“.”。有的应用程序，特别是有的Unix邮件用户代理，并不能正确处理电子邮件地址结尾的“.”。它们甚至很难将地址中的域名交给解析器来处理。例如，这个命令中结尾的“.”：

意味着，“没有必要再花力气去搜索任何其他的域了，这个域名就是全称域名。”这就好像在 Unix 和 MS DOS 文件系统中完整路径名开头的斜杠“/”。不以“/”开头的路径名都被解释成是相对于当前路径的相对路径，而以“/”开头的路径名则被认为是绝对的，是从根开始的。

BIND 4.8.3 搜索列表

使用BIND 4.8.3的解析器，默认搜索列表包括本地域名和那些有两个或更多标号的父域的域名。于是在一个运行着 4.8.3 解析器的主机上并且配置了：

```
domain cv.hp.com
```

那么默认的搜索列表就会首先包括*cv.hp.com*，它就是本地域名，然后是本地域名的父域 *hp.com*，但不包括 *com*，因为它只有一个标号（注 5）。解析器会把搜索列表中的每个元素逐个附加到名字之后进行查找，只有在输入的名字至少包括一个“.”时，最后再查找该名字本身。因此，如果一个用户输入：

```
%  telnet pronto.cv.hp.com
```

就会导致解析器先查找*pronto.cv.hp.com.cv.hp.com*和*pronto.cv.hp.com.hp.com*，然后再查找 *pronto.cv.hp.com* 本身。如果用户在这台主机上输入：

```
%  telnet asap
```

将会导致解析器查找 *asap.cv.hp.com* 和 *asap.hp.com*，而不会查找 *asap*，这是因为输入的名字 *asap* 中不包括点。

注意，一旦收到了对某个查找的域名的回答数据，就会立即停止使用搜索列表。在这个 *asap*的例子中，如果*asap.cv.hp.com*被成功解析为某个地址，那么就不会轮到把搜索列表 *hp.com* 附加在后面。

注 5： 老版本的BIND解析器不加入顶级域的一个原因就是几乎没有主机是在Internet域名空间的第二级中，所以 *foo* 后加上 *com* 或 *edu* 不太像真实主机的域名。另外，查找 *foo.com* 或*foo.edu*的地址还会向根名字服务器发出查询，这将加重根的负担，同时也消耗时间。

BIND 4.9 及后续版本的搜索列表

BIND 4.9及后续版本的解析器中，默认搜索列表只包括本地域名。所以，如果你主机的配置是：

```
domain cv.hp.com
```

那么默认的搜索列表只会包括 *cv.hp.com*。另外，相对早先版本解析器的改变之一是，只有在用户输入的名字被试着解析过之后才会开始使用搜索列表。只要你输入的参数中至少包含一个点，它就会先查找你输入的名字，然后才是查找把搜索列表中的每个元素附加到输入的名字之后生成的名字。如果第一次查找失败了，就开始使用搜索列表。甚至于如果输入的参数不包括点（就是说，是一个单标号名字），在解析器把搜索列表中的元素逐一附加到这个名字的后面查找过之后，它也会试着查找这个名字的。

为什么先试着查找用户输入的名字会比较好呢？DNS的设计者发现，根据经验，如果一个使用者不厌其烦地敲入一个哪怕其中只有一个点的名字时，他往往就是在敲一个不以点结尾的全称域名。如果使用旧的搜索列表的工作方式，解析器在试着查询用户输入的名字之前将会发送好几个没有意义的查询。

所以在使用 4.9 或更新版的解析器时，用户输入：

```
% telnet pronto.cv.hp.com
```

那么会首先查找 *pronto.cv.hp.com*（这个参数中有三个点）。如果这个查询失败了，解析器就会试着再查找 *pronto.cv.hp.com.cv.hp.com*。如果一个用户输入的是：

```
% telnet asap
```

会使解析器首先去查找 *asap.cv.hp.com*，这是因为输入的名字中没有点，然后再查找 *asap*。

search 指令

如果你不喜欢在你设置本地域名时生成的默认搜索列表怎么办？对于BIND 4.8.3和更新版本的解析器，你可以明确地自己来设置搜索列表，按照你希望的各个域被搜索的顺序一个域名一个域名地列出来。你需用 *search* 指令来实现这个功能。

*search*指令除了参数可以是多个域名以外，它的语法和*domain*指令很类似。关键字*search*从行的第一列开始，紧接着是一个空格或制表符，后面依次是一到六个域名，按照你所希望进行搜索的顺序排列（注6）。这个列表中第一个域名也被认为是本地域名，所以*search*和*domain*指令在这一点上是互斥的。如果你在*resolv.conf*文件中使用了这两个指令，那么后出现的那个指令就会覆盖前一个指令的配置。

例如，指令：

```
search corp.hp.com paloalto.hp.com hp.com
```

会告诉解析器首先搜索*corp.hp.com*域，然后是*paloalto.hp.com*，再是这两个域的父域*hp.com*。

这个指令对于那些用户需要经常访问*corp.hp.com*和*paloalto.hp.com*的主机来说是很有用的。另外，在BIND 4.8.3的解析器上，指令：

```
search corp.hp.com
```

就会导致在使用搜索列表时解析器不再搜索本地域名的父域。（在4.9或更新的解析器上，搜索列表中没有父域的名字，所以这就和默认的方式没有什么区别。）如果主机的用户只访问位于本地域中的主机，或者如果到父名字服务器的连接不是太好（因为它把对父名字服务器的不必要查询减少到最小）的话，使用这样的指令就会很有用了。

注意：如果你使用了*domain*指令，并且把你的解析器升级到BIND 4.9版或更新的版本，那些依赖于你本地域名的父域存在于搜索列表中的用户可能会认为解析器突然不能工作了。不过，你可以用*search*指令来配置你的解析器，让它使用和以前一样的搜索列表，这就像恢复了以前的工作方式似的。譬如，在BIND 4.9、BIND 8或BIND 9上，你可以用*search nsr.hp.com hp.com*来代替*domain nsr.hp.com*，实现的效果一样。

---

nameserver指令

在第四章中，我们讨论了两种类型的名字服务器：主名字服务器和辅名字服务器。

注6： 实际上，BIND 9解析器支持搜索列表中使用八个域名。

但是如果你不想在某台主机上运行名字服务器而又想用DNS的话，怎么办？或者对于这个问题而言，你无法在某台主机上运行名字服务器怎么办（例如你的操作系统不支持）？当然你不需要在每台主机上都运行名字服务器，对吧？

是的，你当然不需要这样。默认地，解析器会查找本地主机上的名字服务器，这就是为什么一旦我们配置好名字服务器就能马上在*terminator.movie.edu*和*wormhole.movie.edu* 上使用 *nslookup* 了。但是你也可以指示解析器去使用其他主机的域名服务。这种配置在“ BIND 操作指南 ”中被称为一个 DNS 客户端。

nameserver 指令（是的，两个单词合成了一个）告诉解析器要查询的名字服务器的 IP 地址。例如下面这行：

```
nameserver 15.32.17.2
```

就指示解析器把查询发送给运行在IP地址为15.32.17.2的主机上的名字服务器，而不是本地主机。也就是说，在那些没有运行名字服务器的主机上，你可以使用 *nameserver* 指令将它们指向远程名字服务器。通常，你应该把你主机上的解析器配置成查询你自己的名字服务器。

不过，因为 BIND 4.9 之前的名字服务器完全没有注意到访问控制，而许多较新服务器的管理员也没有对查询做出限制，所以你可以配置你的解析器去查询几乎任何人的名字服务器。当然，不事先得到允许就配置你的主机去使用他人的名字服务器，即使不是十分无礼，也会被认为是很专横的。而且使用你自己的名字服务器性能通常会更好，所以我们认为只能在紧急情况下才采用这种使用别人名字服务器的方法。

也可以通过使用本地主机的IP地址或零地址（zero address），将你的解析器配置成查询本机上的名字服务器。大部分 TCP/IP 实现都将零地址（即，0.0.0.0）解释成“ 本机 ”。当然主机真实的 IP 地址也意味着“ 本机 ”。在那些不认识零地址的主机上你可以使用 127.0.0.1。

那么如果你的解析器查询的名字服务器停机了怎么办？有方法能指定备份吗？难道还要回头再使用主机表？

解析器也允许你用多个 *nameserver* 指令来指定最多三个（记住，是三个）名字服务

器。如果没有收到对查询的响应或者超时了，那么解析器将按照它们排列的顺序依次向这些名字服务器请求查询。例如下面这行：

```
nameserver 15.32.17.2
nameserver 15.32.17.4
```

告诉解析器先查询在15.32.17.2的名字服务器，如果它不响应，再查询在15.32.17.4的名字服务器。注意，你配置的名字服务器的数量将影响解析器其他方面的行为。

警告：如果你使用了多个*nameserver*指令，那么一定不要使用回送地址！如果本地名字服务器没有运行，则某些由Berkeley演化而来的TCP/IP实现中的一个错误将导致与BIND有关问题的产生。在本地名字服务器没有运行时，解析器已连接的数据报套接字（datagram socket）无法重新绑定到新的本地地址上。这样，当解析器向备份的远程名字服务器继续发送查询包时，包的源地址却为127.0.0.1。因此，当远程名字服务器要发送回答时，就会把查询结果发送给它们自己。

配置了一个名字服务器

如果只配置了一个名字服务器，解析器查询它的超时设置是5秒。超时（timeout）就是解析器在进行下一次查询之前，等待名字服务器响应的时间。如果解析器遇到了一个错误，该错误表明名字服务器已经停机了或不可达到，或者如果超时了的话，解析器就会把超时时间加倍，然后再次查询该名字服务器。可能导致这种情况的错误包括（注7）：

- 收到一条*ICMP port unreachable*，这意味着名字服务器没有在名字服务器端口上进行侦听。
- 收到一条*ICMP host unreachable*或者*network unreachable*，这意味着无法将查询发送到目的IP地址。

如果域名或数据不存在，解析器将不再重试查询。至少在理论上，每个名字服务器对名字空间的了解程度都是一致的，没有理由相信这个而不相信那个。所以，如果一个名字服务器告诉你某个域名不存在，或者对应你指定域名的所查数据不存在，

注7：当我们说“配置了一个名字服务器”时，就是说*resolve.conf*中有一个*nameserver*指令，或者不使用*nameserver*指令以表明本机上就有一个名字服务器。

别的其他任何名字服务器也会给你同样的答案（注8）。如果解析器每次发送查询都收到一个网络错误（最多共四次，注9），它就会去使用主机表了。注意，这是错误而不是超时。如果它每次查询都会超时，解析器就会返回一个为空的回答而不会去使用 */etc/hosts*。

配置了多个名字服务器

当配置了多于一个名字服务器时，情况就有些不同了。会发生这样一些变化：解析器会首先查询列表中的第一个名字服务器，超时设置为5秒，就像只配置有一个名字服务器那样。如果解析器超时，或收到一个网络错误，它将转而查询下一个名字服务器，对这个名字服务器也等待同样长的超时时间。不幸的是，许多可能的错误解析器都收不到；解析器所使用的套接字是“未连接”（unconnected），因为它必须能够从它查询的任何一个名字服务器那里接收响应，而未连接的套接字是无法收到ICMP错误消息的。如果解析器查询了所有配置的名字服务器，还是没有结果，它就会更新超时时间并再次开始循环。

解析器进行下一轮查询时使用的超时时间将基于*resolv.conf*中配置的名字服务器的个数。第二轮查询所使用的超时设置就是用10去除以配置的名字服务器的个数，再取整。后一轮查询的超时时间都是前一轮的两倍。在三组重发之后（对于每一台配置的名字服务器总共是四次超时），解析器就放弃查询名字服务器了。

在BIND 8.2.1中，ISC将解析器变成了只发送一遍重试，也就是说对*resolv.conf*中的每个名字服务器都进行两次查询。这是为了减少在所有名字服务器都不响应的情况下用户等待解析器返回回答的时间。

表6-1说明了在你配置一台、两台或三台名字服务器时，超时时间会是个什么样子：

注8： DNS本身所固有的延迟使这成为了一个小小的谎言——一个区的权威主名字服务器可能与同样是该区权威的辅名字服务器有不同的数据。主名字服务器可能刚刚从磁盘上加载了新的区数据，而辅名字服务器还没有来得及从主名字服务器转移新的区数据。两个名字服务器都返回该区的权威回答，但主名字服务器可能会知道某个新的主机，而辅名字服务器却不知道。

注9： 对于BIND 8.2.1及更新的版本来说是两个。

表 6-1 BIND 4.9 到 8.2 中的解析器超时时间

	配置的名字服务器		
重试	1	2	3
0	5s	(2x) 5s	(3x) 5s
1	10s	(2x) 5s	(3x) 3s
2	20s	(2x) 10s	(3x) 6s
3	40s	(2x) 20s	(3x) 13s
总数	75s	80s	81s

表 6-2 给出了 BIND 8.2 及后续版本解析器的默认超时时间。

表 6-2 BIND 8.2.1 及后续版本中的解析器超时时间

	配置的名字服务器		
重试	1	2	3
0	5s	(2x) 5s	(3x) 5s
1	10s	(2x) 5s	(3x) 3s
总数	15s	20s	24s

所以，如果你配置了三台名字服务器，解析器查询第一台名字服务器，超时时间是 5 秒。如果超时，那么解析器就查询第二个服务器，超时时间同上，第三台也是一样。如果解析器要再从头来一遍，那么它就把超时增加到 10 秒，再除以 3（10/3 再取整就是 3 秒），然后再查询第一台名字服务器。

这些时间太长了吗？要记得这里讲的是最坏情况。如果运行名字服务器的主机速度足够快，而服务器又正常运行的话，你的解析器将在不到1秒的时间里就获得答案。只有配置的所有服务器都很忙，或是这些服务器或是你的网络出了问题，解析器才会执行所有的重发过程，最后再放弃。

解析器在放弃查询后又会做些什么呢？它会认为超时，然后返回一个错误。通常，这个出错信息是这样的：

```
%   telnet tootsie
tootsie: Host name lookup failure
```

当然，要等到这个出错信息出现，需要 75 秒左右的时间，所以要耐心一点。

sortlist 指令

sortlist 指令是 BIND 4.9 及其后续版本中解析器的一种机制，当查询收到的响应包括多个地址时，该机制允许你选择更希望使用的子网和网络。在某些情况下，你想让你的主机通过特定的网络来到达某个目的地。例如，你的工作站和你的NFS服务器都有两个网络接口：一个是在以太网上，子网是128.32.1/24；而另一个在FDDI环网上，子网是128.32.42/24。如果你让你工作站的解析器自己来决定，那么当你从服务器装载一个文件系统时，就有可能是NFS服务器的任意一个IP地址，不妨假定是名字服务器回答包中的第一个。要想确保你的解析器会首先试着用FDDI环上的接口，你可以在 *resolv.conf* 中增加一个 *sortlist* 指令，让128.32.42/24上的地址在返回给程序的结构中排在程序会首先使用的位置上：

```
sortlist 128.32.42.0/255.255.255.0
```

“/”后面的参数是问题中给定子网的子网掩码。如果想使用整个网络，你可以不要“/”和子网掩码：

```
sortlist 128.32.0.0
```

那么解析器就会假定你指的是整个网络128.32/16。（解析器从IP地址的前两位得出该网络默认的没有进行子网划分的网络掩码。）

当然，你还可以指定多个（最多十个）优先的子网和网络：

```
sortlist 128.32.42.0/255.255.255.0 15.0.0.0
```

解析器会按照这些参数在指令中出现的顺序检查回答中的每个地址，看它们是否与之相符，并且将不相符的地址放在最后面。

options 指令

options 指令是在BIND 4.9中引入的，它允许你把几个内部解析器的设置合在一起。第一个是 debug 标志：RES_DEBUG。下面这个指令：

```
options debug
```

设置RES_DEBUG，这将在标准输出上产生大量让人兴奋的调试信息，不过这首先

假设你的解析器配置里定义了DEBUG。(实际上，这个要求并不见得都能满足，因为大部分供应商编译它们的解析器时都没有定义DEBUG。)如果你尝试使用你的解析器或名字服务来分析问题时，这会是非常有用的，不过在其他情况下会变得很烦人。

你能够修改的第二个设置是 *ndots*，如果域名参数中包含的“ . ”的个数大于或等于这个值，那么解析器会在使用搜索列表之前，先查找这个域名。默认地，一个或多个点都行，这等价于 *ndots:1*。只要输入的域名包含至少一个“ . ”，解析器就会先试着查找输入的域名。如你相信你的用户更有可能输入部分域名而需要使用搜索列表时，你可以增加这个界限值。例如，如果你的本地域名是 *mit.edu*，并且你的用户习惯于输入：

```
% ftp prep.ai
```

然后由解析器自动把 *mit.edu* 附加在后面成为 *prep.ai.mit.edu*，你可能想把 *ndots* 增加到 2，这样一来，你的用户就不会因为不知情而导致向根名字服务器去查询顶级域 *ai* 中的名字了。你可以这样做：

```
options ndots:2
```

BIND 8.2引入了四个新的解析器选项：*attempts*、*timeout*、*rotate*和*no-check-names*。*attempts* 允许你指定在放弃之前向 *resolv.conf* 文件中每个名字服务器发送查询的次数。如果你认为新的默认值 2 对你的名字服务器来说太小了，可以将它改回 BIND 8.2.1 之前的默认值 4，用下面的选项：

```
options attempts:4
```

最大值为 5。

timeout 允许你指定每个对 *resolv.conf* 文件中名字服务器的查询的初始超时时间。默认值为 5 秒。如果你希望解析器更快地重新发送，那么可以将之改为 2 秒：

```
options timeout:2
```

最大值为 30 秒。对第二轮以及接下来的几轮查询，解析器会把初始超时时间加倍，再除以 *resolv.conf* 文件中名字服务器的数量。

*rotate*使你的解析器可以使用*resolv.conf*文件中所有的名字服务器而不止是第一个。只要你的解析器的第一个名字服务器总是运行正常，它会回答你的解析器的所有查询。除非该名字服务器非常繁忙或停机了，你的解析器是不会查询*resolv.conf*文件中的第二个或第三个名字服务器的。如果你想分散负载，可以设置：

```
options rotate
```

使得被配置的解析器按照轮转的顺序使用*resolv.conf*文件中的名字服务器。换句话说，解析器还是首先会查询*resolv.conf*文件中的第一个名字服务器，但是对要查询的第二个域名，它会首先查询第二个名字服务器，依次类推。

注意，许多程序不能利用这一功能，因为大多数程序是初始化解析器、查找一个名字、然后退出。例如，这种轮转对反复使用的*ping*命令没有用，因为每个*ping*进程都会初始化解析器、查询*resolv.conf*文件中第一个名字服务器、在再次使用解析器之前退出。每个连续的*ping*调用都不清楚上一次使用的名字服务器是哪一个——即使上一次运行的就是一个*ping*命令。但是那些要发送很多查询的运行时间较长的进程（比如，*sendmail*守护进程）就可以利用轮转的好处。

轮转还有可能使调试更困难。如果你使用这个选项，当你的*sendmail*守护进程收到古怪回答时，你也搞不清用的到底是*resolv.conf*文件中哪个名字服务器。

最后，*no-check-names*允许你关掉解析器的名字检查功能，其默认值是打开的（注10）。这些例程检查响应中的域名，确保它们都遵循Internet主机命名标准，这个标准只允许主机名中使用字母、数字和连字符。如果你想使你的用户能解析带下划线或其他不合法字符的域名，就要设置这个值。

如果你想指定多个选项，还可以在*resolv.conf*文件中把它们写在同一行中：

```
options attempts:4 timeout:2 ndots:2
```

注释

从BIND 4.9的解析器开始能在*resolv.conf*中添加注释。以“#”或者“;”开头的行都被解释为注释，会被解析器忽略。

注10： 从BIND 4.9.4起的所有解析器都支持该功能。

关于 4.9 解析器指令的一个说明

如果你刚开始使用 BIND 4.9.3 或 4.9.4 解析器，要小心使用这些新的指令。你的主机上可能还有静态连接到程序中的旧解析器代码。通常，这不会是个问题，因为Unix解析器会忽略它们不认识的指令。但是不要指望你主机上的所有程序都会遵守新的指令。

如果你主机上的程序包括很老的解析器代码（4.8.3 以前的），而你仍然想使用那些能够利用 *search* 指令的程序，这里有一个技巧：在 *resolv.conf* 中同时使用 *domain* 指令和 *search* 指令，要把 *domain* 指令放在前面。老的解析器能理解 *domain* 指令并忽略 *search* 指令，因为它们不认识这个指令。新的解析器也能理解 *domain* 指令，但是 *search* 指令将覆盖它的作用。

解析器配置示例

讲了这么多理论，让我们来看看真实主机上的 *resolv.conf* 文件是什么样的吧。解析器的配置需求根据主机上是否有本地名字服务器是不一样的，所以我们将谈到这两种情况：主机上有本地名字服务器，以及主机使用远程名字服务器。

惟一解析器

我们作为 *movie.edu* 的管理员，要为某位教授配置她的新工作站，这个工作站上没有运行名字服务器。要决定这个工作站所在的域很容易 —— 因为只有一个 *movie.edu* 域可供选择。不过，这位教授正在与 Pixar 的研究者一起研究电信号补偿算法，所以将 *pixar.com* 加到她的工作站的搜索列表中是比较明智的。*search* 指令：

```
search movie.edu pixar.com
```

使她的工作站的本地域名为 *movie.edu*，同时如果在 *movie.edu* 中没有找到所需域名，就会在 *pixar.com* 中搜索。

新工作站是在网络 192.249.249/24 上，所以最近的名字服务器是 *wormhole.movie.edu*（192.249.249.1）和 *terminator.movie.edu*（192.249.249.3）。作为一条原则，你总是应该将主机配置成先使用最近的名字服务器。（这个可能最近的名字服务器是本

地主机上的名字服务器；其次是同一子网或同一网络上的名字服务器。）在这个例子中，两个名字服务器都一样近，但我们知道*wormhole.movie.edu*比较大一点（这台主机的速度更快一点，容量更大一点）。所以*resolv.conf*中的第一个*nameserver*指令应该是：

```
nameserver 192.249.249.1
```

因为，如果这位教授在使用计算机时出了问题，她的脾气会很大，我们还加入了*terminator.movie.edu*（192.249.249.3）作为后备的名字服务器。这样一来，即使由于某种原因*wormhole.movie.edu*停机了，教授的工作站也能获得名字服务（假设*terminator.movie.edu*和网络的其他部分都是正常的）。

最后*resolv.conf*看起来是这样的：

```
search movie.edu pixar.com
nameserver 192.249.249.1
nameserver 192.249.249.3
```

本地名字服务器

接下来，我们要配置大学的邮件分发器（mail hub）：*postmanrings2x.movie.edu*，让它能使用域名服务。*postmanrings2x.movie.edu*由*movie.edu*域中的所有组共享。近来，我们在主机上配置了一个名字服务器以减小其他名字服务器上的负载，所以我们要保证解析器首先查询本地主机上的名字服务器。

这种情况下，最简单的解析器配置就是什么都不配置：不创建*resolv.conf*文件，让解析器默认地使用本地名字服务器。*hostname*应该被设置为该主机的全称域名，这样解析器就可以确定本地域名了。

如果我们认为需要一个备份的名字服务器，这是一个谨慎的决定，我们可以使用*resolv.conf*。我们是否要配置一个备份的名字服务器很大程度上取决于本地名字服务器的可靠性。一个好的BIND名字服务器的实现会比一些操作系统的运行还要长久，所以可能不需要备份。但是，如果本地名字服务器在运行中曾经出现过问题，比如说，它偶尔会挂起并且停止响应查询，那最好还是增加一个备份名字服务器。

增加备份名字服务器时，只需要在*resolv.conf*里先列出本地名字服务器的IP地址

（写该主机的IP地址或者全零地址0.0.0.0，都可以），然后再列出一到两个备份名字服务器。除非你知道你系统的TCP/IP栈没有前面我们提到的那种问题，否则记住不要使用回送地址作为本机地址。

我们可不想事后再后悔，所以我们准备增加两个备份名字服务器。*postmanrings2x.movie.edu*在网络192.249.249/24上，所以*terminator.movie.edu*和*wormhole.movie.edu*对它来说都是最近的名字服务器（除了它自己以外）。我们将把前面解析器的例子中服务器被查询的顺序颠倒一下，这会有助于平衡这两台服务器之间的负荷。另外，因为我们不想在解析器尝试第二个名字服务器之前等满5分钟，所以我们将超时时间降低为2秒。*reslov.conf*文件最后看起来就是这样的：

```
domain movie.edu
nameserver 0.0.0.0
nameserver 192.249.249.3
nameserver 192.249.249.1
options timeout:2
```

把损失与不便降低到最小

现在你已经配置了你的主机，可以使用DNS了，还有什么要改变的呢？你的用户会被迫输入很长的域名吗？他们必须改变他们的邮件地址和邮件列表吗？

幸亏有了搜索列表，大部分东西还会和以前一样。不过也有一些例外，而且还有一些程序在使用DNS时会有显著的差异。我们会试着涉及所有常见的内容。

服务行为的差异

就像你在本章前面看到的，*telnet*、*ftp*、*rlogin*和*rsh*这样的程序会对不以“.”结尾的输入域名参数使用搜索列表。这意味着，如果你在*movie.edu*中（也就是，你的本地域名是*movie.edu*，而且你的搜索列表包括*movie.edu*），你可以输入：

```
%  telnet misery
```

或者：

```
%  telnet misery.movie.edu
```

甚至是：

```
%   telnet misery.movie.edu.
```

它们的效果都是一样的，对于其他服务来说也一样。同时你还能从另外一种不同的行为中获益：因为在你查找一个地址时，名字服务器可能会返回多个IP地址，现在的Telnet、FTP和Web浏览器会先尝试着去连接返回的第一个IP地址，例如如果连接被拒绝或是超时，就会接着试第二个，依次类推：

```
%   ftp tootsie
ftp: connect to address 192.249.249.244: Connection timed out
Trying 192.253.253.244...
Connected to tootsie.movie.edu.
220 tootsie.movie.edu FTP server (Version 16.2 Fri Apr 26
    18:20:43 GMT 1991) ready.
Name (tootsie: guest):
```

记住，通过使用 *resolv.conf* 的 *sortlist* 指令，甚至可以控制你的应用程序使用这些IP地址的顺序。

一个比较奇怪的服务是NFS。命令 *mount* 可以很正确地处理域名，而且你还可以在 */etc/fstab*（你的供应商可能称之为 */etc/checklist*）中加入域名。但是要小心 */etc/exports* 和 */etc/netgroup* 这两个文件。*/etc/exports* 控制着你允许不同的客户端使用NFS安装哪些文件系统。你也可以在 *netgroup* 文件中为一组主机分配一个名字，然后在 *exports* 文件中使用这一组名。

不幸的是，旧版本的NFS并没有真地使用DNS来检查 *exports* 或 *netgroup*，客户端在一个RPC（Remote Procedure Call，远程过程调用）包中告诉NFS服务器它的身份。因此，客户的身份是由客户自己来说的。在Sun RPC中主机使用的身份是本地主机的 *hostname*。所以你在上面两个文件中使用的名字都要和客户的 *hostname* 一致，而 *hostname* 不一定是主机的域名。

电子邮件

包括*sendmail*在内的一些电子邮件程序也没有像预期的那样工作，*sendmail*并没有像其他程序那样使用搜索列表。相反，当被配置成使用一个名字服务器后，它会使用一个叫做*canonicalization*的进程，把电子邮件地址中的名字转换成一个完整的规范域名。

在规范化过程中，*sendmail*将搜索列表应用到名字上，以ANY类型来查找数据，这就会匹配所有类型的记录。*sendmail*使用和较新解析器相同的规则：如果要规范化的名字中至少包含一个点，它就会先查找这个名字。如果被查询的名字服务器找到一个CNAME记录（一个别名），*sendmail*用别名所指向的规范名代替所查找的名字并将之规范化（如果该别名的目标也是一个别名）。如果被查询的名字服务器发现了一个A记录（一个地址），*sendmail*就会将解析到该地址的域名作为规范名字。如果名字服务器没有找到任何地址，但是找到了一个或多个MX记录，那么就执行下面中的一条：

- 如果还没有附加上搜索列表，那么*sendmail*就将解析到MX记录的域名作为规范名字。
- 如果已经加过了搜索列表中一个或多个元素，那么*sendmail*会注意到这个域名可能是一个规范名字，就会继续把搜索列表中的元素附加到它后面。如果在附加了搜索列表中一个元素之后发现返回了一个地址，那么返回该地址的域名就被认为是规范名字。否则*sendmail*把第一个找到的MX记录的域名用做规范名字（注11）。

*sendmail*在处理SMTP消息时会多次使用规范化，它会对目的地址和SMTP首部中的好几个字段都进行规范化（注12）。

当*sendmail*守护进程启动时，它也会为规范化后的域名设置宏$w。所以即使你只把你的*hostname*设置成一个很短的、只有一部分的名字，*sendmail*也会使用*reslov.conf*中定义的搜索列表来对它进行规范化。然后*sendmail*会把宏*$w*和规范化中遇到的所有别名都加到类*$=w*中，这就是邮件服务器的其他名字列表。

这一点很重要，因为类*$=w*名字是*sendmail*惟一能识别的名字，默认地，是本地主机的名字。*sendmail*会试着去转发那些它认为不是写给本地域名的邮件。所以，例如，除非你配置*sendmail*能够识别主机所有的别名（通过将它们添加到类*w*或文件

注11： 处理带有通配符的MX记录，必须要有这么复杂，我们将在第十六章中讨论这个问题。

注12： 一些老版本的*sendmail*使用不同的规范化技术：使用搜索列表，向名字服务器查询这个名字的CNAME记录。CNAME只匹配CNAME记录。如果找到了一个记录，就用CNAME记录右边的域名代替这个名字。

类*w*, 就像我们在第五章中讲的那样), 否则主机就会试图转发收到的所有目的地址不是规范域名的邮件。

关于类*$=w*还有一个重要的含义：*sendmail*只把类*$=w*的内容作为MX列表中本地主机的名字。因此，如果你在MX记录的右边用了不在*$=w*中的名字，那么主机就有可能不认识它。这会导致邮件循环，然后被退回给发信人。

关于*sendmail*最后要注意的是：当你启动名字服务器时，如果你正在运行着一个老版本的*sendmail*（版本8之前的），那么你应该在*sendmail.cf*文件中设置I选项。选项I决定了当查找一个目的主机失败时*sendmail*应该采取的动作。当使用*/etc/hosts*时，查找失败是致命的。如果你在主机表中搜索一个名字但是没有找到，那么以后它也不太可能奇迹般地出现，所以邮件收发器可能就把信退回去了。但是当使用DNS时，一次查询失败可能只是暂时的，譬如可能只是网络暂时出了问题。设置选项I会指示*sendmail*在查询失败时，把邮件先放入队列而不是退回给发信人。只用把OI添加到你的*sendmail.cf*文件中就设置了选项I。

更新 .rhosts、hosts.equiv 等等

一旦开始使用DNS, 你或许还需要消除你主机授权文件中主机名的歧义。现在那些使用简单的、只有一个部分的条目被认为是在本地域中。例如, *wormhole.movie.edu*上的*lpd.allow*文件可能包括：

```
wormhole
terminator
diehard
robocop
mash
twins
```

不过，如果我们把*mash*和*twins*移动到*comedy.movie.edu*区中，那么就不会允许它们访问*lpd*，因为*lpd.allow*中的条目只允许*mash.movie.edu*和*twins.movie.edu*进行访问。所以我们还必须对*lpd*服务器的本地域以外的主机名添加相应的域名：

```
wormhole
terminator
diehard
robocop
mash.comedy.movie.edu
twins.comedy.movie.edu
```

为了保证域名的正确性，其他一些需要检查主机名字的文件有：

```
hosts.equiv
.rhosts
X0.hosts
sendmail.cf
```

有时，只用通过规范化过滤程序（canonicalization filter）运行一下这些文件，就能够消除歧义了，这个规范化过滤程序就是一个使用搜索列表把主机名翻译成域名的程序。下面是一个很简短的用 Perl 写的规范化过滤程序，会对你有帮助的：

```
#!/usr/bin/perl -ap
# 要求每行一个主机名，在第一个字段(按照 .rhosts、X0.hosts 的方式)

s/$F[0]/$d/ if ($d)=gethostbyname $F[0];
```

提供别名

即使在配置了你的主机使用 DNS 后转换了所有 *.rhosts*、*hosts.equiv* 和 *sendmail.cf* 文件而且完成了所有基础工作，你的用户仍然必须调整过来使用域名。给他们带来的困惑越少越好，而且这些不便相对于 DNS 带来的好处会显得微不足道。

在配置了DNS后，一种减少困惑的方式是为那些广为人知但却不能再用原来名字访问的主机提供别名。例如，我们的用户习惯于输入 *telnet doofy* 或者 *rlogin doofy* 去访问在城市另外一边由电影工作室管理的BBS。现在他们必须开始使用 *doofy* 的完整域名 *doofy.maroon.com* 了。但是我们的大部分用户并不知道它的完整域名，要想通知他们每个人并让他们习惯这样，还是需要一些时间的。

幸运的是，BIND 将允许你为你的用户定义别名。我们要做的只是将环境变量 HOSTALIASES 设置成包含别名和域名之间映射关系的文件的绝对路径。例如，要设置一个整个系统都可以使用的 *doofy* 的别名，我们可以在系统的 shell 启动文件中将 HOSTALIASES 设置成 */etc/host.aliases*，然后在 */etc/host.aliases* 中增加：

```
doofy   doofy.maroon.com
```

别名文件的格式很简单：别名从行的第一列开始，接着是空白字符，然后是对应于别名的域名。域名不以“.”结尾而别名中不能包括“.”。

现在，当我们的用户输入 *telnet doofy* 或者 *rlogin doofy* 时，解析器就会在名字服务器查询中透明地把 *doofy* 转换成 *doofy.maroon.com*。用户看到的信息就会如下所示：

```
Trying...
Connected to doofy.maroon.com.
Escape character is '^]'.
IRIX System V.3 (sgi)
login:
```

但是，如果解析器回过头去使用 */etc/hosts*，那么我们的 HOSTALIASES 就不起任何作用了。所以我们应当在 */etc/hosts* 中也保留类似的别名。

过一段时间，可能再加上一些说明，用户会开始把他们在 *telnet* 显示信息中的完整域名同他们用的 BBS 联系起来。

如果你知道哪些是你的用户用起来可能会有问题的域名，通过使用 HOSTALIASES，你就能减少一些用户的麻烦。如果你不知道他们希望访问哪些主机，你可以让用户自己创建他们自己的别名文件，然后让每个用户在他的 shell 启动文件中把 HOSTALIASES 变量指向他自己的别名文件。

与供应商有关的选项

Unix 表面上是一个标准的操作系统，但是几乎有多少 Unix 标准就有多少 Unix 变种。同样，有多少 Unix 版本就有多少种解析器的配置。虽然几乎所有的解析器都支持最初的 Berkeley 语法，但是它们大部分又进行了非标准的增强或变化。我们会竭尽所能地覆盖大部分主流的解析器配置类型。

Sun 的 SunOS 4.x

配置运行 SunOS 的主机可是个挑战。SunOS 解析器的行为和其他主要厂商提供的标准 BIND 有很大的不同，这主要是因为 SunOS 的解析器同 Sun 的 NIS（网络信息服务）（就是以前的“黄页”）集成在一起了。

简要来说，NIS 提供了同一网络上的主机间重要文件保持同步的一种机制。这不仅包括 */etc/hosts*，还包括 */etc/services*、*/etc/passwd* 和其他文件。Sun 把 DNS 定位在

NIS 的一个备份选项上；如果 NIS 解析器在 NIS 的主机地图中无法找到一个主机名（或 IP 地址），你可以将它配置成查询名字服务器。

注意，解析器的功能是被作为*ypserv*程序的一部分来实现的，*ypserv*程序也能处理其他类型的NIS查询。所以，如果*ypserv*没有运行的话，你的解析器也就没有运行！（谢天谢地，Solaris 2 中的解析器不再要求你运行*ypserv*了。）使用*ypserv*来解析所有查询的一个好处是你不需要在 NIS 客户机上配置解析器，只需在 NIS 服务器上进行配置（注13）。NIS客户机会向NIS服务器来查询主机数据，如果有必要的话，NIS 服务器就会查询 DNS。

如果你运行的是 SunOS 4.X（Solaris 1），你可以（1）按照常规，将你的解析器配置成把 DNS 作为 NIS 的备份，（2）选择运行 NIS 时不使用主机地图，或（3）违背习惯，重新编译你的解析器只使用 DNS，或者你可以从 Internet 上找到免费的已经修改过的解析器。但是，我们必须警告你，据我们所知，Sun 不会支持修改过的解析器选项。

如果你运行的是 Solaris 2，那么解析器的配置是很简单的，用*nsswitch.conf*文件来指定：你想用 DNS 来进行名字解析。

修改过的解析器

对于这个选项，我们不会说太多，主要是因为关于这个过程已经有很好的文档说明，而且在BIND 4中这个过程几乎是自动化的。这个过程本身通常包括：通过从*libc.so*中删除调用NIS的例程并代之以纯DNS版本的例程来创建一个新的*libc.so*，这是一个标准的共享 C 库。虽然Sun 很大方地提供了必要的替代例程，但是他们并不提供支持。更糟糕的是，SunOS 4.x 中提供的例程是基于 BIND 4.8.1 的。

BIND 4 中含有在 SunOS 4.x 上安装 BIND 解析器例程的说明，它存在于代码包的*shres/sunos*子目录中一个叫 INSTALL 的文件中。不过对于 BIND 8 来说，这些说明就没有用了（而在 BIND 8.2.2 中根本就没有这些说明）。不过你还是可以通过匿名 FTP 从*ftp.isc.org*的*/isc/bind/src*下获得较早版本的BIND源码。如果你想从这些

注 13： 实际上，你也需要配置那些你使用 *sendmail.mx* 的主机上的解析器，*sendmail.mx* 是 Sun 改进版 *sendmail* 的 MX 记录。

源码中编译成一个替代的 SunOS 4.x 解析器，我们建议你使用 BIND 4.9.7 的源码，可以从 *ftp.isc.org* 的 */isc/bind/src/4.9.7/bind-4.9.7-REL.tar.gz* 得到。

如果你愿意放弃创建自己的共享C库，而使用他人的成果，你可以查看一下*resolv+*，它是基于 BIND 4.8.3 解析器的。*reslov+* 是 SunOS 的 4.8.3 解析器例程的增强版。它是由 Bill Wisner 编写的，允许管理员选择使用 NIS 和 DNS 进行查询的先后顺序（很像其他供应商为 Unix 增加的功能扩展，我们会在后面讨论到。）可以从 *ftp.uu.net* 的 */networking/ip/dns/resolv+2.1.1.tar.Z* 文件中得到新的例程，以及如何将它们编译成 *libc.so* 的说明。关于 *resolv+* 提供的功能的更多信息请看本章后面的 Linux 部分。

和 NIS 一起使用 DNS

但是，如果你想使用被广泛接受的方法，你就需要 NIS 和 DNS 和平共处。这可有点麻烦，所以让我们来更详细地讲一讲吧。我们不会涉及如何建立 NIS，这在 Hal Stern 的《Managing NFS and NIS》（O'Reilly）一书中有很详细的介绍。注意，这些说明只适用于 SunOS 4.1 以后的版本。如果你运行的是更古老的 SunOS，考虑一下使用 *ftp.uu.net* 上提供的替代库。或者更新你的 SunOS。

首先，你需要修改 NIS 来建立起它的地图文件 *Makefile*，这个文件会分发给网络上其他的主机。你应该在主 NIS 服务器而不是辅名字服务器上进行这个修改。

NIS 的 *Makefile* 文件是在 SunOS 主机上的 */var/yp/Makefile* 下。你需要做的修改很简单：你只需取消一行注释，再注释掉另外一行。找到下面的行：

```
#B=-b
B=
```

然后改为：

```
B=-b
#B=
```

然后，如下重新编译你的 NIS 主机地图：

```
#   cd /var/yp
#   rm hosts.time
#   make hosts.time
updated hosts
```

```
pushed hosts
```

这将会在主机地图中插入一条指令，它指示NIS在无法从主机地图中找到某个主机名字的时候去查询DNS。现在，当*ypserv*程序查找一个名字时，它就检查本地NIS的*domainname*对应的主机地图，如果它没有找到这个名字，就会去查询名字服务器。*ypserv*查询名字服务器时所使用的搜索列表是来自于本地NIS的*domainname*，或是*resolv.conf*中的*domain*指令。

接下来，如果需要的话，你应该创建一个*resolv.conf*文件。配置SunOS上的解析器要遵守的规则有点不同：

- 你不能把*hostname*设置为一个域名，让解析器从这里推断出本地域。
- 你也不能在*resolv.conf*中使用*search*指令，这是因为SunOS 4.x解析器是基于BIND 4.8.1的。解析器会忽略该指令，不会报错。
- 你可以把NIS的*domainname*设置成一个域名（如果你在使用NIS的话，你就必须把它设置成你的NIS域的名字），而解析器从它那里可以推断出本地DNS域的名字。但是它和BIND有所不同；例如，如果你把*domainname*设置成*fx.movie.edu*，搜索列表将只包括*movie.edu*。为什么搜索列表不包括*fx.movie.edu*？这是因为NIS假定它已经检查过了*fx.movie.edu*主机数据的一个权威数据源，也就是*fx.movie.edu*主机地图。
- 如果你想把本地域名设置成和你NIS的*domainname*一样，你可以在*domainname*前面加一个点“.”或加号“+”。为了把你的本地域名设置为*fx.movie.edu*，你可以把*domainname*设置为*+fx.movie.edu*或者*.fx.movie.edu*。
- 你也可以通过在*resolv.conf*中用*domain*指令来设置本地域名以覆盖NIS的正常行为。所以，如果你想强制解析器把*fx.movie.edu*包括在搜索列表中，你可以在*resolv.conf*中添加*domain fx.movie.edu*。
- 你甚至可以在*resolv.conf*中用*domain*指令设置一个和你NIS的*domainname*完全无关的DNS域名。在某些不幸的情况下，本地的NIS的*domainname*和本地DNS域名不相同，甚至根本就不相像。假设电影大学的信息技术系开始时建立了NIS域*it.ept.movieu*，而且还在使用中。为了不让DNS在这个根本就不存在的*dept.movieu*域中进行错误的查询，这个NIS域中的主机应该在*resolv.conf*中配置有*domain movie.edu*（或者类似的配置）。

最后 Sun 的 *resolv.conf* 对待 *nameserver* 指令就和 vanilla BIND 一样。所以一旦你配置好了 NIS 让它会使用 DNS，并且配置了你 NIS 的 *domainname*，可能还有你的本地 DNS 域名，你就可以在 *resolv.conf* 中任意添加名字服务器了，这样工作就完成了。

忽略 NIS

如果你想保留 Sun 的支持，可又不想使用讨厌的 NIS，你还有一个选择：可以在运行 NIS 时使用一个空白的主机地图。首先，建立起你的 *resolv.conf* 文件，像我们在前面一节中说过的那样，在 NIS 的 *Makefile* 中将 NIS 配置成使用 DNS，然后创建一个空白的主机地图。创建一个空白的主机地图只需要临时把 NIS 主服务器上的 */etc/hosts* 移动到别的地方，生成你的 NIS 主机地图，再替换掉 */etc/hosts* 文件：

```
%    mv /etc/hosts /etc/hosts.tmp
%    touch /etc/hosts # to keep make from complaining
%   cd /var/yp
%   make hosts.time
updated hosts
pushed hosts
%    mv /etc/hosts.tmp /etc/hosts
```

现在当解析器检查 NIS 时，什么都找不到，那它就会直接去查询名字服务器了。

如果你定期重新编译你的 NIS 地图，应该确保主机地图不会意外地根据 */etc/hosts* 来重新编译过了。最好的方法就是删除 NIS 的 *Makefile* 中的主机目标（host target）。你可以在 *Makefile* 中注释掉从

```
hosts.time: $(DIR)/hosts
```

开始到下一个空行之间的所有行。

Sun 的 Solaris 2.x

Solaris 2 到 2.5.1 的解析器都是基于 BIND 4.8.3 解析器的。Solaris 2.6、7 和 8 的解析器是基于 BIND 4.9.4-P1 解析器的。有趣的是，Sun 选择不遵循 RFC 1535 的建议，将搜索列表定义为只包括本地域名，所以即使 BIND 2.6 及其后续版本的解

析器的搜索列表都包括所有至少有两个标号的父域的名字。现在你能获得将Solaris 2.5 和 2.5.1 解析器升级到 BIND 4.9.3 的补丁程序（注 14）。

所有的 Solaris 2.x 解析器都支持一种扩展功能，它允许你指定解析器使用主机信息源的顺序，这些主机信息源包括 DNS、NIS、NIS+ 和 */etc/hosts*。这个服务顺序是在 */etc* 目录下 *nsswitch.conf* 文件中配置的。

实际上 *nsswitch.conf* 是用来配置检查各个源的顺序的。通过指定一个关键字，可以选择你想要配置的数据库。对于命名服务来说，数据库名字就是 *hosts*。*hosts* 数据库可用的服务源有 *dns*、*nis*、*nisplus* 和 *files*（这里指的是 */etc/hosts*）。配置这些服务源使用的顺序也就是在数据库名字后面如何排列这些名字顺序的问题。例如：

```
hosts:   dns files
```

就告诉解析器首先使用 DNS（也就是查询一个名字服务器），然后再检查 */etc/hosts*。默认地，当前服务源不可用或要查找的名字不存在时，解析会转到下一个主机信息源（例如，从 DNS 转到 */etc/hosts*）。你可以在各个主机信息源之间指定条件（condition）和动作（action），用方括号括起来，这样就能改变上述行为。可能的条件有：

UNAVAIL
: 没有配置这个主机信息源（对 DNS 来说就是没有 *resolv.conf* 文件，本地主机也没有运行名字服务器）。

NOTFOUND
: 这个主机信息源无法找到被查找的名字（对 DNS 来说就是被查找的名字或被查找的数据类型不存在）。

TRYAGAIN
: 这个主机信息源忙，但下一次可能可以响应（例如，解析器在试图查找一个名字时超时）。

SUCCESS
: 要查找的名字在指定的信息源中找到。

注 14： 查看一下 *http://sunsolve.sun.com/pub-cgi/show.pl?target=patches/patch-access*，上面列出了最新的补丁号。

对于上述每一个标准，你都可以指定解析器是应该“继续”（continue）再转到下一个主机信息源或是“返回”（return）。默认的动作是*SUCCESS*就返回，其他条件就继续。

例如，如果你希望你的解析器在收到NXDOMAIN（无此域名）回答时就停止查找域名，而当DNS不可用时去检查*/etc/hosts*，你可以用下面这样的配置：

```
hosts:  dns [NOTFOUND=return] files
```

顺便说一下，默认的Solaris的*nsswitch.conf*配置是由你对SunInstall的回答决定的。不过信不信由你，默认的*nsswitch.conf*配置中没有一个是将*dns*作为主机信息源的。这难道就是因为是*.com*的缘故吗？

nscd

在Solaris 2.x中，Sun引入了一个名字服务缓存守护进程，称为*nscd*。*nscd*缓存对*passwd*、*group*和*hosts*主机信息源的查询结果。你可以把*nscd*想像成非常类似于只进行缓存的名字服务器，但是它还提供对*passwd*和*group*主机源中信息的查找。Sun引入*nscd*的意图是希望通过把那些被经常查找的名字保存在缓存中，以此来提高性能。不幸的是，有传言说*nscd*实际上有时会使DNS查找变慢，所以许多人都关闭了这个服务。此外，*nscd*还会与循环反复（round robin）特性相冲突（*nscd*把记录按照一种顺序保存在缓存中，而不会轮转它们的顺序）。

多用户方式启动时，默认地启动*nscd*，并且读入*/etc/nscd.conf*这个配置文件。管理员可以调整*nscd.conf*中的很多参数。最重要的参数有：

enable-cache hosts（yes | no）
: 决定是否让*nscd*把主机查找的结果保存在缓存里。

positive-time-to-live hosts value
: 决定nscd将肯定结果（如，地址）保留在缓存中的时间，以秒为单位。

negative-time-to-live hosts value
: 决定nscd将否定结果（如，NXDOMAIN）保留在缓存中的时间，以秒为单位。

不过，如果你并不确信*nscd*的作用，至少对DNS查找而言，你可以用：

```
enable-cache hosts no
```

来关闭这样的缓存。

HP 的 HP-UX

HP解析器的实现基本上就是BIND, HP-UX 8.0到10.00的解析器是基于BIND 4.8.3的，并且支持标准的 *domain*、*nameserver* 和 *search* 指令。主机使用 DNS、NIS 和主机表的顺序是固定的。如果配置了 DNS（也就是，如果有 *resolv.conf* 文件或在本地运行了名字服务器），那么主机就会使用DNS。如果没有配置DNS而在运行NIS，那么主机就用 NIS。如果 DNS 和 NIS 都没有运行的话，主机就使用主机表。只有在本章前面描述的那些情况下，主机才会去使用其他服务（比如说，解析器只使用一个名字服务器，这个服务器或者是在 *resolv.conf* 中列出，或者是本地主机默认的名字服务器，而且在同这个名字服务器联系时收到了四个错误）。

固定算法与其他供应商提供的算法相比，灵活性要差点，但是更容易排错。当你能按任意顺序来使用 DNS、NIS 和主机表时，诊断用户的问题将会是非常困难的。

HP-UX 10.10 到 11.00 解析器是建立在 BIND 4.9.x 基础上的。因此它们支持 BIND 4.9.x 的搜索列表和 *options ndots* 指令。

可以获得将 HP-UX 10.x 及其后续所有版本的名字服务器和辅助程序升级到 BIND 4.9.7的补丁程序。要获得这些补丁程序，请访问在*http://us-support.external.hp.com* 的 HP-UX 补丁库并且注册。然后可以搜索补丁数据库，找到最新的补丁程序。

HP-UX 11.10 解析器是基于 BIND 8.1.2 的。从配置的角度来说，BIND 8.1.2 解析器的配置同以前基于BIND 4.9.x的解析器的配置几乎是一模一样的：它们理解相同的配置指令，并且按照同样的方式推出默认搜索列表。

HP-UX 10.00引入了Solaris的*nssiwtch.conf*功能；也就是说，你可以使用*nsswitch.conf* 来控制解析器使用不同命名服务的顺序（注 15）。这里配置 *nsswitch.conf* 的语

注 15： 在 HP-UX 10.10 以前，你只能使用 *nsswitch.conf* 来配置主机信息源的解析顺序。从 10.10 开始，你也可以使用 *nsswitch.conf* 来配置 *services*、*networks*、*protocols*、*rpc* 和 *netgroup* 信息源的解析顺序了。

法与在 Solaris 中配置 *nsswitch.conf* 的完全一样。HP-UX 下关于主机数据库的默认设置是：

```
hosts: dns [NOTFOUND=return] nis [NOTFOUND=return] files
```

即使是在HP-UX 9.0这样老版本的补丁中也提供了*nsswitch.conf*功能及BIND 4.9.7名字服务器的升级。可以看一下网上 HP-UX 的补丁库。你可能需要相当多的补丁：

- 有关标准的共享 C 库 *libc.so* 的，它包含 HP-UX 中的解析器例程。
- 有关 *mount* 命令的，它是静态链接的。
- 有关 *nslookup* 的。
- 有关 *ifconfig* 和 *route* 命令的。
- 有关 HP 的可视化用户环境（Visual User Environment，简称 VUE）或公共桌面环境（Common Desktop Environment，简称 CDE）的，它们都是静态链接封装的。

IBM 的 AIX

最近版本的 AIX（包括 4.3 和 4.2.1）封装的解析器也是相当标准的。它的代码是基于BIND 4.9.x的，所以它也能理解*domain*、*search*、*nameserver*、*options*和*searchlist*指令；AIX最多支持三个*nameserver*指令。AIX版本4和版本4.1是基于BIND 4.8.3，所以它们能够处理 AIX 4.2.1 解析器的除了 *options* 和 *sortlist* 之外的所有指令。

AIX和普通BSD行为的一个不同点是，AIX根据*resolv.conf*文件是否存在来决定是否查询名字服务器。如果本地主机上没有*resolv.conf*，那么解析器就去读*/etc/hosts*。这意味着在一台运行名字服务器的主机上，你应该创建一个 0 字节的 */etc/resolv.conf* 文件，即使你不打算在里面使用任何指令。

AIX 4.3

AIX 4.3 解析器也支持两个环境变量：RES_TIMEOUT 和 RES_RETRY，它们允许你控制解析器的初始超时时间（按照*options timeout*指令的方式）和尝试的次数（按照 *options attempts* 的方式）。你可以在一个 shell 启动脚本里或命令行上设置这些值，如下所示：

```
#   RES_TIMEOUT=2 /usr/sbin/sendmail -bd -q1h
```

AIX 4.3支持一个用来控制解析顺序的机制，称为*irs.conf*，它同Solaris的*nsswitch.conf*很像。不过它们的语法有点不同。同*nsswitch.conf*一样，*irs.conf*也把数据库称为主机（hosts）。信息源的名字差不多都一样（*dns*、*nis*和*local*，这里用*local*而不用*files*），但是AIX在一行的结尾用关键字*continue*来表示解析器会接着尝试下一行中列出的下一个信息源。要表明一个信息源是权威的，当它返回否定回答（比如说，*/NOTFOUND=return/*）时解析器不用再尝试下一个信息源了，那么你需要在参数后加一个标签*=auth*。所以要告诉解析器先尝试DNS，只有在DNS没有配置时才接着尝试*/etc/hosts*，你就要使用像下面这样的*irs.conf*文件：

```
hosts dns=auth continue
hosts local
```

如果你要一个用户一个用户地指定顺序或者覆盖系统的默认值，可以使用NSORDER环境变量。NSORDER使用的参数和*irs.conf*一样，不过格式采用的是用逗号分隔的列表，如下：

```
NSORDER=dns,local
```

同*irs.conf*一样，也可以用*=auth*指定一个信息源是权威的：

```
NSORDER=dns=auth,local
```

AIX 4.2.1

AIX 4.2.1控制解析顺序的机制与上面讲的很相似，不过更有限一些。AIX 4.2.1使用一个叫做*/etc/netsvc.conf*的文件。这个*netsvc.conf*文件也称数据库为主机，不过它在数据库名字和信息源之间用的是等号而不是冒号，信息源之间用逗号，而用*bind*表示DNS，用*local*表示*/etc/hosts*。所以：

```
hosts = local,nis,bind
```

会让AIX解析器首先检查本地的*/etc/hosts*，再检查NIS主机地图，最后尝试DNS。同AIX 4.3一样，个人用户或进程可以通过设置NSORDER环境变量来覆盖由*netsvc.conf*配置的系统范围的解析顺序。

应该提醒你，可以使用AIX的系统管理接口工具（System Management Interface Tool，简称SMIT）来配置解析器。

Compaq 的 Tru64 Unix 和 Digital Unix

Tru64 Unix 5.0 封装的解析器是基于 BIND 8.1.2 解析器的。Digital Unix 4.0 封装的解析器是基于BIND 4.9.x解析器的。所以它们都理解本章中提到的所有五个主要解析器指令，但不理解 BIND 8.2 附加的指令，如 *options timeout*。

Tru64 Unix 5.0 解析器会进行名字检查，而且也能指定你允许出现在域名中的原来并不合法的字符。要实现这一目的，很简单，只用在 *allow_special* 指令后列出这些字符，用反斜杠符号引起来。例如，要允许名字中出现下划线，可以使用：

```
allow_special \_
```

你还可以指定参数 *all*，这就使得允许所有的字符，不过这恐怕不是个好主意。

Compaq 的两个版本的 Unix 都允许你通过一个叫做 *svc.conf* 的文件来配置解析器检查 NIS、DNS 和主机表的顺序（查看一下 *svc.conf(4)* 手册）（注 16）。*svc.conf* 也允许你配置其他数据库查询哪些服务，这些数据库包括邮件别名、认证检查（从 IP 地址映射到主机或域名）、口令和组信息，以及许多其他的东西。

通过 *svc.conf* 配置解析器，要使用数据库名字 *hosts*，后面跟着一个等号，然后是代表你想使用的服务的关键字，按照你所希望的顺序用逗号分隔。对于 *hosts* 数据库来说能够使用的合法关键字有 *local*（*/etc/hosts*）、*yp*（指“ Yellow Page ”，黄页，NIS 以前的名字）和 *bind*（代表 DNS）。对于 *hosts* 来说 *local* 必须是第一个服务。除了（可选地）在逗号后面和行尾，在每一行中都不要使用任何空白字符。例如：

```
hosts=local,bind
```

告诉解析器首先用 */etc/hosts* 来检查主机名，如果没有找到匹配的，就去使用域名服务。当主机有一个很小的包括本地主机的域名和 IP 地址、主机默认的路由器和其他启动时需要引用的主机的主机表时，这个配置就会非常有用。首先检查本地的主机表就避免了在启动时使用域名服务，这就避免了因为网络和 *named* 还没有启动所带来的各种问题。

Compaq 的 Unix 也包括一个称为 *svcsetup*（请查看 *svcsetup(8)* 手册）的工具，它允许你交互式地设置 *svc.conf* 文件，而不必使用编辑器。输入 *svcsetup* 就会出现一个

注 16： 老版本的 Ulrix 也支持 *svc.conf*。

窗口，你可以在这里选择你希望配置的数据库。*svcsetup* 还会提示你选择你希望使用的服务的顺序。

Silicon Graphics 的 IRIX

IRIX 6.5有一个BIND 4.9.x解析器和名字服务器。这个解析器理解*domain*、*search*、*nameserver*、*options* 和 *sortlist* 指令。以前版本的 IRIX 6.4 有一个基于 BIND 4.9.x 的名字服务器，而解析器却是基于 BIND 4.8.3 的。即使是对于 IRIX 5.3 这样的老版本也有补丁程序，可以将名字服务器升级到 BIND 4.9.7。查看 *http://support.sgi.com/colls/patches/tools/browse*，可以获得最新的补丁号。

在 IRIX 6.x 中，文件 *resolv.conf* 从原来的位置 */usr/etc/resolv.conf* 移到了更标准的位置 */etc/resolv.conf*。（为兼容老版本 IRIX 下编译的软件，你可能需要创建一个从 */urs/etc/resolv.conf* 到 */etc/resolv.conf* 的链接。）

同Solaris 2.x和HP-UX一样, IRIX 6.5支持*nsswitch.conf*文件。IRIX的*nsswitch.conf*文件和 Solaris 的格式一样，不过在条件列表中增加了一个 *noperm*（不允许使用该服务）。*hosts* 数据库的默认值是：

```
hosts: nis dns files
```

IRIX 的名字服务守护进程 *nsd* 会读取 *nsswitch.conf* 文件。同 Sun 的 *nscd* 一样，*nsd* 维护一个系统范围的缓存，用来存放以前查找的数据，包括从DNS和NIS获得的主机信息。*nsd* 支持许多 Solaris 和 HP-UX 不支持的 *nsswitch.conf* 中的属性设置。例如，你还可以在括号中添加超时时间设置，它将决定 *nsd* 会把来自于DNS的记录缓存多久：

```
hosts: files dns (timeout=600)      # 缓存的超时时间为10分钟
```

还可以用 *negative_timeout* 来指定否定缓存超时时间。要想了解所有的属性，请查阅 *nsd(1m)*手册。

较早版本的IRIX解析器（直到6.4以前的）不支持*nsswitch.conf*, 而支持*hostresorder* 指令。同 *nsswitch* 一样，*hostresorder* 指令允许管理员决定搜索 NIS、DNS 和本地主机表的顺序。个人用户可以通过设置环境变量*HOSTRESORDER*来决定他们的指令使用服务的顺序。IRIX 6.5 解析器忽略 *hostresorder* 指令。

hostresorder 使用一个或多个关键字 *nis*、*bind* 和 *local* 作为参数。（这些关键字分别对应哪些服务是很显然的。）关键字可以用空白字符或者斜杠分隔。空白字符表明，如果上一个服务没有返回结果（例如，主机表中没有找到这个名字，或者名字服务器返回“没有这个域”的信息）或者不可用（例如，名字服务器没有运行）的话，就应该尝试下一个服务。一个斜杠表明前面的服务是权威的，如果它没有返回结果的话，就应该停止解析。只有在前一个服务不可用时才去使用下一个服务。

Linux

从我们第一次发行本书时起，Linux 就席卷了整个计算机界。这有许多原因：Linux 是免费软件，它能比其他厂商的 Unix 版本更好地跟上 Unix 和 Internet 界的发展。一个例证就是 Red Hat 的 Linux 7.0（一种占主导地位的 Linux 的最新版本）封装有 BIND 8.2.2-P5 名字服务器。不过它的解析器还是基于 BIND 4.9.x 解析器的。它也支持文件 *nsswitch.conf*。

不过，老版本的 Linux 解析器还是基于 Bill Wisner 的 *resolv+* 库的，而它又是基于 BIND 4.8.3 的。因此，*resolv.conf* 文件中可以包括任何合法的 4.8.3 解析器指令（*domain*、*search* 和 *nameserver*，但不包括 *options* 或 *sortlist*），以及本章中所讲的较老的默认搜索列表。

顾名思义，*resolv+* 在标准的 4.8.3 解析器的基础上能提供一些增强的功能。其中包括能够决定查询 DNS、NIS 和 */etc/hosts* 的顺序（在较新的版本中被更标准的 *nsswitch.conf* 取代），能够识别某些类型的 DNS 哄骗，还能够为了优先本地子网而对回答中的地址记录进行重新排序。

所有这些增强功能都是由 */etc/host.conf* 文件控制的。*host.conf* 能够接受的最有趣的关键字是：

order

控制查询各种名字服务的顺序；合法的参数包括 *bind*、*hosts* 和 *nis*，关键字 *order* 后至少要有一个参数。如果有多个参数，那么就要用逗号分开。

nospoof

惟一可以设置的参数是 *on* 或 *off*。*nospoof* 指示解析器检查所有从远程名字服务器获得的反向映射（PTR）信息，也就是发射一个对回答中的域名的正向（地

址）查询。如果由地址查询返回的地址与解析器开始想反向映射的地址不相同，那么就忽略这个 PTR 记录。

reorder

惟一的参数可以设置成 *on* 或 *off*。如果把 *reorder* 设成了 *on*，解析器会对多宿主主机的地址进行排序，将在本地子网上的地址排在最前面。

Windows 95

Windows 95 带有自己的 TCP/IP 栈和 DNS 解析器。事实上，Windows 95 包含了两个 TCP/IP 栈：一个是局域网上使用的 TCP/IP，而另一个是拨号连接时使用的 TCP/IP。在 Windows 95 中配置解析器是非常图形化、非常生动的。打开 Control Panel（控制面板），点击 Network（网络），然后选择 TCP/IP Protocol（TCP/IP 协议），就进入了DNS配置面板，会出现一个如图6-1所示的对话框。选择DNS Configuration（DNS 配置）标签。

图 6-1　Windows 95 下的解析器配置

使用这个面板进行配置简直就是不言而喻的：复选 Enable DNS（启用 DNS）就打开了 DNS 解析，然后在 Host（主机）字段中输入主机名（在这个例子中就是它的域名的第一部分），然后在 Domain（域）字段中输入本地域名（第一个“.”后面的所有部分）。在 DNS Server Search Order（DNS 服务器搜索顺序）中，按照你希望的顺序填入最多三个你想要查询的名字服务器的 IP 地址。最后，你还要在 Domain Suffix Search Order（域后缀搜索顺序）下填上搜索列表中的域名，按照你希望的它们被附加的顺序。如果你空着 Domain Suffix Search Order 不填，Windows 95 会以与 BIND 4.8.3 解析器一样的方法从本地域名推出一个搜索列表。

当前 Windows 95 版本中一个有趣的地方：如果你在 Dial_up Networking（拨号网络，DUN）配置中有不止一个 ISP，那么你可以为每个与之对应的拨号连接都配置一组不同的名字服务器。要配置针对不同拨号网络的解析器设置，首先双击桌面上的 My Computer（我的电脑）图标，然后双击 Dial_Up Networking（拨号网络），再右击你想修改配置的连接的名字，选择 Properties（属性）。选择 Server Type（服务器类型）标签，然后选择 TCP/IP Setting（TCP/IP 设置）。你就会看到如图 6-2 所示的窗口。

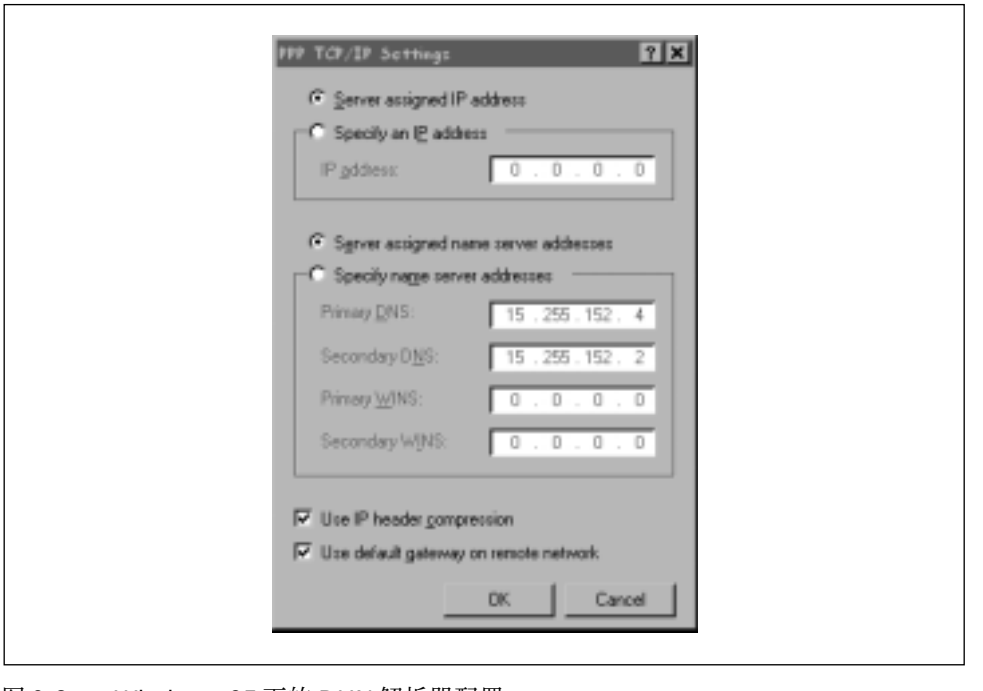


图 6-2　Windows 95 下的 DUN 解析器配置

如果你选择了Server assigned name server address（已分配名字服务器地址的服务器），解析器就会从你拨号进入的服务器那里获得它应该查询的名字服务器的信息。如果你选择了Specify name server addresses（指定名字服务器的地址），并且输入了一个或两个名字服务器的地址，Windows 95将会在拨号网络建立成功后去尝试使用这些名字服务器。

如果你使用多个ISP并且每一个都有自己的名字服务器的话，这就会非常有用。但是，在TCP/IP Properties（TCP/IP属性）面板中配置的名字服务器会覆盖掉和特定拨号网络相关的名字服务器。如果要使用和特定拨号网络相关的名字服务器特性，你除了选择使用DNS和指定本地的主机名外，不能选择TCP/IP属性面板中的任何其他选项。这个限制可能是因为在拨号网络和局域网的TCP/IP栈之间缺乏统一性而造成的，并且在DUN 1.3中得到了纠正。详情请见Knowledge Base中的文章Q191494（注17）。

Windows 98

Windows 98的解析器和Windows 95的解析器基本上是一样的。（实际上从图形上看，它们是完全一样的，所以我们就不再显示其屏幕图了。）两种解析器之间主要的区别在于Windows 98封装了Winsock 2.0（注18）。

例如，Winsock 2.0根据本地路由表来对响应进行排序。所以，如果名字服务器在一个响应中返回了多个地址，而其中有一个地址是在本地主机拥有显式（非默认）路由到达的网络上的，那么解析器就会将该地址排在响应的最前面。详情请见Knowledge Base文章Q182644。

Windows 98中也可以配置针对各个不同DUN的名字服务器。解析器既会查询TCP/IP Properties（TCP/IP属性）面板中列出的名字服务器，同时也会查询针对各个不同DUN的名字服务器，并将两组中最先收到的肯定回答作为回答。如果解析器只收到否定回答，那就返回该否定回答。

注17： 要想通过文章的ID号访问到一篇Knowledge Base的文章，请先登录 *http://search.suppoprt. microsoft.com/kb*，再选中Specific article ID number单选按钮，然后在搜索字段里输入文章的ID号。

注18： Windows 95中的Winsock可以升级到2.0，参见Knowledge Base文章Q182108。

Windows NT 4.0

在Windows NT中，LAN解析器的配置是通过一个看起来非常类似Windows 95风格的窗口来完成的，这是因为Windows NT 4.0结合了Windows 95的Shell。事实上，除了新的Edit（编辑）按钮和一些方便你对名字服务器和搜索列表中的项进行重新排序的小箭头外，这两种操作系统之间确实差别不大，如图6-3所示。

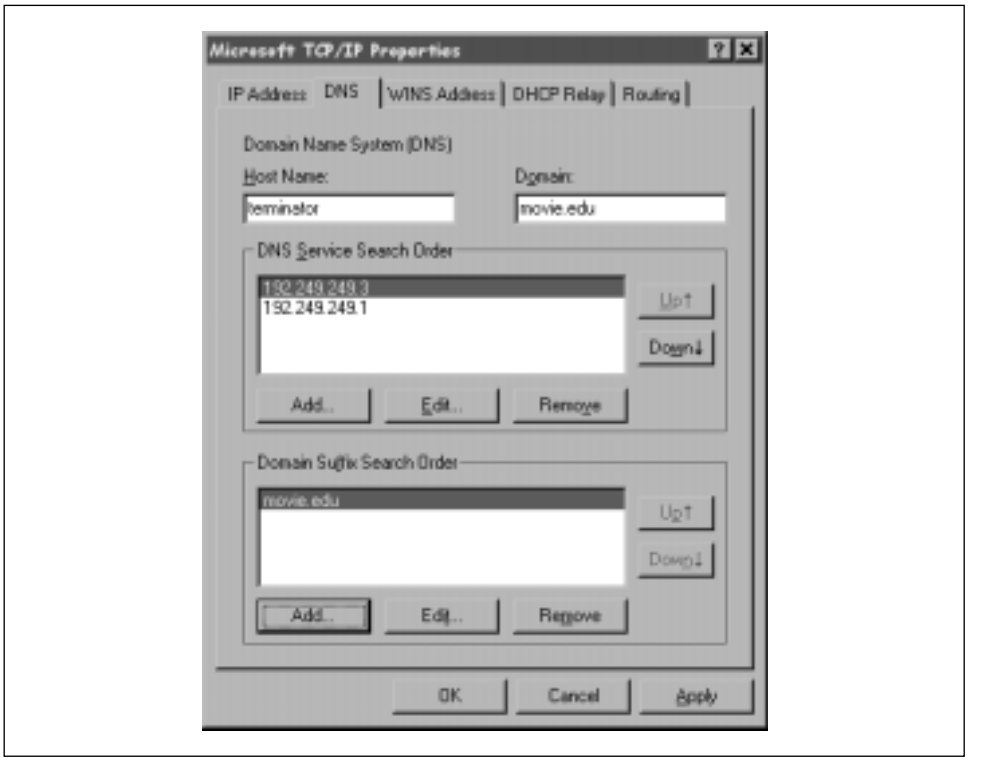


图6-3 Windows NT下的解析器配置

你先进入Control Panel（控制面板），点击Network（网络），然后选择Protocol（协议），就进入了DNS配置面板。双击TCP/IP Protocol（TCP/IP协议），再选择DNS标签。

Windows NT也允许用户配置针对某个拨号网络的解析器设置。如果你想这样做的话，首先进入My Computer（我的电脑），然后进入Dial-Up Networking（拨号网络），再拉下最上面的选择框，从中选择你要配置的解析器的拨号网络名称。然后点

击 More（更多）下拉框，选择 Edit Entry and Modem Properties（编辑入口和调制解调器属性）。在出现的窗口中选择 Server（服务器）标签，再点击 TCP/IP Setting（TCP/IP 设置）按钮。你将会看到和 Windows 95 中非常类似的一个窗口（如前所示）。如果你选择了 Server assigned name server addresses（已经分配地址的名字服务器），那么解析器就会从你所拨入的服务器那里获取要使用的名字服务器信息。如果你选择了 Specify name server addresses（指定名字服务器的地址）并且输入了一台或两台名字服务器的地址，那么 Windows NT 在拨号网络成功建立以后将使用这些名字服务器。当你断开拨号网络连接后，NT 将会重新使用局域网解析器的设置。

Windows NT 4.0 解析器会针对每个进程，根据返回的地址记录的生存期将名字到地址的映射放在缓存中。

在 Windows NT 4.0 的 Service Pack 4 中，微软对解析器进行了大量的更新。SP 4 解析器支持排序列表，就像 BIND 4.9.x 解析器那样，虽然这个排序列表是不可以配置的。不过这个排序列表是基于计算机的路由表的：计算机有直接路由可以到达的网络上的地址就排在响应的最前面。如果你不希望这样（例如，由于它干涉到循环反复），你可以用一个新的注册表值将它禁用。详情参见 Knowledge Base 文章 Q196500。

SP 4 解析器还让你能够用注册表值关掉解析器的缓存。详情参见 Knowledge Base 文章 Q187709。

SP 4 解析器还引入了新的重传送算法。解析器还是会将第一个查询发往 *DNS Server Search Order* 中的第一个名字服务器。不过，解析器只会等待 1 秒钟就重发该查询，而它会把查询重发给它所知道的所有名字服务器 —— 它通过静态配置、DHCP 和 RAS 了解到的所有名字服务器。如果这些名字服务器在 2 秒内都没有响应，解析器就会向所有这些名字服务器重发查询。解析器会不断地将超时时间加倍，再重发，总共是四次重发，15 秒钟。详情参见 Knowledge Base 文章 Q198550。

对你的名字服务器的管理员来说，这意味着给你的名字服务器带来很大的负担，所以要保证列在你的 SP 4 解析器的 *DNS Server Search Order* 中的第一个名字服务器的速度很快（通常会在 1 秒钟以内响应），而且你也不要配置 SP 4 解析器去查询不必要的名字服务器（也就是，使 *DNS Server Search Order* 保持最低限度）。

Windows 2000

Windows 2000的解析器可就有点难找了。点击Start（开始），然后是Settings（设置），接下来是Network and Dial-up Connections（网络和拨号连接），然后就会显示如图6-4所示的窗口了。

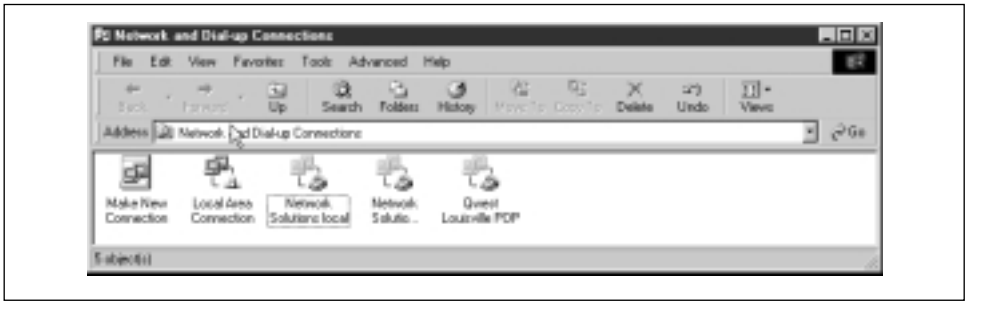


图6-4 Windows 2000的Network and Dial-up Connections（网络和拨号连接）

右击Local Area Connection（本地连接），再选择Properties（属性），这时你就会看到如图6-5所示的窗口了。

双击Internet Protocol（TCP/IP）（Internet协议（TCP/IP））。这就会列出基本解析器配置窗口，如图6-6所示。

如果选中Obtain DNS server address automatically（自动获得DNS服务器地址）单选按钮，解析器就会查询本地DHCP服务器让它使用的名字服务器。如果选中Use the following DNS server addresses（使用下面的DNS服务器地址）单选按钮，解析器就会查询你在Preferred DNS server（首选DNS服务器）和Alternate DNS server（备用DNS服务器）字段中指定的名字服务器（注19）。

注19： 真是应该感谢微软澄清了这些标识。在以前版本的Windows中，名字服务器有时被标成Primary DNS和Secondary DNS。有时这会误导用户在这些字段中输入某个区的主名字服务器和辅名字服务器。除此之外，“DNS”是“Domain Name System”的缩写，而不是“domain name server”的缩写。

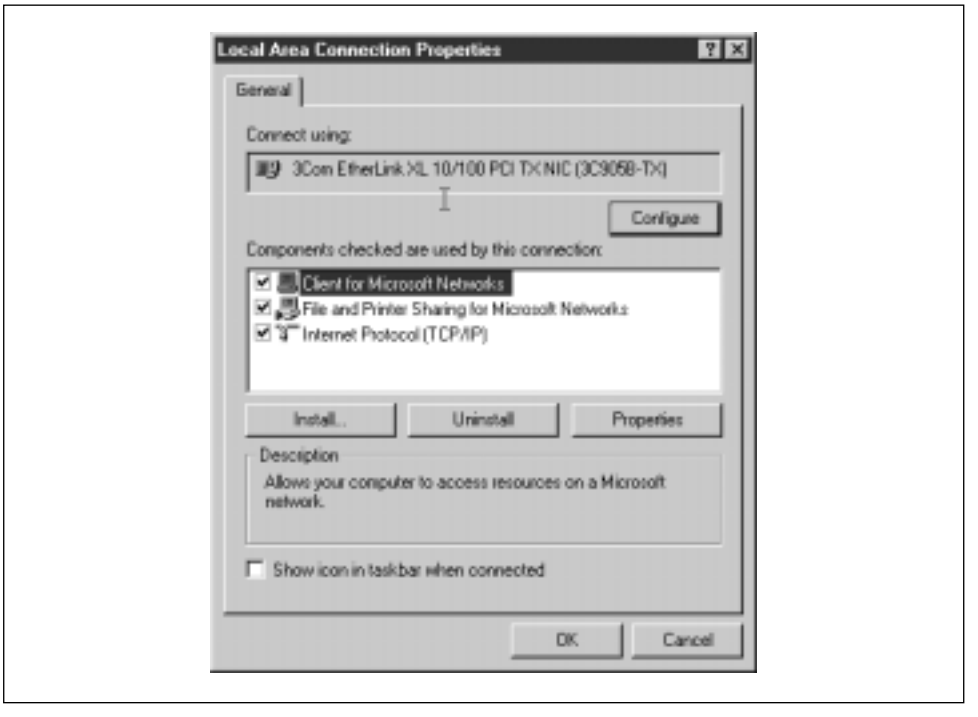


图 6-5 Windows 2000 的 Local Area Connection Properties（本地连接的属性）

要获得更高级的解析器配置，点击 Advanced...（高级 ...）按钮，再点击 DNS 标签，你会看到图 6-7 所示的窗口。

如果在基本解析器配置窗口中你已经指定了要查询的名字服务器的地址，在这个窗口中你又会在 DNS server addresses, in order of use:（DNS 服务器地址，按使用顺序排列）中见到它们。同在 Windows NT 4.0 解析器配置窗口中一样，有按钮允许你添加、编辑、删除和重排序所列出的名字服务器。对你能输入的名字服务器个数好像没有限制，但是列出多于三个就没有太大意义了。

Windows 2000 解析器使用了和 Windows NT 4.0 SP4 解析器一样的重发算法：会向配置过的所有名字服务器重发。因为你可以对每个网络接口（按照微软的说法叫做适配器）都配置一组不同的名字服务器，所以可能有许多的名字服务器。详情请见 Knowledge Base 文章 Q217769。

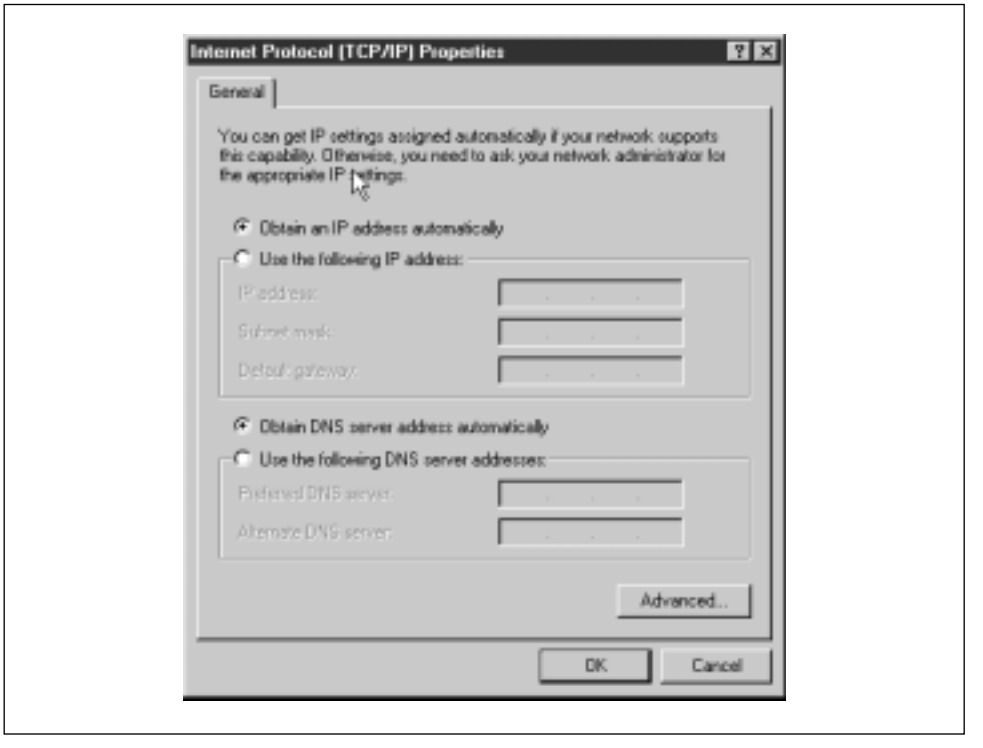


图 6-6 Windows 2000 基本解析器配置

因为目前存在分离的名字空间，所以有可能从两个不同的名字服务器获得两个不同的答案，因此 Windows 2000 解析器在查询多个名字服务器时会暂时忽略否定回答（无此域名和无此数据）。只有当收到来自所有为每个接口配置的名字服务器的否定回答时，解析器才返回一个否定回答。如果解析器哪怕从一个名字服务器收到了肯定回答，它都会返回该肯定回答。

选中 Append primary and connection specific DNS suffixes（附加主要的和连接特定的 DNS 后缀）单选按钮会使解析器使用主 DNS 后缀以及针对每条连接的 DNS 后缀，就像搜索列表一样。针对这条连接的 DNS 后缀也是在这个窗口中设置，在 DNS suffix for this connection（此连接的 DNS 后缀）右边的字段里。另外，DNS 的主后缀在控制面板中，点击 System（系统），选择 Network Identification（网络标识）标签，点击 Properties（属性）按钮，再点击 More...（更多 ...）按钮。然后就会显示如图 6-8 的窗口。

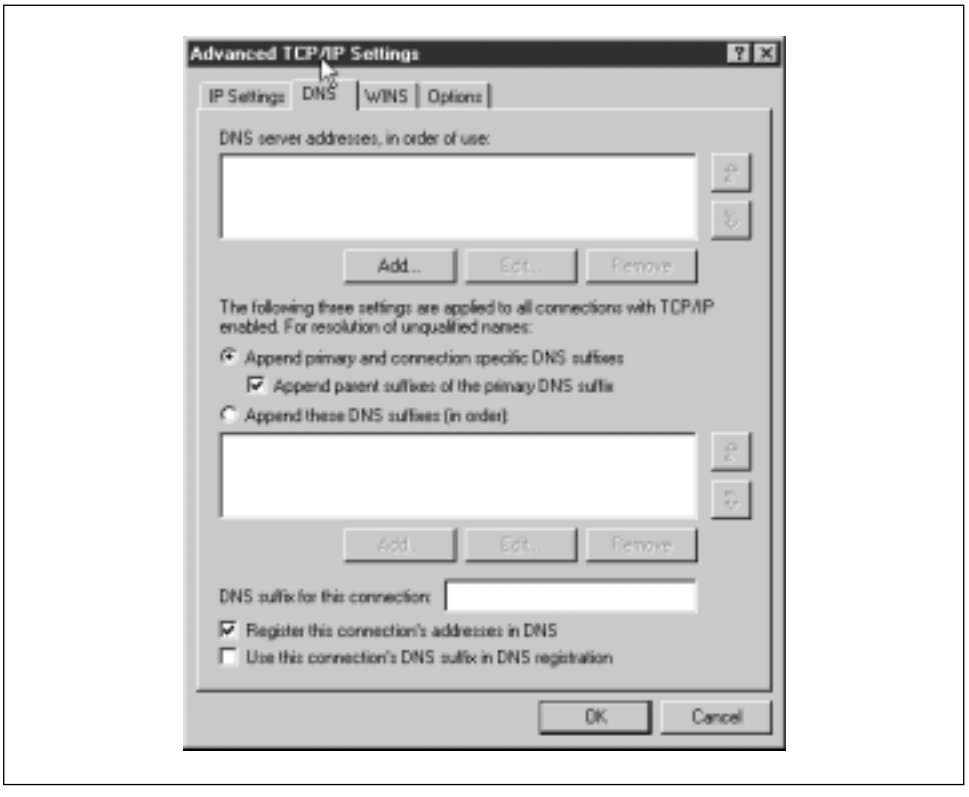


图 6-7 Windows 2000 高级解析器配置

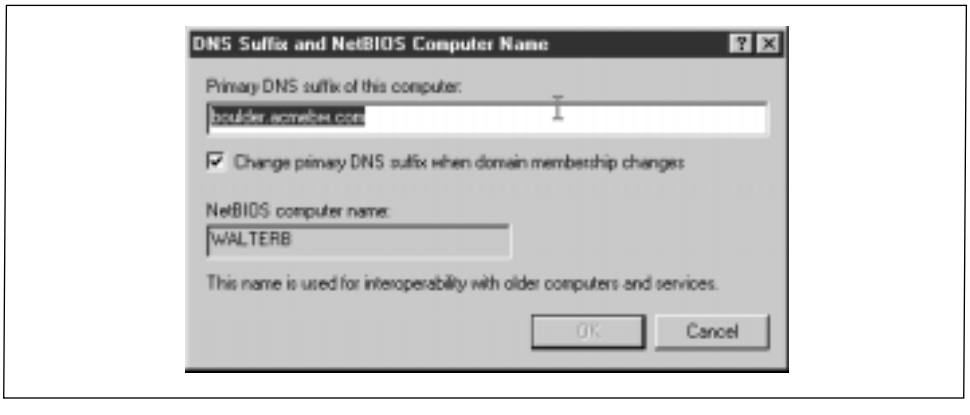


图 6-8 在 Windows 2000 中配置主 DNS 后缀

在 Primary DNS suffix of this computer（本计算机的主 DNS 后缀）标号下面的字段里输入它的值。

标为 Append parent suffixes of the primary domain suffix（附加主 DNS 后缀的父后缀）的复选框（见图 6-7）配置解析器去使用 BIND 4.8.3 样式的从主 DNS 后缀推出的搜索列表。所以，如果你的主 DNS 后缀是 *fx.movie.edu*，那么搜索列表就会包括 *fx.movie.edu* 和 *movie.edu*。注意，针对每个连接的 DNS 后缀不会“被转移”（devolved）（按照微软的说法）到搜索列表中，但如果配置了针对每个连接的 DNS 后缀，它还是会被包括在搜索列表中。

选中 Append these DNS suffixes（in order）（附加这些 DNS 后缀（按顺序））按钮，会配置解析器去使用下面字段中指定的搜索列表。同名字服务器列表一样，你也可以通过按钮和箭头来进行添加、编辑、删除和重排序。

最后，值得一提的是这个窗口中最下面的两个复选框：Register this connection's addresses in DNS（在 DNS 中注册此连接的地址）决定这个客户端是否会对添加将它的名字映射到该连接地址的地址记录使用动态更新；Use this connecton's suffix in DNS registration（在 DNS 注册中使用此连接的 DNS 后缀）控制该更新是否会使用与该连接有关的域名或该计算机的主 DNS 后缀。

这个自动注册的特性是设计用来保证你的 Windows 2000 客户端域名总是指向它当前 IP 地址的，即使该地址是由 DHCP 服务器传送的。（DHCP 服务器实际上添加的是 PTR 记录，该记录将客户端的 IP 地址映射回它的域名。）这也是 WINS（Windows Internet Name Service）——私有的也是备受非议的 Microsoft NetBIOS 命名服务的丧钟。一旦你的所有客户端都运行了 Windows 2000，它们都会使用动态更新来使它们的名字到地址的映射保持最新，而这对 WINS 来说也是一个打击。

不过，允许客户端动态更新区也许是个挑战，在本书最后一章我们还会讲到。

本章内容：

- 控制名字服务器
- 更新区数据文件
- 组织你的文件
- 在BIND 8和9中改变系统文件的位置
- BIND 8和9中的日志
- 使一切平稳运转

第七章

维护 BIND

“在我们国家里，”爱丽丝喘着粗气说，
“如果你像我们现在这样飞快地跑一段时间，
你肯定就到别的地方了。”
“真是个缓慢的国家！”女王说，“现在，
在这里，你瞧，你竭尽所能地跑，也只能保
持在原地。如果你想到别的什么地方去，
你至少要跑得比现在快一倍！”

这一章讨论关于维护名字服务器的话题。我们将会谈到名字服务器的控制、修改区数据文件以及如何保持根线索（root hint）文件永远是最新的。我们还将列出常见的 *syslog* 出错信息以及解释 BIND 的统计数据。

本章不包括解难排错。维护涉及到如何保持你的数据为当前最新的，还有监视你的名字服务器的运行状况。解难排错则是灭火 —— 那些时不时地冒出来的紧急情况。我们将在第十四章中谈论救火问题。

控制名字服务器

通常，管理员都是通过 Unix 信号来控制 BIND 名字服务器 —— *named*。名字服务器将收到某些信号解释为采取某个行动的指令，例如，重新加载所有已经改变的主

区。不过，可用的信号数量有限，而且信号也无法提供额外的信息（如，域名）来添加某个特定的区。

在BIND 8.2中，ISC引入了一种控制名字服务器的方法，它通过在特殊控制通道上给服务器发送消息来达到控制名字服务器的目的。这个控制通道可以是Unix域套接字（Unix domain socket），也可以是名字服务器用来监听消息的TCP端口。因为控制通道不会限于有限数量的离散信号，所以它更灵活、更有力。ISC声称控制通道是未来之路，管理员可以用它而不是信号来管理所有的名字服务器。

要用一个叫做*ndc*（在BIND 8中）或*rndc*（在BIND 9中）的程序来通过控制通道给名字服务器发送消息。从BIND 4.9开始就有*ndc*了，不过在BIND 8.2以前它还只是一个shell脚本，允许你用一些方便的参数（如，reload）来替换信号（如，HUP）。在本章后面我们会谈到那个版本的*ndc*。

ndc与controls（BIND 8）

如果执行时不带参数，*ndc*会试着通过Unix域套接字给本地主机上运行着的名字服务器发送消息，与之通信。这个套接字通常称为*/var/run/ndc*，不过有的操作系统使用不同的路径名。这个套接字通常由超级用户（root）所有，且只能由所有者读和写。BIND 8.2及后续的名字服务器在启动时创建这个Unix域套接字。你可以指定另外的路径名，或者允许套接字使用*controls*语句。例如，将套接字的路径改为*/etc/ndc*，并将所有权授予*named*组，让所有者和这个组都能对该套接字进行读和写，可以用：

```
controls {
    unix "/etc/ndc" perm 0660 owner 0 group 53;  // group 53 is "named"
};
```

其中的permission值必须是八进制数字（开头有个0，表示该数字是八进制的）。如果你不熟悉这个格式，参见*chmod(1)*手册。所有者和组的值也必须都是数字。

ISC建议你将对Unix域套接字的访问权局限于有权对名字服务器进行控制的管理人员，我们也赞成这样的建议。

你也可以用*ndc*通过TCP套接字给名字服务器发送消息，这个名字服务器对运行*ndc*的主机来说还可以是远程的名字服务器。要使用这种操作模式，要在命令行运行

ndc，还要带上 *-c* 参数，指定名字服务器的名字或地址，再跟一个斜杠，后面是名字服务器监听控制消息的端口号。例如：

```
#   ndc -c 127.0.0.1/953
```

将你的名字服务器配置成在某个 TCP 端口上监听控制消息，要使用 *controls* 语句：

```
controls {
    inet 127.0.0.1 port 953 allow { localhost; };
};
```

默认地，BIND 8 名字服务器不会监听任何 TCP 端口。BIND 9 名字服务器默认地监听端口 953，所以在此我们都使用这个端口。现在我们将名字服务器配置成只在本地回送地址上监听消息，只允许来自本地主机的消息。即使这样也并不十分保险，因为任何在本地主机上注册了的人都能控制名字服务器。如果你想更放得开一点（我们并不建议你这样），还可以扩大允许访问的列表，通过下面的配置可以让名字服务器监听所有的本地网络接口：

```
controls {
    inet * port 953 allow { localnets; };
};
```

ndc 支持两种操作方式：交互式和非交互式。在非交互式模式中，你在命令行上指定给名字服务器的命令，例如：

```
#   ndc reload
```

如果你没有在命令行上指定命令，就进入了交互式模式：

```
# ndc
Type   help  -or-   /h   if you need help.
ndc>
```

/h 会列出 *ndc*（不是名字服务器）理解的所有命令。下面这些都只适用于 *ndc* 的操作而不是名字服务器的操作：

```
ndc> /h
        /h(elp)                 this text
        /e(xit)                 leave this program
        /t(race)                toggle tracing (protocol and system events)
        /d(ebug)                toggle debugging (internal program events)
        /q(uiet)                toggle quietude (prompts and results)
        /s(ilent)               toggle silence (suppresses nonfatal errors)
```

```
ndc>
```

例如，*/d* 命令会使 *ndc* 产生调试输出（例如，它向名字服务器发送了什么，而得到的响应又是什么）。该指令对名字服务器的调试等级并没有影响。关于这一点，可以参阅后面会讲到的 *debug* 命令。

注意，是用 */e*，而不是 */x* 或 */q* 来退出 *ndc*，这同我们的直觉有点相悖。

help 命令会告诉你哪些命令可供你使用。下面这些命令控制名字服务器：

```
ndc> help
getpid
status
stop
exec
reload [zone] ...
reconfig [-noexpired] (just sees new/gone zones)
dumpdb
stats
trace [level]
notrace
querylog
qrylog
help
quit
ndc>
```

在此有两个命令没有列出来，不过你仍然可以使用它们：*start* 和 *restart*。没有列出它们是因为 *ndc* 告诉你的是名字服务器而不是 *ndc* 能理解的命令。名字服务器不能执行 *start* 命令，因为它只有在已经运行的状态下才能完成这个命令（而如果它已经在运行了，也就不需要启动了）。名字服务器也不能执行 *restart* 命令，因为如果它退出了，那么也就没有办法自己再重新启动了（它是不可能做到这一点的）。不过这不影响 *ndc* 的 *start* 或 *restart*。

下面是这些命令的具体说明：

getpid

显示名字服务器当前的进程号（processID）。

status

显示许多有用的名字服务器状态信息，包括名字服务器的版本、调试等级、正在进行的区传送的数目以及查询日志是否打开。

start

启动名字服务器。如果你想带命令行参数启动*named*，可以在*start*后指定这些参数。例如，*start —c /usr/local/etc/named.conf*。

stop

导致名字服务器退出，将改变的区数据文件写回。

restart

停止，然后启动名字服务器。同 *start* 命令一样，你可以在 *restart* 后面指定对 *named* 的命令行参数。

exec

停止，然后再启动名字服务器。不过与 *restart* 不同，你不能为 *named* 指定命令行参数；名字服务器只是以自己启动时相同的命令行参数启动自己的一个新备份。

reload

重新加载名字服务器。当一个主名字服务器修改过它的配置文件或区数据文件后，向它发送这个命令。将该命令发送给4.9或后续版本的辅名字服务器会使该名字服务器更新它的辅区，如果这些区不是最新的。你也可以指定一个或多个区的域名作为*reload*的参数；如果是这样，名字服务器就只会重新加载这些区。

reconfig [—noexpired]

告诉名字服务器为新的或已删除了的区检查配置文件。如果你已经添加或删除了一些区，但还没有改变任何现有的区的数据，就向服务器发送该命令。指定*—noexpired*标志是告诉名字服务器不要将有关区已过期的错误信息发给你。如果你的名字服务器是很多很多区的权威，而你又想避免看到大量的你早已经知道了的过期消息，那么这个标志就会很有用了。

dumpdb

将名字服务器的内部数据库转储（dump）到*/usr/tmp*（BIND 4）或名字服务器的当前目录（BIND 8）下的*named_dump.db*。

stats

将名字服务器的统计数据附加到*/usr/tmp*（BIND 4）或名字服务器当前目录（BIND 8）下的*named.stats*后面。

trace [level]

将调试信息附加到 */usr/tmp*（BIND 4）或名字服务器当前目录（BIND 8）下的 *named.run* 后面。较高的调试等级会增加调试信息的细节数量。想了解每个等级都会记录什么样的日志，请参阅第十三章。

notrace

关闭调试。

querylog（或 qrylog）

切换为用 *syslog* 记录所有查询。以 LOG_INFO 优先级记录该日志。*named* 必须编译成定义了 QRYLOG（默认地是定义了的）。这个特性是在 BIND 4.9 中增加的。

quit

结束 controls 会话。

rndc 与 controls（BIND 9）

同 BIND 8 一样，BIND 9 也是用 *controls* 语句来决定名字服务器如何监听控制信息的。除了允许 *inet* 子语句以外，其他语法都是一样的。（BIND 9.1.0 不支持以 Unix 域套接字为控制通道，而 ISC 暗示 BIND 9 可能永远也不会这样的。）

使用 BIND 9 可以不指定端口，名字服务器默认地监听端口 953。你还需要指定一个 *keys*：

```
controls {
       inet * allow { any; } keys { "rndc-key"; };
};
```

这决定了 *rndc* 用户要用什么加密密钥来验证身份才能给名字服务器发送控制消息。如果你没有指定 *keys*，在名字服务器启动之后会看到这样的消息：

```
Jan 13 18:22:03 terminator named[13964]: type 'inet' control channel
has no 'keys' clause; control channel will be disabled
```

在 *keys* 子语句中指定的密钥必须在一个 *key* 语句中定义：

```
key "rndc-key" {
        algorithm hmac-md5;
```

```
        secret "Zm9vCg==";
};
```

key 语句可以直接写在 *named.conf* 文件中，不过，如果无论什么人都能读到你的 *named.conf* 文件，把它放在另外的不能随便被人访问到的文件里会更保险一点，那么就在 *named.conf* 中包括该文件：

```
include "/etc/rndc.key";
```

目前惟一支持的算法是HMAC-MD5，该技术是使用快速MD5安全散列算法来进行身份认证（注1）。密码是简单地使用基数64对口令进行编码得到的，这个口令只有 *named* 和已被认证的 *rndc* 用户才知道。你可以用 *mmencode* 和 *dnssec-keygen* 这样的程序来产生密码，BIND自带有这些程序，我们会在第十一章中谈到这个问题。

例如，你可以用 *mmencode* 产生 *foobarbaz* 的基数64编码：

```
% mmencode
foobarbaz
CmZvb2JhcmJh
```

要使用 *rndc*，你需要创建一个 *rndc.conf* 文件来告诉 *rndc* 要用哪个认证密钥，而哪些名字服务器要使用它们。*rndc.conf* 文件通常是在 */etc* 下。下面就是一个简单的 *rndc.conf* 文件：

```
options {
        default-server localhost;
        default-key "rndc-key";
};

key "rndc-key" {
        algorithm hmac-md5;
        secret "Zm9vCg==";
};
```

这个文件的语法同 *named.conf* 的语法很相似。在 *options* 语句中，你可以定义默认的名字服务器，要给它发送控制消息（也可以用命令行覆盖），还可以定义呈现给远程名字服务器的默认密钥的名字（也可以用命令行覆盖）。

key 语句的语法同 *named.conf* 中 *key* 的语法一样，前面我们已经讲过了。*rndc.conf* 中的密钥的名字和 secret 的名字一样，必须同 *named.conf* 中密钥的定义相匹配。

注 1： 有关 HMAC-MD5 的更多信息请参见 RFC 2085 和 2104。

注意：记住，因为你在*rndc.conf*和*named.conf*中存储了密钥，所以要确保未被授权控制名字服务器的用户不能读到这两个文件。

如果你只用*rndc*来控制一台名字服务器，它的配置是很简单明了的。你要用与*rndc.conf*和*named.conf*一样的*key*语句来定义一个认证密钥。然后在*rndc.conf*的*options*语句中用*default-server*子语句将你的名字服务器定义为默认的待控制的服务器，用*default-key*子语句将这个密钥定义为默认的密钥。然后再运行*rndc*：

```
%  rndc reload
```

如果你要控制好几个名字服务器，你可以给每个名字服务器配不同的密钥。在不同的*key*语句中定义不同的密钥，然后在*server*语句中将每个密钥配给不同的服务器：

```
server localhost {
     key "rndc-key";
};

server wormhole.movie.edu {
     key "wormhole-key";
};
```

然后带*-s*选项运行*rndc*，指定要控制的服务器：

```
%  rndc -s wormhole.movie.edu reload
```

如果没有给某个名字服务器配密钥，你仍然可以用带*-y*选项的命令行来指定它要使用哪个密钥：

```
%  rndc -s wormhole.movie.edu -y rndc-wormhole reload
```

最后，如果你的名字服务器在一个非标准的端口上监听控制消息（也就是除了953以外的端口），你必须用*-p*选项来告诉*rndc*要连接到哪个端口：

```
%  rndc -s terminator.movie.edu -p 54 reload
```

现在有个坏消息：在BIND 9.0.0中，*rndc*只支持*reload*命令，而多个区的重新加载直到BIND 9.1.0才支持。虽然BIND 9.1.0不支持所有的BIND 8命令，但是它支持*reload*、*stop*、*stats*、*querylog*和*dumpdb*命令，还有新的*refresh*和*halt*命令：

refresh

安排立即维护某个 slave 区。

halt

停止名字服务器，不将未决的更新保存到日志文件（journal file）中。

使用信号

现在，让我们再回到过去，那时候我们只能用信号来控制名字服务器。如果你还在使用过去的 BIND（BIND 8.2 以前的），你就需要使用信号来管理你的名字服务器。我们给出了可以向名字服务器发送的信号列表，还会告诉你它们分别对应于现在的哪个 *ndc* 命令。如果你有 *ndc* 的 shell 脚本（从 BIND 4.9 到 8.1.2），那就不用太在意信号的名字，因为 *ndc* 会将命令翻译成相应的信号。注意，不要将 BIND 4 版本的 *ndc* 用在 BIND 8 上，因为请求统计数据的信号已经变化了。

命令	信号
reload	HUP
dumpdb	INT
stats	ABRT (BIND 4) or ILL (BIND 8)
trace	USR1
notrace	USR2
querylog	WINCH
stop (BIND 8)	TERM

在使用老版本的 *ndc* 时，要切换查询日志，需要用：

```
#  ndc querylog
```

就像用较新版本的 *ndc* 一样。不过在这种情况下，*ndc* 会追踪 *named* 的 PID（进程号），给它发送一个 WINCH 信号。

如果没有 *ndc*，你就不得不自己手工地完成 *ndc* 所做的工作了：找到 *named* 的进程号，再给它发送适当的信号。BIND 名字服务器将它的进程号放在一个称为 *pid* 文件的磁盘文件中，这样就非常容易找到这个进程号而不必使用 *ps* 命令。*pid* 文件的路径通常是 */var/run/named.pid*。有的系统上 *pid* 文件是 */etc/named.pid*。检查一下

named 手册就能知道你的系统上 *named.pid* 在哪个目录下了。因为名字服务器的进程号是 *pid* 文件的惟一内容，发送一个 HUP 信号十分简单：

```
#    kill -HUP `cat /var/run/named.pid`
```

如果你找不到 *pid* 文件，你总是可以用 *ps* 命令找到进程号的。在一个基于 BSD 的系统上可以使用：

```
%    ps -ax | grep named
```

在一个基于 SYS V 的系统上使用：

```
%    ps -ef | grep named
```

不过，使用 *ps* 你可能会发现有多个 *named* 进程正在运行，这是因为名字服务器会生成多个子进程去进行区传送。在区传送期间，需要区数据的名字服务器（辅）会启动一个子进程，而提供区数据的名字服务器（它的主名字服务器）也会启动一个子进程。我们在这里要稍微离一下题，解释一下为什么要使用子进程。

BIND 4 和 BIND 8 的辅名字服务器生成一个子进程来执行区传送。这就使得当子进程将区数据从主服务器传送到本地磁盘上时，辅名字服务器仍然能够继续回答查询。一旦传输完毕，辅名字服务器就会读入新的数据。使用一个子进程来进行区传送解决了BIND 4.8.3以前版本中的一个问题，也就是辅名字服务器在进行区传送时无法回答查询。对那些需要加载许多区或很大的区的名字服务器来说，这个问题是非常讨厌的，它们可能会在相当长的时间里不响应任何查询。

BIND 9 的辅名字服务器采用了新的结构，不再需要产生子进程来避免区传送时不能响应查询。名字服务器可以在进行区传送的同时回答查询。

版本8和9的主名字服务器不会生成子进程来为辅名字服务器提供区。与之相反，主服务器在响应查询的同时传送区。如果主服务器在进行该区的传送过程中，又从区数据文件中加载新的区数据，它将会中止区传送，而去从区数据文件中加载新的区数据。而辅服务器就必须在主名字服务器完成新区的加载后，再试着进行区传送。

版本 4 的主名字服务器会生成一个子进程来为辅名字服务器提供区。这对运行主名字服务器的主机来说会增加额外的负荷，特别是，如果这个区非常大或者同时进行多个区传送。

如果 *ps* 命令的输出表明有多个名字服务器，你应该很容易就能指出哪个是父进程，哪些是子进程。由辅名字服务器生成的用来进行区传送的子进程被称为 *named-xfer* 而不是 *named*：

```
root  548 547  0 22:03:17 ?     0:00 named-xfer -z movie.edu
      -f /usr/tmp/NsTmp0 -s 0 -P 53 192.249.249.3
```

一个由主名字服务器生成的子进程会改变它的命令行选项来指明它正在为哪个辅名字服务器提供区数据：

```
root 1137 1122 6 22:03:18 ?     0:00 /etc/named -zone XFR
     to [192.249.249.1]
```

你可能会遇到某个不会修改其命令行选项的 *named* 版本，不过你还是可以通过检查它们的进程号和父进程号来辨别多个 *named* 进程之间的互相关系。所有的子进程都会把它们的父名字服务器的进程号作为它们的父进程号。这看起来似乎是不言而喻的，但是你只能发送信号给父名字服务器进程。这些子进程在完成区传送之后就会消失。

更新区数据文件

你的网络上总有些什么在发生着变化 —— 增加了新的工作站、最终淘汰了一些陈旧的设备，或者把一台主机移到另一个网络上。每次这样的变化都要求你去修改区数据文件。你要手工修改这些文件吗？或者你觉得自己挺没用的要去找一个工具来帮忙吗？

我们首先要讨论如何手工地修改这些文件。然后再讨论一个很有用的工具：*h2n*。实际上，我们建议你使用工具来创建这些区数据文件，我们说你没有用才使用工具，这只是在开玩笑。或者至少用一个工具来帮助你增加序列号。DNS 区数据文件的语法很容易让你写错。把地址记录和指针记录分开放在不同的文件中也没有什么作用，因为两者之间必须保持一致。但是，即使你使用了工具，你知道更新这些文件时会发生什么情况也是很重要的，所以我们首先要谈谈手工更新的方法。

添加和删除主机

在最初创建你的区数据文件后，很明显，当要添加一台新主机时你需要去修改些什么。我们将逐一讨论这些步骤，以免你并不是设置这些文件的人，或者你只想遵循一些列好的步骤来执行。要修改的必须是你的主名字服务器的区数据文件。如果你修改的是你辅名字服务器上的备份区数据文件，那么辅名字服务器的数据会发生变化，但是下次区传送将会覆盖这些修改。

1. 更新 *db.DOMAIN* 文件中的序列号。序列号可能在文件的开始位置，所以先改这个很容易，而且也免得你忘了。

2. 添加有关该主机的所有 A（地址）、CNAME（别名）和 MX（邮件交换器）记录到 *db.DOMAIN* 文件中。当我们网络上添加了一台新的主机（*cujo*）时，我们将在 *db.movie.edu* 文件中添加下列资源记录：

```
cujo  IN  A  192.253.253.5   ; cujo 的 internet 地址
      IN MX  10 cujo         ; 如果可能，直接 mail 到 cujo
      IN MX  20 terminator   ; 否则，递送到我们的邮件分发器
```

3. 更新每一个该主机地址对应的 *db.ADDR* 文件的序列号，并且将 PTR 记录添加到其中。*cujo* 只有一个地址，位于网络 192.253.253/24 上，所以我们在 *db.192.253.253* 文件中添加如下的 PTR 记录：

```
5  IN PTR cujo.movie.edu.
```

4. 重新加载主名字服务器，这会迫使它加载新的信息：

```
#  ndc reload
```

   如果你有时髦的 BIND 8.2 或更新的版本，可以只重新加载改变了的区：

```
#  ndc reload movie.edu 253.253.192.in-addr.arpa
```

主名字服务器将加载新的区数据。辅名字服务器将在SOA记录中定义的刷新数据的时间间隔到了以后去加载新的数据。

有时，你的用户可能不愿意等辅名字服务器自己去加载新的区数据，他们想马上在辅名字服务器上使用这些新的数据。（读到这里时，你是摇头否认还是点头同意呢？）能强制一个名字服务器马上开始加载新的数据吗？如果是版本 8 或 9 的主名字服务器和辅名字服务器的话，辅名字服务器会很快加载新数据的，这是因为主名字服务器在自己修改完成后的15分钟内会通知辅名字服务器发生了改变。如果你的名字服务器是4.9或后续版本，你可以像对待主名字服务器那样重新加载服务器。重

新加载会使名字服务器更新它的所有辅区。如果你的名字服务器是4.8.4或者更早的版本，那就要删除辅名字服务器的所有备份区数据文件（或者只删除你希望强制它现在加载的文件），然后终止辅名字服务器，再重新启动。因为备份文件不见了，辅名字服务器就必须马上从主名字服务器那里加载新的区数据。

要删除一台主机，你需要从*db.DOMAIN*文件以及从每个与该主机相关的*db.ADDR*文件中删除相应的资源记录。增加你所修改的每个区数据文件的序列号，再重新加载你的主名字服务器。

SOA 序列号

每个区数据文件都有一个序列号。每次修改区数据文件中的数据后，都必须增加其序列号。如果不增加序列号的话，该区的辅名字服务器就不会获取修改后的数据。

增加序列号很简单。如果最初区数据文件中有如下的 SOA 记录：

```
movie.edu. IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                                  100     ; 序列号
                                  3h      ; 刷新
                                  1h      ; 重试
                                  1w      ; 期满
                                  1h )    ; 否定缓存 TTL
```

那么修改过的区数据文件的 SOA 记录就应该如下所示：

```
movie.edu. IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                                  101     ; 序列号
                                  3h      ; 刷新
                                  1h      ; 重试
                                  1w      ; 期满
                                  1h )    ; 否定缓存 TTL
```

这个简单的修改对于将区数据分发到你所有的辅名字服务器上是至关重要的。修改了区之后没有增加序列号是最常见的错误。头几次修改区数据文件时，因为很新鲜而且你的注意力很集中，你可能还记着增加序列号。到了后来，当修改区数据文件几乎变成你的本能后，你可能想改得快一点，忘记了增加序列号……结果没有一个辅名字服务器会加载新的区数据。这就是为什么你需要一个工具来帮助你增加序列号！这个工具可以是*h2n*或者是你自己写的工具，不过最好还是使用工具来增加序列号。

BIND 确实也允许你用小数作为序列号（如，1.1），但是我们推荐你只使用整数值。BIND 版本 4 是这样处理小数序列号的：如果序列号中有小数点，BIND 将把小数点左边的数乘以 1000，然后把小数点右边的数字连接到结果的后面。因此，1.1 这样的数字在内部会被转换为 10001，1.10 被转换为 10010。这必然会产生一些不正常的情况，比如说 1.1 比 2“大”，而 1.10 比 2.1“大”。这个看起来很别扭，与我们的常识不符，所以最好还是用整数序列号。

管理整数序列号有很多好方法。最明显的一种方法就是使用计数器：每次修改文件后将序列号加 1。另外一种方法是根据日期来生成序列号。例如，你可以使用 YYYYMMDD 格式的 8 位整数。假定今天是 1997 年 1 月 15 日，那么你的序列号就是 19970115。不过这个方法只允许每天修改一次文件，可能会不够用。那就在最后再增加两位来表示这是当天的第几次修改。1997 年 1 月 15 日的第一个序列号就是 1997011500。这一天的下一次修改就把序列号变成 1997011501。这个方案允许每天最多修改 100 次。这样做还有一个好处就是能在区数据文件中告诉你上次修改序列号的时间。如果你使用 *-y* 选项的话，*h2n* 会根据日期生成序列号。无论你使用哪种方法，序列号都必须是一个 32 位的整数。

重新开始一个新的序列号

如果你的某个区的序列号意外地变得非常大，而你想把它改回到一个比较合适的值该怎么办？有个方法对所有 BIND 的版本都适用，有一种方法适用于 BIND 4.8.1 及后续版本，还有一个方法适用于 BIND 4.9 及后续版本。

适用于所有 BIND 版本的方法是删除你所有辅名字服务器所记录的序列号。然后你就可以从 1（或者其他合适的值）开始计数。步骤是这样的：首先改变你主名字服务器上的序列号，然后重新启动，这时主名字服务器的序列号就是新的了。然后登录到你的一个辅名字服务器主机上，用 *ndc stop* 命令来终止 *named* 进程。删除所有的备份区数据文件（例如，*rm bak.movie.edu bak.192.249.249 bak.192.253.253*），再启动你的辅名字服务器。因为备份文件被删除了，辅名字服务器就必须加载一套新的区数据文件，同时也就获取了新的序列号。对每个辅名字服务器都重复同样的过程。如果有辅名字服务器不受你的控制，那你就必须和它们的管理员联系，要求他们也完成这个过程。

如果你的所有辅名字服务器都运行的是比 4.8.1 新而又比 BIND 9 老的 BIND（我们希望你不要使用 BIND 4.8.1），你可以利用特殊的序列号0。如果将一个区的序列号设为 0，那么在每个辅名字服务器下次检查时都会进行区传送。实际上，每次辅名字服务器检查的时候，都会传送该区，所以一旦所有的辅名字服务器的序列号都已经同步为 0 了以后，就别忘了增加该序列号。但是对于能将序列号增加到多大是有限制的。继续读下去吧。

如果我们先讲一些背景知识的话，会更容易理解另外一种处理序列号的方法（适用于版本4.9和后续版本的辅名字服务器）。DNS序列号是一个32位的无符号整数，它的取值范围从0到4,294,967,295。这个序列号使用的是“ 顺序空间算法 ”（sequence space arithmetic），就是说对于任何序列号而言，这个数字空间中有一半的数字（2,147, 483,647 个）比该序列号要小，而另外一半则比它要大。

让我们讨论一个有关顺序空间数字的例子。假定序列号是 5，那么从 6 开始到（5+2,147,483,647）之间的序列号都比序列号 5 大；从（5+2,147,483,649）到 4 之间的则是比它小的序列号。请注意，在达到 4,294,967,295 后序列号就会转向于 4。还要注意，我们没有包括序列号（5+2,147,483,648），这是因为它正好是数字空间的中点，而且根据实现方法的不同它有可能比 5 小，也有可能比 5 大。为安全起见，还是不要用它了。

现在回到原来的问题上。如果你的区序列号是 25,000，并且你想再次从 1 开始，那你可以通过两个步骤来加速使用完这些序列号。首先，把你的序列号增加到所允许的最大值（25,000 ＋ 2,147,483,647 = 2,147,508,647）。如果你所得到的结果比 4,294,967,295 还要大（最大的 32 位数），你应该把结果减去 4,294,967,295 以回到数字空间的起始位置。在修改了序列号之后，你必须等待你所有的辅名字服务器都从主名字服务器这里加载完新的区数据。然后，把区的序列号改为希望的值（1），这个值是大于当前的序列号（2,147,508,647）的。在你的辅名字服务器再次从这里加载了新的区数据后，一切就完成了！

其他的区数据文件条目

在你的名字服务器运行了一段时间后，你可能想在你的名字服务器中增加一些数据来帮助你更好地管理你的区。当别人问你的某台主机在哪里时，你被难倒过吗？你可能甚至都不记得那台主机是什么类型的了。现在管理员都必须管理越来越大的主

机群，所以很容易就忘记这样一些信息了。名字服务器可以帮助你。如果你的某台主机运行不正常，而其他地方的某个人注意到了，名字服务器能帮助他们和你进行联系。

迄今为止，这本书中已经讨论了SOA、NS、A、CNAME、PTR和MX记录。这些记录对于日常运行是至关重要的，名字服务器的运行需要它们，应用程序也要查找这些类型的数据。不过，DNS还定义了许多其他的数据类型。比较有用的资源记录类型就是TXT和RP，它们可以用来告诉你一台主机的位置和负责人。想了解常见（但又不是非常常见）的资源记录，请参阅附录一。

通用文本信息

TXT代表TeXT。这些记录只是一些简单的字符串，每个的长度要少于256个字母。BIND 4.8.3以前的版本不支持TXT记录。在版本4中，BIND将区数据文件的TXT记录限制为一个大概为2K的字符串数据。

你可以用TXT记录去记录任何东西，可以用它来记录主机的位置：

```
cujo  IN  TXT  "Location: machine room dog house"
```

BIND 8和9也有同样的2K限制，不过它允许你将TXT记录指定为多个字符串：

```
cujo  IN  TXT  "Location:" "machine room dog house"
```

负责人

毫无疑问，域管理员与RP（Responsible Person，负责人）记录之间是一种又爱又恨的关系。RP记录可以和任何域名连在一起，可以是域名空间中内部节点的域名，也可以是末端叶子节点的域名，它说明是谁在负责管理该主机或该区。例如，这就使得你可以找到那些给你惹麻烦的主机的相关负责人。但是在你的主机犯事时，这也能使别人找到你。

这个记录有两个参数来作为特定于每个记录的数据：一个遵循域名格式的电子邮件地址和一个指向有关联系人其他数据的域名。这个电子邮件地址和SOA记录中的那个格式一样：用“.”代替“@”。下一个参数是一个必须对应有TXT记录的域名。

TXT记录就包括有关于联系人的譬如全名和电话号码这样的任意格式的信息。如果你忽略了任何一项，都必须使用根域（“ . ”）来作为一个占位符。

这里是一些关于 RP（以及相关的）记录的例子：

```
robocop      IN  RP   root.movie.edu.  hotline.movie.edu.
             IN  RP   richard.movie.edu.  rb.movie.edu.
hotline      IN  TXT  "Movie U. Network Hotline, (415) 555-4111"
rb           IN  TXT  "Richard Boisclair, (415) 555-9612"
```

请注意，因为 *root.movie.edu* 和 *richard.movie.edu* 并不是真正的域名，而只是遵循域名格式的电子邮件地址，所以不需要有相关的 TXT 记录。

BIND 4.8.3 出现时还没有这种资源记录，但是 BIND 4.9 就支持它了。在使用 RP 之前，查看一下你名字服务器的文档看看是否支持。

通过主机表来生成区数据文件

正如在第四章中看到的那样，我们定义了一个将主机表信息转换为区数据的过程。我们已经用 Perl 写了一个这样的工具，叫做 *h2n*（注 2），它能自动执行这一过程。使用工具来生成你的数据有一个非常大的优势：假定我们写的 *h2n* 没有什么问题的话，那么在你的区数据文件中就不会有语法错误或前后不一致。一个常见的不一致错误就是，有一个关于某主机的 A（地址）记录，但是却没有相应的 PTR（指针）记录，或者其他类似问题。因为这些数据分布在不同的区数据文件中，所以很容易犯这样的错误。

h2n 都做些什么呢？根据 */etc/hosts* 文件和一些命令行选项，*h2n* 会创建有关你的区的数据文件。作为系统管理员，你要保证主机表永远是最新的。每次修改完主机表，你都要重新运行一次 *h2n*，*h2n* 就会重新创建每个区数据文件，为每个新文件都分配下一个大一点的序列号。你可以手工运行它，也可以每天晚上通过一个 *cron* 程序来运行它。如果使用了 *h2n*，你将永远不用担心会忘记增加序列号了。

首先，*h2n* 需要知道你的正向映射区的域名和你的网络号。（*h2b* 能够区别反向映射区的名字和网络号。）这些直接就对应成区数据文件的名字：*movie.edu* 区数据在文

注 2： 如果你忘了如何获取 *h2n*，请看前言中的“ 获取示例程序 ”。

件 *db.movie* 中，网络 192.249.249/24 数据则对应文件 *db.192.249.249*。你的正向映射区的域名和网络号是通过 *–d* 和 *–n* 选项来指定的，如下所示：

–d domain name
: 你的正向映射区的域名。

–n network number
: 你的网络的网络号。如果你生成多个网络的文件，就要在命令行中使用多个 *–n* 选项。要去掉网络号结尾的 0 及网络掩码的指定。

h2n 命令要求 *–d* 选项和至少一个 *–n* 选项。它们没有相应的默认值。例如，要创建区 *movie.edu* 的数据文件，该区包括两个网络，就要使用下面这样的命令：

```
%  h2n -d movie.edu -n 192.249.249 -n 192.253.253
```

如果要进一步控制这些数据，你可以使用其他一些选项：

–s server
: NS 记录的名字服务器。就像使用 *–n* 那样，如果你有多个主名字服务器或辅名字服务器的话，就要使用多个 *–s* 选项。BIND 8 或 9 名字服务器将会在区数据发生改变时 NOTIFY（通知）这些服务器。默认值是你运行 *h2n* 的那台主机。

–h host
: SOA 记录的 MNAME 字段对应的主机。这里的 *host* 必须是主名字服务器，这样才能确保正确使用 NOTIFY 特性。默认值是运行 *h2n* 的那台主机。

–u user
: 负责该区数据的人的电子邮件地址。默认值是运行 *h2n* 的主机的超级用户。

–o other
: 除序列号外的其他 SOA 值，是由冒号隔开的列表。默认值是 10800:3600:604800:86400。

–f file
: 从由 *file* 指定名字的文件中而不是从命令行中读取 *h2n* 的各个选项。如果你使用了许多选项，那么就把它们保留在一个文件里。

–v 4 | 8
: 生成 BIND 4 或 BIND 8 的配置文件，默认的是 BIND 4 的配置文件。因为 BIND

9 的配置文件格式基本上与 BIND 8 的相同，所以你可以用 *-v 8* 生成 BIND 9 名字服务器的配置文件。

-y
根据日期生成序列号。

下面这个例子使用了到目前为止所有提到过的选项：

```
%  h2n -f opts
```

文件 *opts* 的内容是：

```
-d movie.edu
-n 192.249.249
-n 192.253.253
-s terminator.movie.edu
-s wormhole
-u al
-h terminator
-o 10800:3600:604800:86400
-v 8
-y
```

如果一个选项要求一个主机名作为参数，你可以提供一个完整的域名（例如，*terminator.movie.edu*），或者只提供主机名（例如，*terminator*）。如果你只给出了主机名，*h2n* 将会把由 *-d* 选项给定的域名添加到它后面，生成一个完整的域名。（如果这个名字要求结尾有“ . ”，*h2n* 也会自动加上的。）

除了这些选项外，*h2n* 还有许多其他的选项。要了解所有的选项，请查看一下手册。

当然，要想从 */etc/hosts* 生成某些类型的资源记录并不那么容易，因为它并不包括所有必需的数据。你可能需要手工添加这些记录。但是 *h2n* 总是重写区数据文件，那你手工所做的修改会被覆盖掉吗？

h2n 提供了一个“ 后门 ”以方便插入这种数据。将这些特殊记录保存在一个名为 *spcl.DOMAIN* 的文件中，这里 *DOMAIN* 就是你区的域名的第一个标号。当 *h2n* 发现这个文件时，它就会在 *db.DOMAIN* 文件最后加上这么一行：

```
$INCLUDE spcl.DOMAIN
```

从而把这个文件也包括在区数据文件中。（本章后面会讨论 $INCLUDE 语句的。）例

如，*movie.edu*的管理员可以在*spcl.movie*文件中添加其他的MX记录，这样用户就可以直接向*movie.edu*发送邮件，而不必将邮件发送到*movie.edu*域中的主机。在找到这个文件后，*h2n*就会在区数据文件*db.movie*最后加上一行：

```
$INCLUDE spcl.movie
```

保持根线索是最新的

正如在第四章中解释过的那样，根线索文件告诉你的名字服务器根区名字服务器的位置。这个文件必须定期更新。根名字服务器虽然不会经常发生变化，但它们还是会有变化的。每个月或每两个月去检查一下你的根线索文件是个好主意。在第四章中，我们告诉过你通过FTP到*ftp.rs.internic.net*，就能获得这个文件。而且这可能也是保持文件最新的一个最佳方法。

如果你有*dig*这个工具的话，它工作起来非常类似*nslookup*，而且是包括在BIND当中的，你只用运行下面这个命令就能获得当前的根名字服务器列表：

```
%  dig @a.root-servers.net  .  ns > db.cache
```

组织你的文件

当你第一次建立起你的区时，组织文件很简单：只需把它们都放在一个目录里。其中包括一个配置文件和多个区数据文件。不过随着时间的推移，你的责任也越来越大。将会增加更多的网络，相应地有更多的*in-addr.arpa*区，也可能会对一些子域进行授权，你还要开始备份其他的区。过了一段时间后，你就会发现，如果在名字服务器的目录下执行一个*ls*命令，一屏可能都显示不下所有的文件了。这时你就该重新组织这些文件了。BIND有几个特性能够帮助你重新组织你的文件。

BIND 4.9及其后续版本的名字服务器支持一个叫做*include*的配置文件语句，它允许你在当前配置文件中插入另一个文件的内容。这就允许你把一个非常大的配置文件分成多个小文件。

区数据文件（所有BIND版本都有）都支持两个（注3）控制语句：$ORIGIN和

注3： 如果你算上$TTL就是三个，BIND 8.2及后续名字服务器支持它。

$INCLUDE。$ORIGIN 语句改变一个区数据文件的起点（origin），$INCLUDE 则在当前区数据文件中插入一个新的文件。这些控制语句不是资源记录，它们帮助维护DNS数据。特别的是，因为这些语句允许你把每个子域的数据存放在不同的文件中，所以把你的区划分成子域变得更容易。

使用多个目录

组织你的区数据文件的一个方法就是把它们放在不同的目录中。如果你的名字服务器是多个地点的区（既是正向映射又是反向映射）的主名字服务器，你可以把每个地点的区数据文件都放在各自的目录下。另外一种安排就是将所有的主区的数据文件保存在一个目录下，而将所有的备份区数据文件保存在另外一个目录下。如果你选择将你的主和辅区分开的话，让我们来看一下 BIND 4 配置文件会是什么样的：

```
directory /var/named
;
; 这些文件不专属于任何区
;
cache    .                          db.cache
primary  0.0.127.in-addr.arpa       db.127.0.0
;
; 这些是我们的主区文件
;
primary  movie.edu                  primary/db.movie.edu
primary  249.249.192.in-addr.arpa   primary/db.192.249.249
primary  253.253.192.in-addr.arpa   primary/db.192.253.253
;
; 这些是我们的辅区文件
;
secondary ora.com                   198.112.208.25 slave/bak.ora.com
secondary 208.112.198.in-addr.arpa 198.112.208.25 slave/bak.198.112.208
```

下面是 BIND 8 格式的同样的配置文件：

```
options { directory "/var/named"; };
//
// 这些文件不专属于任何区
//
zone "." {
        type hint;
        file "db.cache";
};
zone "0.0.127.in-addr.arpa" {
        type master;
        file "db.127.0.0";
};
```

```
//
// 这些是我们的主区文件
//
zone "movie.edu" {
        type master;
        file "primary/db.movie.edu";
};
zone "249.249.192.in-addr.arpa" {
        type master;
        file "primary/db.192.249.249";
};
zone "253.253.192.in-addr.arpa" {
        type master;
        file "primary/db.192.253.253";
};
//
// 这些是我们的辅区文件
//
zone "ora.com" {
        type slave;
        file "slave/bak.ora.com";
        masters { 198.112.208.25; };
};
zone "208.112.192.in-addr.arpa" {
        type slave;
        file "slave/bak.198.112.208";
        masters { 198.112.208.25; };
};
```

这种划分方法的一种演化是将配置文件分为三个文件：主文件、包括所有 *primary* 条目的文件, 以及包括所有 *secondary* 条目的文件。下面是采用这个方案的 BIND 4 的主配置文件：

```
directory /var/named
;
; 这些文件不专属于任何区
;
cache    .                          db.cache
primary  0.0.127.in-addr.arpa       db.127.0.0
;
include  named.boot.primary
include  named.boot.slave
```

下面是 *named.boot.primary*（BIND 4）文件：

```
;
; 这些是我们的主区文件
;
primary  movie.edu                  primary/db.movie.edu
primary  249.249.192.in-addr.arpa   primary/db.192.249.249
```

```
primary  253.253.192.in-addr.arpa  primary/db.192.253.253
```

下面是 *named.boot.slave*（BIND 4）文件：

```
;
; 这些是我们的辅区文件
;
secondary ora.com                  198.112.208.25 slave/bak.ora.com
secondary 208.112.198.in-addr.arpa 198.112.208.25 slave/bak.198.112.208
```

下面是 BIND 8 或 9 格式的上述两个文件：

```
options { directory "/var/named"; };
//
// 这些文件不专属于任何区
//
zone "." {
        type hint;
        file "db.cache";
};
zone "0.0.127.in-addr.arpa" {
        type master;
        file "db.127.0.0";
};

include "named.conf.primary";
include "named.conf.slave";
```

下面是 *named.conf.primary*（BIND 8 或 9）文件：

```
//
// 这是我们的主区文件
//
zone "movie.edu" {
        type master;
        file "primary/db.movie.edu";
};
zone "249.249.192.in-addr.arpa" {
        type master;
        file "primary/db.192.249.249";
};
zone "253.253.192.in-addr.arpa" {
        type master;
        file "primary/db.192.253.253";
};
```

下面是 *named.conf.slave*（BIND 8 或 9）文件：

```
//
// 这是我们的辅区文件
```

```
//
zone "ora.com" {
        type slave;
        file "slave/bak.ora.com";
        masters { 198.112.208.25; };
};
zone "208.112.192.in-addr.arpa" {
        type slave;
        file "slave/bak.198.112.208";
        masters { 198.112.208.25; };
};
```

你也可能认为这种组织方法会更好：添加一条新的转到*primary*目录的*directory*指令，从而将带*primary*指令的配置文件放在这个子目录中，还要把原来每个文件名中的*primary/*删除，这是因为名字服务器现在就运行在这个新的子目录中了。同时也要对带*secondary*行的配置文件进行类似的处理。不幸的是，这样做行不通了。BIND 8和9名字服务器只让你定义一个工作目录。BIND 4名字服务器允许你用多个*directory*指令来定义多个工作命令，但与其说这是个特性，不如说是个失误。因为当名字服务器不停地在各个目录间转来转去时，一切变得让人很迷糊，比如，如果备份区数据文件是在名字服务器最后一次转到的那个目录中，而当名字服务器重新加载时，如果主配置文件不是在服务器的启动目录中的话，它就可能找不到主配置文件（如果配置文件指定的是相对路径的话）。

在区数据文件中更改起点

使用BIND时，区数据文件默认的起点（origin）在BIND 4中是*named.boot*文件中*primary*或*secondary*指令的第二个字段，在BIND 8或9中是*named.conf*文件中*zone*语句的第二个字段。起点是一个域名，它会被自动地附加到文件中所有不以“.”结尾的名字后面。可以在区数据文件中用$ORIGIN控制语句来改变起点。在区数据文件中，$ORIGIN后面跟一个域名。（如果你使用的是完整域名的话，千万别忘了结尾的“.”！）自此以后，所有不以“.”结尾的名字后面都会加上这个新的起点。如果你的区（譬如说是*movie.edu*）有多个子域，你就可以使用$ORIGIN语句来重新设置起点，简化区数据文件。例如：

```
$ORIGIN classics.movie.edu.
maltese       IN  A  192.253.253.100
casablanca    IN  A  192.253.253.101
$ORIGIN comedy.movie.edu.
mash          IN  A  192.253.253.200
twins         IN  A  192.253.253.201
```

我们将在第九章中更加深入地讨论创建子域的内容。

包括其他区数据文件

如果按上述方法来划分你的区，那么你会发现将每个子域的记录保存在不同的文件中会更加方便。而 $INCLUDE 控制语句会让你达到这个目的：

```
$ORIGIN classics.movie.edu.
$INCLUDE db.classics.movie.edu

$ORIGIN comedy.movie.edu.
$INCLUDE db.comedy.movie.edu
```

想再进一步简化这些文件的话，你可以在一行上指定被包括的文件和新的起点：

```
$INCLUDE db.classics.movie.edu classics.movie.edu.
$INCLUDE db.comedy.movie.edu   comedy.movie.edu.
```

当你在一行上指定了起点和被包括的文件时，这个对起点的更改只适用于你所包括的那个文件。例如，*comedy.movie.edu*这个起点只适用于*db.comedy.movie.edu*文件中的名字。在把 *db.comedy.movie.edu* 文件包括进来以后，起点就又重新变成 $INCLUDE以前的那个，即使在*db.comedy.movie.edu*文件中也有一个$ORIGIN语句。

在 BIND 8 和 9 中改变系统文件的位置

BIND 8 和 9 允许你改变下列系统文件的名字和位置：*named.pid*、*named-xfer*、*named_dump.db*和*named.stats*。大多数人没有必要使用这个特性，不要只是因为你可以改变这些文件的名字和位置就认为你必须要改变它们。

如果为了安全起见，你改变了那些由名字服务器生成的文件（*named.pid*、*named_dump.db*或*named.stats*）的位置，那么你应该选择一个不是谁都可以写的目录来保存它们。虽然我们还不知道有非法侵入是因为这些文件引起的，但是为了保险起见，你还是应该遵循这条原则。

*named.pid*的完整路径通常是*/etc/named.pid*或者*/var/run/named.pid*。你要改变这个文件默认位置的一个原因就是你在一台主机上运行了多个名字服务器。（老天！怎

么会有人这样做？）第十章给出了一个在一台主机上运行两个名字服务器的例子。你可以在配置文件中为每个服务器指定不同的 *named.pid* 文件：

```
options { pid-file "server1.pid"; };
```

named-xfer 的路径通常是 */etc/ named-xfer* 或 */usr/sbin/ named-xfer*。slave 服务器在从远程服务器到本地服务器的区传送中会用到这个文件。你可以更改这个文件默认位置的一个原因是，希望在本地目录中编译连接和测试一个新版本的 BIND，测试版的 *named* 就可以配置成使用它自己的 *named-xfer*：

```
options { named-xfer "/home/rudy/named/named-xfer"; };
```

因为 BIND 9 不使用 *named-xfer*，所以在 BIND 9 中当然也就不需要使用这个子语句了。

在你让名字服务器转储其数据库时，名字服务器会把 *named_dump.db* 文件写到它的当前目录下（BIND 8 或 9）。下面是一个如何修改转储文件位置的例子：

```
options { dump-file "/home/rudy/named/named_dump.db"; };
```

在你让名字服务器转储其统计数据时，名字服务器会把 *named.stats* 文件写到它的当前目录下（BIND 8 或 9.1.0 及其后续版本）。下面是一个如何修改该文件位置的示例：

```
options { statistics-file "/home/rudy/named/named.stats"; };
```

BIND 8 和 9 中的日志

BIND 4 有广泛的日志系统，可以将信息写到一个调试文件中，还可以将信息发送给 *syslog*。不过 BIND 4 允许你对日志过程进行的控制很有限，你只能够将调试设定到某一等级，仅此而已。BIND 8 和 9 的日志系统与 BIND 4 的相同，但是这两种 BIND 能使你对日志系统进行 BIND 4 所不能进行的控制。

但是这种控制是要付出代价的，在你能有效配置这个子系统前，需要学习很多东西。如果你没有时间来研究日志，那么就使用默认的配置，过后再来看这部分内容吧。你们大部分人都不必改变默认的日志设置。

在日志中主要有两个概念：通道（channel）和类别（category）。通道指定了应该向那里发送日志数据：是发送给 *syslog*，还是写在一个文件里，或是发送给 *named* 的标准错误输出，还是发送到位存储桶（bit bucket）。类别则规定了哪些数据需要记录。在 BIND 源码中，名字服务器记录的大多数消息是按照与它们相关的代码的功能来分类的。举个例子来说，一个由 BIND 处理动态更新的那部分产生的消息就应该在 *update* 类中。等一会我们会给出类别的列表。

每种类别的数据可以被发送给一个或多个通道。在图 7-1 中，所有的查询被记录在一个文件中，而统计数据则既被记录到另一个文件中，也被记录到 *syslog* 中。

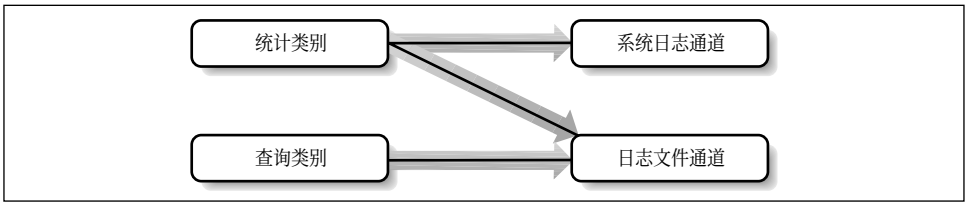


图 7-1　将不同类别的数据记录到不同的通道

通道允许你根据不同的级别对消息进行过滤。下面是不同的级别列表，按照严重性递减的顺序：

```
critical
error
warning
notice
info
debug [level]
dynamic
```

最前面的五个级别（*critical*、*error*、*warning*、*notice* 和 *info*）是我们很熟悉的 *syslog* 使用的五个级别。其他两个（*debug* 和 *dynamic*）则是 BIND 8 和 9 特有的。

debug 是你可以指定的对名字服务器进行调试的级别。如果你不指定调试级别，那么就假定为 1。如果你指定了调试的级别，当名字服务器的调试功能打开时，你就会看到该级别的消息。（例如，如果你指定了“ debug 3 ”，那么当你只是向名字服务器发送了一个跟踪命令时，就会看到第三级调试消息。）如果你规定的是 *dynamic* 这个级别，那么名字服务器就会记录所有与调试级别匹配的日志消息。（例如，如果你发送给名字服务器一个跟踪命令，它就记录从第一级起的调试消息。如果你发送

三个跟踪命令给名字服务器，那它就会记录所有从第一级到第三级的调试消息。）默认的级别是 *info*，也就是说，除非你指定了等级，否则看不到任何调试消息。

注意：你可以将一个通道配置成把调试消息和*syslog*消息同时记录到一个文件中。不过反过来就不行了，你无法配置一个通道用*syslog*来同时记录调试消息和*syslog*消息，调试消息不能发送到*syslog*。

让我们配置几个通道来告诉你这是如何工作的。第一个通道是到*syslog*，并会使用*daemon*这个工具来记录，将发送*info*及其以上级别的消息。第二个通道是到一个文件，记录所有级别的调试消息和*syslog*消息。下面是 BIND 8 或 9 配置文件中的*logging*语句：

```
logging {
  channel my_syslog {
     syslog daemon;
     // 调试消息不会发送到 syslog，所以不必将严重性设置为 debug 或 dynamic；
     // 使用最低的 syslog 级别：info。
     severity info;
  };
  channel my_file {
     file "log.msgs";
     // 设置 severity 为 dynamic 以观察所有的调试消息
     severity dynamic;
  };
};
```

现在已经配置了几个通道，我们必须很确切地告诉名字服务器需要向这些通道发送哪些消息。让我们来实现图 7-1 所示的那样，把统计数据发送给*syslog*并写入一个文件，而所有的查询都写入文件中。类别的指定是*logging*语句的一部分，所以我们就在上面提到的那个*logging*语句上进一步修改：

```
logging {
  channel my_syslog {
     syslog daemon;
     severity info;
  };
  channel my_file {
     file "log.msgs";
     severity dynamic;
  };

  category statistics { my_syslog; my_file; };
  category queries { my_file; };
```

```
};
```

在你的配置文件中加入上述*logging*语句，启动你的名字服务器，然后向它发送几个查询。但是*log.msgs*中没有写入任何消息！（如果你等待足够长的时间，名字服务器的统计数据就会出现在*log.msgs*中。）你本来希望这些查询会被记录下来。唉，你还必须打开名字服务器的调试功能才能记录这些查询：

```
#  ndc trace
```

现在，如果向你的名字服务器发送一些查询，它们就会被记录到*log.msgs*中了。不过在名字服务器的当前目录中找一找，你会发现一个叫做*named.run*的新文件。所有的其他调试信息会记录在里面。但是你并不想要这些其他的调试信息，只想要统计数据和查询。那么你要怎样才能避免生成*named.run*呢？

还有一个特殊的类别我们没有告诉你：*default*。如果对于某个类别你没有指定任何通道，那么BIND将会把这些消息发送到*default*类别所分配的通道。我们来更改一下*default*类别，丢弃所有的日志消息（为了这个目的有一个名为*null*的通道）：

```
logging {
  channel my_syslog {
     syslog daemon;
     severity info;
  };
  channel my_file {
     file "log.msgs";
     severity dynamic;
  };

  category default { null; };
  category statistics { my_syslog; my_file; };
  category queries { my_file; };
};
```

现在，启动你的服务器，把调试级别设置为1，再发送一些查询。这些查询就会被记录在*log.msgs*中，虽然还是创建了*named.run*，但是其内容为空。好极了！我们终于达到目的了。

又过了几天，你的一个同事注意到名字服务器向*syslog*发送的消息和过去相比少了很多。事实上*syslog*记录的惟一消息就是统计数据消息。你的同事所注意到是有关区传送的消息不见了。到底怎么回事呢？

是这样的，默认的 *default* 类别被设置成把消息同时发送给 *syslog* 和调试文件（*named.run*）。当 *default* 类别被配置为 *null* 通道后，同时也就关闭了其他的 *syslog* 消息。下面是我们为这个问题所使用的配置：

```
category default { my_syslog; };
```

这会把 *syslog* 消息发送给 *syslog*，但是不会把调试或 *syslog* 消息写到文件里。

请记住，我们说过你必须经过一段时间的试验，日志系统才会完全按照你所需要的方式运行。我们希望这个例子给你提供了一些可能遇到的问题。现在，让我们看看有关日志的更详细的说明。

Logging 语句

下面我们来讲讲 *logging* 语句的语法。它的语法还是有点吓人的。我们一边看些例子，一边解释每个子句分别是什么意思：

```
logging {
  [ channel channel_name {
    ( file path_name
       [ versions ( number | unlimited ) ]
       [ size size_spec ]
     | syslog ( kern | user | mail | daemon | auth | syslog | lpr |
                news | uucp | cron | authpriv | ftp |
                local0 | local1 | local2 | local3 |
                local4 | local5 | local6 | local7 )
     | stderr
     | null );
    [ severity ( critical | error | warning | notice |
                 info  | debug [ level ] | dynamic ); ]
    [ print-category yes_or_no; ]
    [ print-severity yes_or_no; ]
    [ print-time yes_or_no; ]
  }; ]

  [ category category_name {
    channel_name; [ channel_name; ... ]
  }; ]
  ...
};
```

下面是默认的通道。即使你不想要，名字服务器也还是会创建这些通道的。你不能重定义这些通道，你只能再添加新的通道：

```
channel default_syslog {
    syslog daemon;          // 发送到 syslog 的守护进程
    severity info;          // 只发送严重性为 info 或更高级别的数据
};

channel default_debug {
    file "named.run";       // 写到工作目录下的 named.run 文件中
    severity dynamic;       // 以服务器当前的 debug 级别记录日志
};

channel default_stderr {  // 写到 stderr 中
    stderr;                 // 仅有 BIND 9 允许你定义自己的 stderr 通道，尽管
                            // BIND 8 有内置的 default_stderr 通道。
    severity info;          // 仅发送严重性为 info 或级别更高的数据
};

channel null {
    null;                   // 扔掉发送到这个通道的任何数据
};
```

如果你没有给 *default*、*panic*、*packet* 和 *eventlib* 类别分配通道，BIND 8 名字服务器会默认地为它们分配这些通道：

```
logging {
    category default { default_syslog; default_debug; };
    category panic { default_syslog; default_stderr; };
    category packet { default_debug; };
    category eventlib { default_debug; };
};
```

BIND 9 名字服务器会使用这样的默认 logging 语句：

```
logging {
     category default {
          default_syslog;
          default_debug;
     };
};
```

正如前面我们所提到的那样，*default* 类别会把日志同时记录到 *syslog* 和调试文件（默认的是 *named.run*）当中。也就是说，所有的 *info* 以及更高级别的 *syslog* 消息都会被送到 *syslog*，而当调试打开时，*syslog* 消息和调试消息会被写到 *named.run* 当中。这或多或少有点像 BIND 4。

有关通道的详细说明

通道可以定义成到一个文件、*syslog* 或是空的。

文件通道

如果通道定义成通向一个文件，你必须指定文件的路径名。可选地，你还可以指定允许同时存在多少个版本的该文件，以及文件最多能有多大。

如果你指定为可能有三个版本，BIND 8 或 9 会保存 *file*、*file.0*、*file.1* 和 *file.2*。在名字服务器启动或重新加载后，它会将 *file.1* 变成 *file.2*，*file.0* 变成 *file.1*，*file* 变成 *file.0*，并且创建一个新的 *file*。如果你不限制版本数量，BIND 会保存 99 个版本。

如果你指定了文件大小的上限，名字服务器会在文件达到指定大小时停止写入该文件。与 *versions* 子语句（在上一段中我们提到的）不同，当达到指定大小时，名字服务器不会循环使用，而是再打开一个新文件。名字服务器只会停止向该文件写入。如果你不指定文件大小，文件会无限增大。

下面这个例子是一个使用了 *versions* 和 *size* 子语句的文件通道：

```
logging{
  channel my_file {
     file "log.msgs" versions 3 size 10k;
     severity dynamic;
  };
};
```

正如例子中所示的，size 语句中可以使用表示大小的单位。*K* 或 *k* 代表千字节，*M* 或 *m* 代表兆字节，*G* 或 *g* 代表千兆字节。

如果你想要看调试信息的话，将等级指定为 *debug* 或 *dynamic* 是很重要的。默认的等级是 *info*，它只显示 *syslog* 消息。

syslog 通道

如果一个通道定义成通向 *syslog*，你可以将工具指定为下列中的任意一个：*kern*、*user*、*mail*、*daemon*、*auth*、*syslog*、*lpr*、*news*、*uucp*、*cron*、*authpriv*、*ftp*、*local0*、*local1*、*local2*、*local3*、*local4*、*local5*、*local6* 和 *local7*。默认值是 *daemon*，我们建议你使用默认值。

下面这个例子是一个使用工具 *local0* 而不是 *daemon* 的 *syslog* 通道：

```
logging {
```

```
    channel my_syslog {
        syslog local0;          // 发送到 syslog 的 local0 工具
        severity info;          // 仅发送严重性为 info 或更高级别的数据
    };
};
```

stderr 通道

有一个预定义的通道称为*default_stderr*，你可以用它来将你所希望的任何消息写到名字服务器的 stderr（标准错误输出）文件描述符中。在 BIND 8 中，你不能将它配置成其他任何的文件描述符。而在 BIND 9 中你可以。

null 通道

有一个预定义的通道称为 null（空），你可以用它来丢弃那些你不想要的消息。

所有通道的数据格式

BIND 8 和 9 记录日志的工具还允许你控制消息的格式。你可以为消息添加时间戳、类别或是等级。

下面是一个添加了所有这些额外项的调试消息的例子：

```
01-Feb-1998 13:19:18.889 config: debug 1: source = db.127.0.0
```

这个消息的类别是 *config*，严重性是 *debug*，级别是 1。

下面是一个包括三个附加项的通道配置的例子：

```
logging {
  channel my_file {
     file "log.msgs";
     severity debug;
     print-category yes;
     print-severity yes;
     print-time yes;
  };
};
```

给发送到 *syslog* 通道的消息加时间戳没有什么意义，因为 *syslog* 会自己添加时间和日期的。

有关类别的详细说明

BIND 8和9都有很多类别。不幸的是，BIND 8和9的类别并不相同，在此，我们都列出来供你参考。与其让你试着去配置出你想要看见的样子，我们还是建议你先让你的名字服务器显示出所有的日志消息，以及它们的类别和等级，再挑选出你想要的那些。在介绍完各种类别之后，我们再告诉你如何来做。

BIND 8类别

default

如果你不为某个类别指定通道，那么就使用*default*类别。从这种意义上来说，*default*是所有类别的同义词。不过，还是有一些消息是不属于任何类别的。所以，即使你分别为每个类别都指定了通道，还是应该为了所有不属于任何类别的消息，给*default*类指定一个通道。

如果你没有为*default*类别指定通道，系统就会给你指定一个：

```
category default { default_syslog; default_debug; };
```

cname

CNAME错误（例如，“ ... 有CNAME和其他数据 ”）。

config

高级配置文件处理。

db

数据库操作。

eventlib

系统事件，必须指向一个文件通道。默认值是：

```
category eventlib { default_debug; };
```

insist

内部一致性检查失败。

lame-servers

发现错误授权。

load

区加载消息。

maintenance

定期维护事件（例如，系统查询）。

ncache

否定缓存事件。

notify

异步区变动通知。

os

操作系统的问题。

packet

解码包的接收和发送，必须指向一个文件通道。默认值是：

```
category packet { default_debug; };
```

panic

导致服务器关闭的问题。这类问题既记录在 *panic* 类别中，又记录在它们本身所属的类别中。默认值是：

```
category panic { default_syslog; default_stderr; };
```

parser

低级配置文件处理。

queries

类似 BIND 4 的查询日志。

response-checks

格式错误的回答，无关的附加信息，等等。

security

认可 / 非认可的请求。

statistics

活动的定期报告。

update

动态更新事件。

xfer-in

从远程名字服务器到本地名字服务器的区传送。

xfer-out

从本地名字服务器到远程名字服务器的区传送。

BIND 9 类别

default

同 BIND 8 一样，BIND 9 的 *default* 类别匹配所有未明确指定通道的类别。不过同 BIND 8 不一样的是，BIND 9 的 *default* 类别不匹配不属于任何类别的消息。这些不属于任何类别的消息属于下面列出的这些类别。

general

general 类别包括所有未明确分类的 BIND 消息。

client

处理客户端请求。

config

配置文件分析和处理。

database

同 BIND 内部数据库相关的消息，用来存储区数据和缓存记录。

dnssec

处理 DNSSEC 签名的响应。

lame-servers

发现错误授权（在 BIND 9.1.0 中重新加入的；在此之前，这个消息是被记录到 *resolver* 中的）。

network

网络操作。

notify

异步区变动通知。

queries

类似 BIND 8 的查询日志（在 BIND 9.1.0 中增加的）。

resolver

名字解析，包括对来自解析器的递归查询的处理。

security

认可 / 非认可的请求。

update

动态更新事件。

xfer-in

从远程名字服务器到本地名字服务器的区传送。

xfer-out

从本地名字服务器到远程名字服务器的区传送。

查看所有类别的消息

在你开始修改日志之前，最好是将你的名字服务器配置成将所有的消息记录到一个文件中，包括类别和等级，然后再从中挑出你感兴趣的消息。

前面我们列出了默认配置了的类别。对于 BIND 8 来说，是这样的：

```
logging {
    category default { default_syslog; default_debug; };
    category panic { default_syslog; default_stderr; };
    category packet { default_debug; };
    category eventlib { default_debug; };
};
```

对 BIND 9 来说，是这样的：

```
logging {
    category default { default_syslog; default_debug; };
};
```

默认地，写到*default_debug*通道中的消息不包括类别和等级。为了能看到所有的日志消息以及它们的类别和等级，你必须要自己配置每一个类别。

下面就是一个这样的 BIND 8 的 *logging* 语句：

```
logging {
  channel my_file {
```

```
        file "log.msgs";
        severity dynamic;
        print-category yes;
        print-severity yes;
    };

    category default  { default_syslog; my_file; };
    category panic    { default_syslog; my_file; };
    category packet   { my_file; };
    category eventlib { my_file; };
    category queries  { my_file; };
};
```

（BIND 9的 *logging* 语句中没有 *panic*、*packet* 或 *eventlib* 类别。）

注意，我们将每一个类别都定义为包括通道 *my_file*。我们还添加了一个类别 *queries*，它并不在前面默认的那个 *logging* 语句中。如果你不配置 *queries* 类别，它是不会显示出来的。

启动你的名字服务器，将调试打开成第一级。然后你会在 *log.msgs* 中看到如下所示的消息：

```
queries: info: XX /192.253.253.4/foo.movie.edu/A
default: debug 1: req: nlookup(foo.movie.edu) id 4 type=1 class=1
default: debug 1: req: found 'foo.movie.edu' as 'foo.movie.edu' (cname=0)
default: debug 1: ns_req: answer -> [192.253.253.4].2338 fd=20 id=4 size=87
```

一旦你决定了你所感兴趣的消息，就将服务器配置成只记录这些消息。

使一切平稳运转

维护的一个重要部分就是能在真正出现问题之前，察觉到有东西出错了。如果你能及早发现问题，可能修复起来就会容易得多。

这里可不是排错，后面我们用了整整一章来讲排错，就把它看成是“ 预排错 ”吧。如果忽视维护，在问题变得复杂之后，你就不得不进行排错了，你需要根据症状来判断出问题到底是出在哪儿的。

接下来的两节讲述预防性的维护：定期查看 *syslog* 文件以及 BIND 名字服务器统计数据，看是否有问题出现。可以把它看成是给名字服务器检查身体。

常见的 syslog 消息

named 能发出大量的 *syslog* 消息。而实际上，你只会见到其中的一部分。在此我们将谈谈最为常见的 *syslog* 消息，不包括区数据文件中的语法错误报告。

每次启动 *named* 时，它会以优先级 LOG_NOTICE 发送一个消息。对 BIND 8 名字服务器来说，看起来如下所示：

```
Jan 10 20:48:32 terminator named[3221]: starting.  named 8.2.3 Tue May 16 09:39:40
MDT 2000 ^Icricket@huskymo.boulder.acmebw.com:/usr/local/src/bind-8.2.3/src/bin/
named
```

对 BIND 9 来说就短多了：

```
Jul 27 16:18:41 terminator named[7045]: starting BIND 9.1.0
```

这个消息记录了 *named* 在此时启动，告诉你正在运行的 BIND 的版本，以及谁在何处编译的它（对 BIND 8 而言）。当然，这没有什么要特别注意的。不过，如果你不太确信你的操作系统所支持的 BIND 版本，这倒是个该看一看的地方。（较早版本的 BIND 写的是 “ restarted ” 而不是 “ starting ”。）

每次你给名字服务器发送重新加载命令时，BIND 8 名字服务器都会以 LOG_NOTICE 优先级发送下面这个消息：

```
Jan 10 20:50:16 terminator named[3221]: reloading nameserver
```

BIND 9 名字服务器日志：

```
Jul 27 16:27:45 terminator named[7047]: loading configuration from '/etc/named.
conf
```

这个消息只是告诉你 *named* 在此时重新加载了它的数据库（是由重新加载命令引起的）。这也没有什么要特别注意的地方。当你在跟踪一个坏的资源记录在你的区数据中已经呆了多久，或跟踪由于在更新过程中的错误一整个区已经丢失了多久时，你可能会对这个消息感兴趣。

在你的名字服务器启动后不久，你可能还会看到这样一条消息：

```
Jan 10 20:50:20 terminator named[3221]: cannot set resource limits on
                this system
```

这意味着你的名字服务器认为你的操作系统不支持 *getrlimit()*和 *setrlimit()*系统调用，这两个函数是你试图在 BIND 8 或 9 名字服务器上定义 *coresize*、*datasize*、*stacksize* 或 *files* 时所使用的。你是否真的在配置文件中使用了这些子语句并没有什么关系，BIND 总是要显示这些消息的。如果你没有使用这些子语句，就忽略这个消息。如果你使用了，而你又认为你的操作系统是支持 *getrlimit()*和 *setrlimit()*的，你就要定义 HAVE_GETRUSAGE，再重新编译 BIND。这个消息的优先级为 LOG_INFO。

如果你是在有很多网络接口（特别是虚拟网络接口）的主机上运行名字服务器的，在启动后不久，甚至是在你的名字服务器已经运行了一段时间之后，你还会看到这样的消息：

```
Jan 10 20:50:31 terminator named[3221]: fcntl(dfd, F_DUPFD, 20): Too
                many open files
Jan 10 20:50:31 terminator named[3221]: fcntl(sfd, F_DUPFD, 20): Too
                many open files
```

这意味着 BIND 用完了所有的文件描述符。BIND 使用了相当多数量的文件描述符：每个它监听的网络接口两个（UDP 一个，TCP 一个），而且每个打开的区数据文件也要有一个。如果这超出了你的操作系统对进程所做的限制，BIND 就不能使用更多的文件描述符，你就会看到上面这个消息了。该消息的优先级取决于是 BIND 的哪部分不能获取文件描述符：该子系统越重要，该消息的优先级也就越高。

接下来，你可以让 BIND 少使用一点文件描述符，或者提高操作系统对 BIND 能使用文件描述符的数量所做的限制：

- 如果你不需要 BIND 监听所有的网络接口（特别是虚拟网络接口），用 *listen-on* 子语句来配置 BIND，让它只监听需要监听的接口。关于 *listen-on* 的语法的详细信息，请参阅第十章。
- 如果你的操作系统支持 *getrlimit()*和 *setrlimit()*（就像前面讲过的那样），用 *files* 子语句来将你的名字服务器配置成使用更多数量的文件。关于 *files* 子语句使用的详细信息，请参阅第十章。
- 如果你的操作系统对打开文件数量的限制过于严格，那么在启动 *named* 之前，用 *unlimit* 命令来提高这个限制。

每次BIND 8名字服务器加载一个区的时候, 它会发送一个LOG_INFO级别的消息：

```
Jan 10 21:49:50 terminator named[3221]: master zone "movie.edu" (IN)
                Loaded (serial 1996011000)
```

（BIND 4.9 名字服务器称之为 “ primary zone ”，而不是 “ master zone ”。）这条消息告诉你名字服务器是何时加载该区、区的类（在这里是IN），以及该区的SOA 记录中的序列号。BIND 9 名字服务器（比如，BIND 9.1.0）不会告诉你何时加载了区。

大约每隔一小时，BIND 8 名字服务器会发送一个 LOG_INFO 级别的消息，记录下当前的统计数据：

```
Feb 18 14:09:02 terminator named[3565]: USAGE 824681342 824600158
                CPU=13.01u/3.26s CHILDCPU=9.99u/12.71s
Feb 18 14:09:02 terminator named[3565]: NSTATS 824681342 824600158
                A=4 PTR=2
Feb 18 14:09:02 terminator named[3565]: XSTATS 824681342 824600158
                RQ=6 RR=2 RIQ=0 RNXD=0 RFwdQ=0 RFwdR=0 RDupQ=0 RDupR=0
                RFail=0 RFErr=0 RErr=0 RTCP=0 RAXFR=0 RLame=0 Ropts=0
                SSysQ=2 SAns=6 SFwdQ=0 SFwdR=0 SDupQ=5 SFail=0 SFErr=0
                SErr=0 RNotNsQ=6 SNaAns=2 SNXD=1
```

（在 BIND 4.9 到 BIND 4.9.3 中也有这一特性，不过在 4.9.4 服务器中关闭了。BIND 9 不支持该特性，比如 BIND 9.1.0。）每条消息中的头两个数是时间。如果你将第二个数减去第一个数, 就会得到你的服务器已经运行的秒数。(你可能希望名字服务器来帮你完成这一工作。）CPU 项告诉你你的名字服务器处于用户态（13.01 秒）和系统态（3.26 秒）的时间。接下来，它会告诉你子进程的这些统计数据。NSTATS 消息列出了你的服务器收到的查询类型及其次数。XSTATS 消息列出了其他一些统计数据。我们会在本章后面更详细地解释 NSTATS 和 XSTATS 中的统计数据。

如果 BIND 4.9.4 及其后续版本（但不包括 BIND 9，比如 BIND 9.1.0，它尚未实现名字检查）发现了一个不遵循 RFC 952 的名字，它就会记录一条 *syslog* 错误：

```
Jul 24 20:56:26 terminator named[1496]: owner name "ID_4.movie.edu IN"
                                        (primary) is invalid - rejecting
```

这条消息的级别是 LOG_INFO。关于主机的命名规则请参见第四章。

还有一条 LOG_INFO 级别的 *syslog* 消息是关于区数据的警告消息：

```
Jan 10 20:48:38 terminator named[3221]: terminator2 has CNAME
                and other data (invalid)
```

这条消息是说你的区数据出了问题。例如，你可能有像下面这样的一些记录：

```
terminator2  IN  CNAME t2
terminator2  IN  MX    10 t2
t2           IN  A     192.249.249.10
t2           IN  MX    10 t2
```

*terminator2*的MX记录是不正确的，会导致出现上面列出的消息。*terminator2*是*t2*的别名，*t2*才是规范名。正如我们前面所说到的那样，当名字服务器查找一个名字时发现了一个CNAME记录，它会用规范名替换原来的名字，然后再试图去查找这个规范名。因此，当名字服务器查找*terminator2*的MX数据时，它找到一个CNAME记录，然后查找*t2*的MX记录。由于服务器会沿着*terminator2*的CNAME记录查找，它永远也不可能用到*terminator2*的MX记录，实际上，这个记录是不合法的。换句话说，一个主机的所有资源记录都应该使用规范名，在该用规范名的地方使用别名就会出错。

我们意识到有点重复了，但是BIND 9直到9.1.0之前都不能检查出这个错误。

下面这条消息表明当BIND 4或8的辅名字服务器试图进行区传送时，不能访问任何主名字服务器：

```
Jan 10 20:52:42 wormhole named[2813]: zoneref: Masters for
                secondary zone "movie.edu" unreachable
```

BIND 9的辅名字服务器中该消息会是这样的：

```
Jul 27 16:50:55 terminator named[7174]: refresh_callback: zone movie.edu/IN:
failure for 10.0.0.1#53: timed out
```

这条消息的级别在BIND 4或8上是LOG_NOTICE，在BIND 9上是LOG_INFO，只有在第一次区传送失败时才发送。当最终区传送成功时，4.9及其后续版本的名字服务器会再发送一个*syslog*消息，告诉你区数据已经传送了。当这条消息刚出现时，你不必立即采取行动。名字服务器会根据SOA记录中的重试时间，继续试着传送区的。过了几天（或期满时间过半）之后，你要检查一下看服务器是否能够传送该区了。在那些当区传送时并不发送*syslog*消息的服务器上，你可以根据备份文件的时间戳来检查，确认区是否已经传送了。当区传送成功时，就会创建新的备份区数据

文件。当名字服务器发现区数据是最新的时，它会“ touche ”备份文件（按照Unix *touch* 命令的方式）。在上述两种情况中，备份文件的时间戳都会被更新，所以在辅名字服务器上运行 *ls −l /usr/local/named/db** 命令。这就能告诉你每个区与它的主名字服务器最后一次同步的时间。我们会在第十四章中讲到如果对辅名字服务器传送区失败进行排错。

如果你查看 BIND 4.9 或其后续版本主名字服务器上的 *syslog* 消息，当辅名字服务器获取新的区数据或当使用 *nslookup* 这样的工具传送区的时候，你会看到一个 LOG_ INFO 级别的 *syslog* 消息：

```
Mar  7 07:30:04 terminator named[3977]: approved AXFR from
                            [192.249.249.1].2253 for "movie.edu"
```

同样地，BIND 9 在这种情况下不会记录任何东西，正如 BIND 9.1.0。

如果你用了 BIND 4 的 *xfrnets* 配置文件指令或 BIND 8 的 *allow-transfer* 子语句（在第十章中会讲到）来限制哪些服务器能加载区，那么这条消息中的“ approved ”（认可）就会换成“ unapproved ”（非认可）。BIND 9 名字服务器会报告：

```
Jul 27 16:59:26 terminator named[7174]: client 192.249.249.1#1386: zone transfer
denied
```

只有在你捕获 LOG_INFO 等级的 *syslog* 消息时，才会看到这条 *syslog* 消息：

```
Jan 10 20:52:42 wormhole named[2813]: Malformed response
                from 192.1.1.1
```

通常，这条消息是说名字服务器中的某些小错误使它发送了错误的响应包。这个错误很有可能出现在远程名字服务器（192.1.1.1）上，而不是本地服务器（wormhole）上。分析此类错误，要从网络追踪（network trace）中捕获该响应包，再解码。手工解码DNS包已经超出了本书的范围，所以我们不会详细地讲解它。当响应包说它的回答部分中包含多个回答（比如，四个地址资源记录），而回答部分里只含有一个回答时，你会看到这种类型的错误。惟一可做的就是通过电子邮件（假设你能通过查找主机的地址获得它的名字）通知那台犯错主机的管理员。如果下层网络改变（破坏）了 UDP 响应包的某些地方，你也会看到这样的消息。因为对 UDP 包的校验和是可选的，所以这样的错误可能不会在低层被发现。

当你试图将别的记录加入到你的区数据文件中时，BIND 4.9或8的*named*会将这样的消息记录到日志中：

```
Jun 13 08:02:03 terminator named[2657]: db.movie.edu:28: data "foo.bar.edu"
                          outside zone "movie.edu" (ignored)
```

BIND 9 的 *named* 会记录：

```
Jul 27 17:07:01 terminator named[7174]: dns_master_load: db.movie.edu:28: ignoring
out-of-zone data
```

例如，如果我们试图使用这样的区数据：

```
robocop       IN A  192.249.249.2
terminator    IN A  192.249.249.3

; 添加这项到名字服务器的缓存中
foo.bar.edu.  IN A  10.0.7.13
```

我们会将 *bar.edu* 区的数据添加到我们的 *movie.edu* 区数据文件中。4.8.3 名字服务器会盲目地将 *foo.bar.edu* 添加到它的缓存中，不会检查 *db.movie.edu* 文件中的数据是否都是在 *movie.edu* 区中。不过，你不能再愚弄 4.9 版本以后的服务器了。这条 *syslog* 消息的级别是 LOG_INFO。

在本书前面我们说过，你不能在一个资源记录的数据部分中使用 CNAME。BIND 4.9 和 8 能捕捉到这样的误用：

```
Jun 13 08:21:04 terminator named[2699]: "movie.edu IN NS" points to a
                                        CNAME (dh.movie.edu)
```

BIND 9 不能发现这样的误用，正如 BIND 9.1.0。

下面是一个错误资源记录的例子：

```
@                       NS       terminator.movie.edu.
                        NS       dh.movie.edu.
terminator.movie.edu.   IN A     192.249.249.3
diehard.movie.edu.      IN A     192.249.249.4
dh                      IN CNAME diehard
```

第二个 NS 记录应该列出 *diehard.movie.edu* 而不是 *dh.movie.edu*。你的名字服务器启动时，不会立即出现这个 *syslog* 消息。

注意：只有在查找这个错误数据时，你才会看到这条 *syslog* 消息。在 4.9.3 或 BIND 8 服务器中这条消息的级别是 LOG_INFO，而 4.9.4 到 4.9.7 服务器中它的级别为 LOG_DEBUG。

下面这个消息表明你的名字服务器在保护它自己，抵御一种类型的网络攻击：

```
Jun 11 11:40:54 terminator named[131]: Response from unexpected source
                                       ([204.138.114.3].53)
```

你的名字服务器向远程名字服务器发送了一个查询，收到的响应却不是来自于任何一个你列出的该远程名字服务器的地址。可能的安全破坏是：一个入侵者使你的名字服务器查询一个远程名字服务器，而同时入侵者发回响应（假装成是来自该远程服务器的响应），入侵者希望你将该响应加入到你的缓存中。他可能会发送一个假的 PTR 记录，将他的某台主机的 IP 地址指向你信任的某个主机的域名。一旦假的 PTR 记录在你的缓存中了，入侵者就使用某个 BSD “r”命令（例如，*rlogin*）来获得对你的系统的访问权。

不那么偏执的管理员会意识到，如果父名字服务器只知道孩子区的多宿主主机的一个 IP 地址，也会出现同样的情况。父名字服务器告诉你的名字服务器它所知道的那个 IP 地址，当你的服务器查询远程名字服务器时，它却从其他 IP 地址进行响应。如果远程名字服务器主机上运行的是 BIND 的话，就不会出现这样的情况，因为 BIND 总是使用查询进来的那个 IP 地址进行响应。这个 *syslog* 消息的级别为 LOG_INFO。

下面是个很有趣的 *syslog* 消息：

```
Jun 10 07:57:28 terminator named[131]: No root nameservers for
                class 226
```

目前定义的类只有：类 1，Internt（IN）；类 3，Chaos（CH）；和类 4，Hesiod（HS）。哪来的类 226 呢？你的名字服务器通过这条 *syslog* 消息是想说：一定出了什么问题，因为根本就没有类 226。那么你该怎么办呢？什么都不能做，真的。这条消息没有提供给你足够的信息，你不知道这个查询从哪里来或是要查询什么。再重申一次，如果类字段坏掉了，查询中的域名可能也就没用了。造成这个问题真正的原因可能是远程名字服务器或解析器坏掉了，或者这个 UDP 数据报坏掉了。这个 *syslog* 消息的级别是 LOG_INFO。

如果你正在备份其他某个区，可能会出现这样的消息：

```
Jun  7 20:14:26 wormhole named[29618]: Zone "253.253.192.in-addr.arpa"
                (class 1) SOA serial# (3345) rcvd from [192.249.249.10]
                is < ours (563319491)
```

唉，那个讨厌的*253.253.192.in-addr.arpa*管理员改变了序列号的格式，却忘了告诉你。这难道是对你管理他的区的辅名字服务器的一种感谢吗？给那个管理员发一个消息，看看这个改动是特意的，还只是个笔误。如果改动是特意的，或者你不想同管理员联系，那么你可以在本地处理这个问题：结束你的辅名字服务器，删除该区的备份文件，再重启动你的服务器。这个过程将删除你的辅名字服务器对老的序列号的任何理解，所以它会很高兴地接受新的序列号。这个 *syslog* 消息的级别是 LOG_NOTICE。

顺便说一句，如果那个讨厌的管理员运行的是 BIND 8 或 9 名字服务器，那他一定没注意到（或忽视了）他服务器记录在日志中的消息，这个消息告诉他，他把区的序列号循环了。在 BIND 8 名字服务器上，这个消息看上去是这样的：

```
Jun 7 19:35:14 terminator named[3221]: WARNING: new serial number < old
              (zp->z_serial < serial)
```

在 BIND 9 名字服务器上，它看上去是这样的：

```
Jun 7 19:36:41 terminator named[9832]: dns_zone_load: zone movie.edu/IN: zone
serial has gone backwards
```

这个消息的级别是 LOG_NOTICE。

你可能还想要提醒这个管理员在对名字服务器做了任何修改后都别忘了检查一下 *syslog*。

下面这个 BIND 8 消息对你来说无疑会是很熟悉的：

```
Aug 21 00:59:06 terminator named[12620]: Lame server on 'foo.movie.edu'
         (in 'MOVIE.EDU'?): [10.0.7.125].53 'NS.HOLLYWOOD.LA.CA.US':
         learnt (A=10.47.3.62,NS=10.47.3.62)
```

在 BIND 9 中，它看起来是这样的：

```
Jan 15 10:20:16 terminator named[14205]: lame server on 'foo.movie.edu' (in
'movie.EDU'?): 10.0.7.125#53
```

“是的，船长，她正在吸泥浆呢！”在 Internet 的大海里确实也有泥巴，那就是坏的授权。父名字服务器将子域授权给一个孩子名字服务器，而这个孩子名字服务器却不是该子域的权威。在这里，*edu* 名字服务器将 *movie.edu* 授权给 10.0.7.125，而在这个主机上的名字服务器却不是 *movie.edu* 的权威。除非你认识 *movie.edu* 的管理员，否则可能对此你也无能为力。4.9.3 服务器中这个 *syslog* 消息的级别是 LOG_WARNING， 4.9.4 到 4.9.7 服务器中则是 LOG_DEBUG，而在 BIND 8 或 9 中为 LOG_INFO。

如果你的 BIND 4.9 或其后续版本的名字服务器的配置文件中有：

```
options query-log
```

或者你的版本 8 或 9 的服务器的配置文件中有：

```
logging { category queries { default_syslog; }; };
```

那么对你的名字服务器收到的每个查询都会有一个 LOG_INFO 等级的 *syslog* 消息：

```
Feb 20 21:43:25 terminator named[3830]:
            XX /192.253.253.2/carrie.movie.edu/A
Feb 20 21:43:32 terminator named[3830]:
            XX /192.253.253.2/4.253.253.192.in-addr.arpa/PTR
```

BIND 9 名字服务器支持查询日志记录，正如 BIND 9.1.0。不过格式有了一点变化：

```
Jan 13 18:32:25 terminator named[13976]: client 192.253.253.2#1702: query: carrie.
movie.edu IN A
Jan 13 18:32:42 terminator named[13976]: client 192.253.253.2#1702: query: 4.
253.253.192.in-addr.arpa IN PTR
```

这些消息中包括有发送查询的主机的 IP 地址以及查询本身。在 BIND 8.2.1 或其后续版本的名字服务器上，递归查询会被标记为 XX+，而不是 XX。如果你要在一个很繁忙的名字服务器上记录所有的查询，就要保证有足够的磁盘空间。（在正在运行的名字服务器上，你可以用 *querylog* 命令来打开或关闭查询日志记录。）

启动 BIND 8.1.2 的服务器，你可能会看见这样一组 *syslog* 消息：

```
May 19 11:06:08 named[21160]: bind(dfd=20, [10.0.0.1].53):
                Address already in use
May 19 11:06:08 named[21160]: deleting interface [10.0.0.1].53
May 19 11:06:08 named[21160]: bind(dfd=20, [127.0.0.1].53):
```

```
                Address already in use
May 19 11:06:08 named[21160]: deleting interface [127.0.0.1].53
May 19 11:06:08 named[21160]: not listening on any interfaces
May 19 11:06:08 named[21160]: Forwarding source address
                is [0.0.0.0].1835
May 19 11:06:08 named[21161]: Ready to answer queries.
```

在 BIND 9 名字服务器上，会是这样：

```
Jul 27 17:15:58 terminator named[7357]: listening on IPv4 interface lo, 127.0.0.
1#53
Jul 27 17:15:58 terminator named[7357]: binding TCP socket: address in use
Jul 27 17:15:58 terminator named[7357]: listening on IPv4 interface eth0, 206.168.
194.122#53
Jul 27 17:15:58 terminator named[7357]: binding TCP socket: address in use
Jul 27 17:15:58 terminator named[7357]: listening on IPv4 interface eth1, 206.168.
194.123#53
Jul 27 17:15:58 terminator named[7357]: binding TCP socket: address in use
Jul 27 17:15:58 terminator named[7357]: couldn't add command channel 0.0.0.0#953:
address in use
```

这是因为你已经运行了一个名字服务器，却又启动了一个而没有关闭前一个。和你期望的不太一样，这第二个名字服务器仍然能够运行，只是不监听任何的接口。

理解 BIND 的统计数据

应该定期浏览一下你的名字服务器上的统计数据，只要看看它们都有多忙。我们会给你一个名字服务器统计数据的例子，并且讨论每一行都是什么意思。在正常运行过程中，名字服务器会处理很多的查询和响应，所以首先，我们要告诉你一个典型的交互是什么样的。

如果在脑袋里没有一个查找过程是如何进行的概念，是很难理解统计数据的意思的。为了帮助你理解名字服务器的统计数据，图 7-2 表明了一个应用程序想查找一个名字时会发生什么。这个应用程序 FTP 会查询本地名字服务器。本地名字服务器以前曾查找过这个区中的数据，知道了远程名字服务器在哪儿。它查询每一个远程名字服务器，每个两次，试着找到答案。同时，应用查询超时了，又发送了一个查询，要求同样的信息。

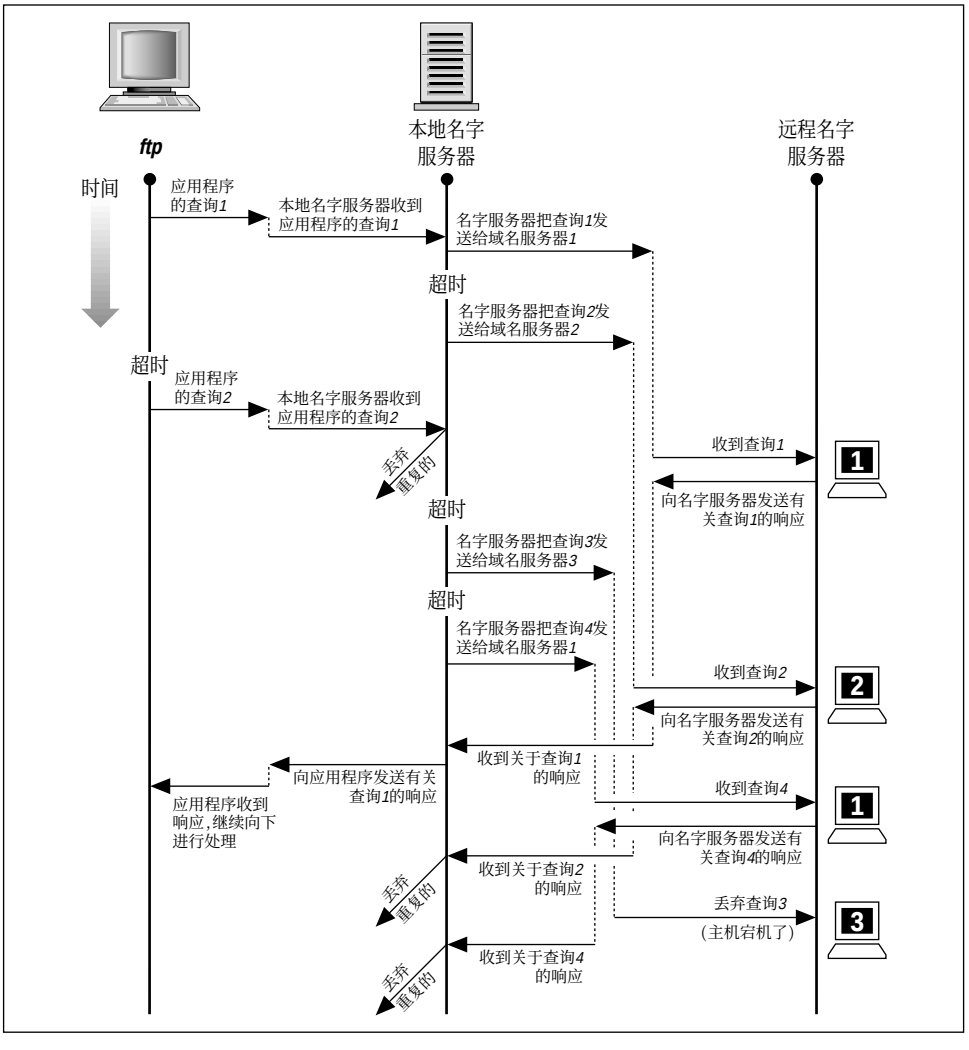


图 7-2　查询 / 响应交互的例子

注意，BIND 名字服务器只有在它仍然试图回答原来的那个查询时才能识别重复的查询。本地名字服务器能识别来自应用程序的重复查询，因为它仍然在试图回答这个查询。但是远程名字服务器 1 并不能识别来自本地名字服务器的重复的查询，因为它已经回答了前一个查询。在本地名字服务器收到来自远程名字服务器 1 的第一个响应，所有其他的响应都被作为重复而丢弃了。这个会话需要下面这些交互：

记住，即使名字服务器给远程名字服务器发送了一个查询，远程的名字服务器也不一定马上就能收到这个查询。查询有可能被下层网络延迟或丢失，或者可能是因为远程名字服务器正忙于处理另一个应用程序。

交互	数量
应用程序到本地名字服务器	2 个查询
本地名字服务器到应用程序	1 个响应
本地名字服务器到远程名字服务器 1	2 个查询
远程名字服务器 1 到本地名字服务器	2 个响应
本地名字服务器到远程名字服务器 2	1 个查询
远程名字服务器 2 到本地名字服务器	1 个响应
本地名字服务器到远程名字服务器 3	1 个查询
远程名字服务器 3 到本地名字服务器	0 个响应

这些交互会在本地名字服务器上产生下面这样的统计数据：

统计数据	原因
收到 2 个查询	来自本地主机上的应用程序
1 个重复查询	来自本地主机上的应用程序
发送 1 个回答	发给本地主机上的应用程序
收到 3 个响应	来自远程名字服务器
2 个重复响应	来自远程名字服务器
2 个 A 查询	查询地址信息

在我们这个例子中，本地名字服务器只收到来自一个应用程序的查询，它也向远程名字服务器发送查询。正常地，本地名字服务器也会收到来自远程名字服务器的查询（也就是说，除了询问远程名字服务器它需要知道的信息之外，远程名字服务器也会询问本地名字服务器它们想要知道的信息），不过为了简化问题，我们没有显示任何远程的查询。

BIND 4.9 和 8 的统计数据

现在你已经看过了应用程序和名字服务器之间的一个典型交互，以及由此产生的统

计数据，就让我们来看一个量更大的统计数据的例子。用 *ndc* 可以从你的 BIND 8 名字服务器获得统计数据：

```
#  ndc stats
```

在较老版本的 BIND 4 名字服务器上没有 *ndc*，要向 *named* 发送 ABRT 信号：

```
#  kill -ABRT `cat /var/run/named.pid`
```

（在 SVR4 以前的文件系统中，进程 ID 是存放在 *etc/named.pid* 中的。）等几秒钟，然后看一下名字服务器的工作目录中的 *named.stats* 文件（BIND 8）或查看一下文件 */var/tmp/named.stats* 或 */usr/tmp/named.stats*（BIND 4）。如果统计数据没有存储到这个文件中，你的服务器可能在编译时没有定义 STATS，所以无法收集统计数据。下面是 Paul Vixie 的一个 BIND 4.9.3 名字服务器的统计数据。BIND 8 名字服务器有这里列出的所有项，除了 RnotNsQ，另外排列的顺序也有些不同。BIND 9 名字服务器（如 BIND 9.1.0）有完全不同的一套统计数据，在下一节中我们会讲到。

```
+++ Statistics Dump +++ (800708260) Wed May 17 03:57:40 1995
746683    time since boot (secs)
392768    time since reset (secs)
14        Unknown query types
268459    A queries
3044      NS queries
5680      CNAME queries
11364     SOA queries
1008934   PTR queries
44        HINFO queries
680367    MX queries
2369      TXT queries
40        NSAP queries
27        AXFR queries
8336      ANY queries
++ Name Server Statistics ++
(Legend)
      RQ    RR    RIQ   RNXD    RFwdQ
      RFwdR RDupQ RDupR RFail   RFErr
      RErr  RTCP  RAXFR RLame   ROpts
      SSysQ SAns  SFwdQ SFwdR   SDupQ
      SFail SFErr SErr  RNotNsQ SNaAns
      SNXD
(Global)
   1992938 112600 0 19144 63462 60527 194 347 3420 0  5 2235 27 35289 0
   14886 1927930 63462 60527 107169  10025 119 0 1785426 805592  35863
[15.255.72.20]
   485 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 485 0 0 0  0 0 0 0 485  0
[15.255.152.2]
```

```
   441 137 0 1 2 108 0 0 0 0  0 0 0 0 0  13 439 85 7 84  0 0 0 0 431  0
[15.255.152.4]
   770 89 0 1 4  69 0 0 0 0  0 0 0 0 0  14 766 68 5 7  0 0 0 0 755  0
...  <lots of entries deleted>
```

如果你的 BIND 8 名字服务器在“ Global ”后面没有任何针对每个 IP 地址的部分，要是你想查看每个主机的统计数据，就要在*options*语句中将*host-statistics*设成yes：

```
options {
     host-statistics yes;
};
```

不过，保存主机统计数据需要相当数量的内存，所以你可能并不想总是这样做，除非你想建立一份有关你的名字服务器活动的档案。

让我们一行一行地来看一看这些统计数据：

```
+++ Statistics Dump +++ (800708260) Wed May 17 03:57:40 1995
```

这是该部分统计数据被存储下来的时间。括号中的数字（800708260）是从 Unix 纪元 —— 1970 年 1 月 1 日到该数据被存储下来的时间之间所经过的秒数。BIND 为你把它转换成了真实的日期和时间：1995 年 5 月 17 日，上午 3 点 57 分 40 秒。

```
746683    time since boot (secs)
```

这是本地名字服务器已经运行的时间长度。要将它转换成天数，就把它除以 86400（60 × 60 × 24，一天中的秒数）。这个服务器已经运行大约 8.5 天了。

```
392768    time since reset (secs)
```

这是本地名字服务器自从上一次重新加载以后已经运行的时间长度。只有当该服务器是一个或多个区的主名字服务器时，你可能会看到这个数和启动以来的时间不一样。是某些区的辅名字服务器会自动通过区传送获取新数据，通常不会重新加载。因为我们这里的这个名字服务器已被重置，它可能就是某些区的主名字服务器。

```
14        Unknown query types
```

这个名字服务器收到了14个查询，请求它不认识的类型的数据。要么是有人在试验某种新类型，要么是实现中有某种缺陷，要么是 Paul 该升级他的名字服务器了。

```
268459    A queries
```

有 268459 个地址查询。地址查询通常是最常见的查询类型。

```
3044      NS queries
```

有 3044 个名字服务器查询。在内部，当名字服务器试图查找根区的服务器时，会产生 NS 查询。在外部，*dig* 或 *nslookup* 这样的应用程序也会被用来查找 NS 记录。

```
5680      CNAME queries
```

有些版本的 *sendmail* 使用 CNAME 程序来规范化邮件地址（用规范名来替代别名）。而其他版本的 *sendmail* 则使用 ANY 查询（待会儿我们就会讲到）。此外，CNAME 查找通常是来自于 *dig* 或 *nslookup* 的。

```
11364     SOA queries
```

SOA 查询是辅名字服务器用来检查它们的区数据是否是最新的。如果数据不是最新的，接下来就是一个 AXFR 查询来进行区传送。由于这一组统计数据也显示了 AXFR 记录，我们可以作出结论说辅名字服务器从这个服务器加载了区数据。

```
1008934   PTR queries
```

指针查询将地址映射成名字。许多种软件都查找 IP 地址：*inetd*、*rlogind*、*rshd*、网络管理软件以及网络追踪软件。

```
44        HINFO queries
```

主机信息查询最有可能是来自交互式查找 HINFO 记录的人。

```
680367    MX queries
```

像 *sendmail* 这样的邮件发送程序会进行 MX 查询，这是正常电子邮件传送过程的一部分。

```
2369      TXT queries
```

一定有些应用程序在进行文本程序，因为这个数有这么大。可能是像 *Harvest* 这样的工具，它是一项由科罗拉多大学研究的信息搜索和检索技术。

```
40        NSAP queries
```

这是一个相对而言比较新的数据类型，用来将域名映射成OSI网络服务访问点（OSI Network Service Access Point）地址。

```
27        AXFR queries
```

辅名字服务器使用 AXFR 查询来进行区传送。

```
8336      ANY queries
```

ANY查询请求有关一个名字的任意类型的数据。*sendmail*是最常使用这个查询类型的程序。由于 *sendmail* 查找某个邮件目的地的CNAME、MX和地址记录，它就会进行一个请求ANY记录类型的查询，这样所有的资源记录马上就都放到本地名字服务器的缓存中了。

剩下的统计数据每个主机都不一样。如果你看一看和你的名字服务器交换包的主机列表，你就会发现你的名字服务器是多么繁琐、冗长了，你会看到列表中有上百个，甚至上千个主机。这个列表大得惊人，但是统计数据本身只是有点趣而已。我们将解释所有这些统计数据，甚至那些计数为0的项，不过你可能会发现只有几个统计数据是有用的。为了使统计数据更好读一点，你需要一个工具来扩展一下这些统计数据，因为输出的格式有点太紧凑了。我们写了一个工具，叫做 *bstat*，就是完成这个功能的。它的输出看起来是这样的：

```
hpcvsop.cv.hp.com
        485 queries received
        485 responses sent to this name server
        485 queries answered from our cache
relay.hp.com
        441 queries received
        137 responses received
          1 negative response received
          2 queries for data not in our cache or authoritative data
        108 responses from this name server passed to the querier
         13 system queries sent to this name server
        439 responses sent to this name server
         85 queries sent to this name server
          7 responses from other name servers sent to this name server
         84 duplicate queries sent to this name server
        431 queries answered from our cache
hp.com
        770 queries received
         89 responses received
          1 negative response received
          4 queries for data not in our cache or authoritative data
```

```
 69 responses from this name server passed to the querier
 14 system queries sent to this name server
766 responses sent to this name server
 68 queries sent to this name server
  5 responses from other name servers sent to this name server
  7 duplicate queries sent to this name server
755 queries answered from our cache
```

在原始统计数据（不是*bstat*的输出）中每个主机IP地址后面都跟有一个计数表。在开头，这个表的列标题是一个像密码一样的图表符号。这个图表符号分为几行，但是主机统计数据都是在一行上的。在接下来这一节中，我们将简要地解释一下每一列都是什么意思，以与这个名字服务器交谈的一个主机15.255.152.2（*relay.hp.com*）的统计数据为例。我们先解释一下*relay*的图表符号中的列标题（例如，RQ）以及后面跟的它的计数值。

```
RQ 441
```

RQ是从*relay*收到的查询的数量。进行这些查询是因为*relay*需要有关由这个名字服务器服务的区的信息。

```
RR 137
```

RR是*relay*收到的响应的数量。是这个名字服务器对查询的响应。不要把这个值同RQ联系起来，因为它们之间并没有什么关系。RQ是*relay*问的问题的个数；RR是relay给这个名字服务器的回答的个数（因为这个名字服务器也向*relay*询问了信息）。

```
RIQ 0
```

RIQ是收到的来自*relay*的反向查询的个数。进行反向查询原来是为了将地址映射成名字，但是现在这个功能由PTR记录来实现了。较老版本的*nslookup*在启动时使用反向查询，所以你可能会看到RIQ计数不为0。

```
RNXD 1
```

RNXD是收到的来自*relay*的no such domain（没有这样的域）回答的个数。

```
RFwdQ 2
```

RFwdQ是收到的来自*relay*的在回答之前还需要进一步处理的查询的个数。这个计

数值要大大超过将它们的解析器（用 *resolv.conf*）配置成向你的名字服务器发送所有的查询的主机数。

```
RFwdR 108
```

RFwdR 是收到的来自 *relay* 的回答原始查询（译注 1）的响应的个数，这些响应将会传回进行该查询的应用程序。

```
RDupQ 0
```

RDupQ 是来自 *relay* 的重复查询的个数。只有在解析器（用 *resolv.conf*）配置成查询这个名字服务器时，你才会看到重复查询。

```
RDupR 0
```

RDupR 是来自 *relay* 的重复响应的个数。当名字服务器在它的悬而未决的查询列表中找不到引起该响应的原始查询时，这个响应就是重复响应。

```
RFail 0
```

RFail 是来自 *relay* 的 SERVFAIL 响应的个数。SERVFAIL 响应表示某种服务器错误。服务器错误响应的发生通常是因为远程服务器读了一个区数据文件，发现有语法错误。任何请求那个带有错误区数据文件的区中数据的查询都会导致远程名字服务器发送一个回答，指示服务器错误。这可能是引起SERVFAIL回答最常见的原因。服务器错误响应的发生也会是由于远程名字服务器试图分配更多的内存却不成功，或者是由于远程辅名字服务器的区数据期满。

```
RFErr 0
```

RFErr 是来自 *relay* 的 FORMERR 响应的个数。FORMERR 的意思是远程名字服务器说本地名字服务器的查询有格式错误。

```
RErr  0
```

RErr 既不是 SERVFAIL 也不是 FORMERR 的错误的个数。

```
RTCP  0
```

译注 1： 这里的原始查询是相对于重复查询而言的。

RTCP 是通过 TCP 连接收到的来自 *relay* 的查询个数。（大多数查询使用的是 UDP。）

```
RAXFR 0
```

RAXFR 是进行的区传送的个数。计数值为 0 表明 *relay* 并不是由这个名字服务器服务的任何区的辅名字服务器。

```
RLame 0
```

RLame 是收到的坏授权的个数。如果这个计数值不为 0，就意味着有的区被授权给了在这个 IP 地址的名字服务器，而这个名字服务器并不是这个区的权威。

```
ROpts 0
```

ROpts 是设置了 IP 选项的包的个数。

```
SSysQ 13
```

SSysQ 是发送到 *relay* 的系统查询的个数。系统查询是由本地名字服务器发起的查询。因为系统查询是用来保持根名字服务器列表为最新的，所以大多数系统查询都会到根名字服务器。不过，如果地址记录在名字服务器记录之前超时了，系统查询也用来查找该名字服务器的地址。由于 *relay* 并非根名字服务器，所以发送这些查询一定是出于后一种原因。

```
SAns  439
```

SAns 是发送到 *relay* 的查询的个数。这个名字服务器对 *relay* 发给它的 441 个（RQ）查询回答了 439 个。我很想弄清楚那两个没有回答的到底怎么样了呢……

```
SFwdQ 85
```

SFwdQ 是当答案不在这个名字服务器的区数据或缓存中时发送（转发）给 *relay* 的查询的个数。

```
SFwdR 7
```

SFwdR 是来自其他名字服务器发送（转发）给 *relay* 的响应的个数。

```
SDupQ 84
```

SDupQ是发送给*relay*的重复查询的个数。不过并不像看起来的那么糟。如果查询先被发送给其他的名字服务器，这个重复查询计数值也会加1。所以，*relay*可能在第一次收到这些查询时就已经回答了，只不过因为这个查询在送到*relay*之前也发送给了其他名字服务器，所以这个查询也算作重复查询。

```
SFail 0
```

SFail是发送给*relay*的SERVFAIL响应的个数。

```
SFErr 0
```

SFErr是发送给*relay*的FORMERR响应的个数。

```
SErr  0
```

SErr是以*relay*为目的地的失败的*sendto()*系统调用的个数。

```
RNotNsQ 0
```

RNotNsQ是收到的并非来自于名字服务器端口——端口53的查询的个数。在BIND 8以前，所有的名字服务器查询都来自端口53。任何来自端口53以外的查询都是来自解析器的。不过BIND 8名字服务器可以从53以外的端口进行查询，这就使得这个统计数据没有用了，因为你不再能够区分解析器查询和名字服务器查询。因此，BIND 8就删除了RNotNsQ这个统计数据。

```
SNaAns 431
```

SNaAns是发送给*relay*的非权威回答的个数。在发送给*relay*的439个回答（SAns）中，有431个是来自于缓存数据。

```
SNXD  0
```

SNXD是发送给*relay*的no such domain（没有这个域）回答的个数。

BIND 9统计数据

BIND 9.1.0是最早记录统计数据的BIND 9版本。要用*rndc*来使得BIND 9存储它的统计数据：

```
%  rndc stats
```

与 BIND 8 名字服务器一样，BIND 9 名字服务器会将统计数据存储到它的工作目录下的 *named.stats* 文件中。不过，这些统计数据与 BIND 8 的完全不同。下面是我们的一个 BIND 9 名字服务器的统计数据文件的内容：

```
+++ Statistics Dump +++ (979436130)
success 9
referral 0
nxrrset 0
nxdomain 1
recursion 1
failure 1
--- Statistics Dump --- (979436130)
+++ Statistics Dump +++ (979584113)
success 651
referral 10
nxrrset 11
nxdomain 17
recursion 296
failure 217
--- Statistics Dump --- (979584113)
```

每次在收到一个统计数据命令时，名字服务器都在最后增加了一个新的统计数据部分（“ +++ Statistics Dump +++ ”与“ --- Statistics Dump --- ”之间的部分）。括号中的数字同以前的统计数据文件一样，是从Unix纪元到该数据被存储下来的时间之间所经过的秒数。不幸的是，BIND 没有为你转换这个数字，不过你可以用 date 命令将它转换成更容易阅读的格式。例如，要转换从 Unix 纪元（1970 年 1 月 1 日）开始的 979584113 秒，你可以使用：

```
%   date -d '1970-01-01 979584113 sec'
Mon Jan 15 18:41:53 MST 2001
```

现在让我们一行一行地仔细阅读这些统计数据：

```
success 651
```

这是名字服务器处理的成功的查询数目。成功的查询是指那些不以指向别的名字服务器或错误为结果的查询。

```
referral 10
```

这是名字服务器处理的以指向别的名字服务器为结果的查询的个数。

```
nxrrset 11
```

这是名字服务器处理的一种查询的数目，这种查询的结果是响应会说：对于指定的域名没有所要求查询的记录类型。

```
nxdomain 17
```

这是名字服务器处理的结果为查询者所指定的域名不存在的查询的个数。

```
recursion 296
```

这是名字服务器收到的请求递归处理获得回答的查询的个数。

```
failure 217
```

这是名字服务器收到的一种查询的个数，这种查询的结果是除了 *nxrrset* 和 *nxdomain* 中所包括的以外的其他错误。

显然，这些统计数据比 BIND 8 名字服务器所记录的要少得多，但以后的 BIND 9 版本也许会记录更多的内容。

使用 BIND 统计数据

你的名字服务器工作正常吗？你怎么知道正常的运转是什么样的呢？从这么一次的统计数据中我们无法判断这个名字服务器是否正常。你必须要观察你的名字服务器在一段时间内产生的统计数据，这样才能知道对于你的配置来说，什么样的数据才算是正常。名字服务器与名字服务器之间，这些数据会有显著的不同，这取决于产生查询的应用程序，服务器的类型［主、辅还是只缓存（caching-only）］和这个服务器所服务的区在名字空间中所处的层次。

在这些统计数据中要注意的一点是你的名字服务器每秒钟收到的查询的个数。用收到查询的个数除以名字服务器运行的秒数，就可以得到。Paul 的 BIND 4.9.3 名字服务器在 746683 秒内收到了 1992938 个查询，大约是每秒 2.7 个查询，一个不算特别忙的服务器（注 4）。如果你的服务器收到的查询数量有点太多了，看看是哪些主机是主要的查询者，考虑一下它们发送这些查询是否合理。有时你可能会决定你需要更多的服务器来处理负荷；我们将在下一章讲到这种情况。

注 4： 回想一下运行最简单的 BIND 的根名字服务器，它们可以每秒处理上千个查询。

本章内容：

- 需要多少名字服务器呢？
- 增加更多的名字服务器
- 注册名字服务器
- 更改 TTL
- 预防灾难
- 应付灾难

第八章

扩展你的域

“ 你想要变成多大呢？ ”它问。
“ 哦，我对大小没有什么特别的要求，”
爱丽丝急忙回答，“ 只是不要老是这么频繁地
变来变去，你知道 ... ”
“ 那你现在满意了吧？ ”毛毛虫说。
“ 好的，不过，如果你不介意的话，先生，
我想再大一点 ... ”

需要多少名字服务器呢？

在第四章中，我们建立起了两个名字服务器。你最少也需要两个名字服务器。根据你网络的大小，可能远不止只需要两个名字服务器。运行五到七个服务器，且还有一个不在你自己的网络上，这也是很常见的。要多少个名字服务器才算够呢？这要视你的网络而定了。下面就是一些对你很有用的指导原则：

- 在你的每个网络或子网上都运行至少一个名字服务器，这就排除了路由器失败的影响。充分利用你的多宿主主机，因为它们（根据它的定义就能知道）与多个网络相连。
- 如果你有一个文件服务器，还有一些无盘节点机，就在文件服务器上运行一个名字服务器，来为这一组机器服务。

- 要在靠近大的、多用户计算机的地方运行名字服务器，但不一定非要在该机器上运行。用户以及他们的进程可能会生成许多查询，而作为管理员，你必须努力保持多用户主机的工作正常。但是你需在满足用户的要求和在有许多人访问的系统上运行对安全性要求非常高的名字服务器所承担的风险之间做出一种平衡。
- 在你的网络之外至少再运行一个名字服务器。这使得你的网络不可用时，数据却仍然可用。你可能会说当你不能访问那个主机的时候，查找它的地址又有什么用呢？不过，如果你的网络可用，但其他的名字服务器出问题时，那么在你网络之外的名字服务器就能起作用了。如果你和 Internet 上某个组织关系密切（比如说，某个学校或商业伙伴），它们会愿意帮你管理一个辅服务器的。

图 8-1 显示的是一个网络拓扑图的例子，后面是一个简要的分析，告诉你它是如何工作的。

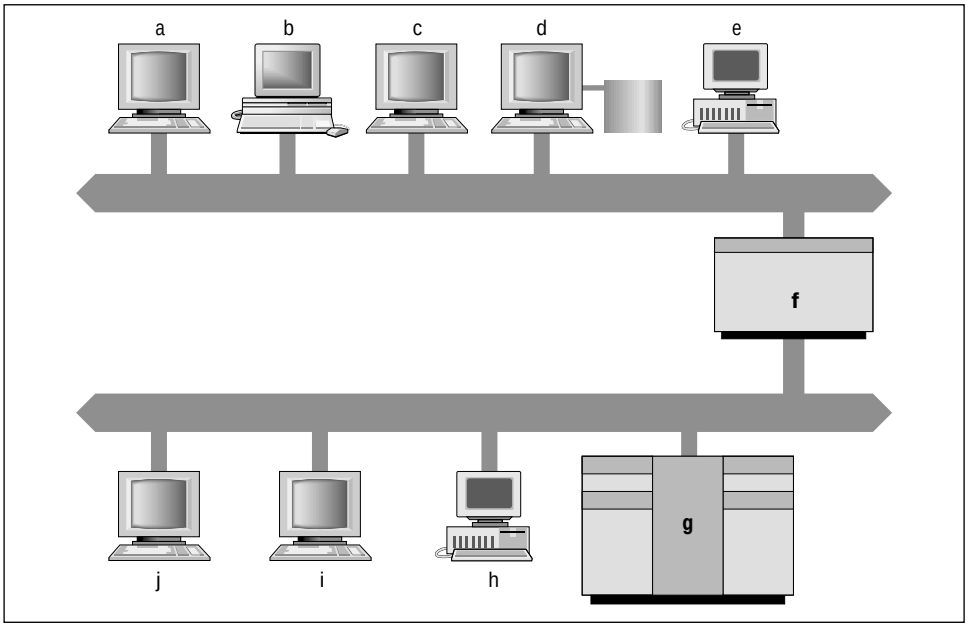


图 8-1　一个网络拓扑的例子

注意，即使你遵循我们的指导原则，还是会有许多运行名字服务器的位置来让你选择的。主机 *d* 是主机 *a*、*b*、*c* 和 *e* 的文件服务器，可以在此运行一个名字服务器。主

机 *g* 是一个大的多用户的主机，也是一个很好的候选者。但是最好的选择也许是主机 *f*，它比较小，而且在两个网络上都有接口。你只需要运行一个名字服务器，而不是两个，而且这个服务器是运行在一个被密切关注着的主机上的。如果你想要每个网络上都有不止一个名字服务器，你还可以在主机 *d* 或 *g* 上再建一个。

把我的名字服务器放在哪儿呢？

除了要给你一个需要多少个名字服务器的概念之外，上面这些标准还有助于你决定将名字服务器安置于何处（比如说，可以安置在文件服务器和多宿主主机上）。不过，在选择正确的主机时，还有其他一些很重要的因素要考虑。

要考虑的其他因素包括：主机的连接性、它运行的软件（是 BIND，还是其他什么）、维护名字服务器的同构性，还有安全性：

连接性

名字服务器连接得好是很重要的。如果一个主机陷在某个用速度缓慢又脆弱的串联线连接起来的处于停滞状态的子网当中，它就算速度再快，可靠性再好，也不会有任何用处。试着找一台靠近你到 Internet 连接的主机（如果你有的话），或者找一个连接很好的 Internet 主机作为你的区的辅名字服务器。至于在你自己的网络上，要将名字服务器安装在靠近你网络集线器的主机上。

你的主名字服务器是否连接得好是非常非常重要的。若想使区传送很可靠，那么主名字服务器到所有从它那里更新的辅名字服务器之间的连接要非常好。当然，像其他名字服务器一样，高速、可靠的网络连接对它会很有好处。

软件

在为名字服务器选择主机时还要考虑的一个因素就是主机运行的软件。从软件的角度讲，运行名字服务器的主机最好是一台运行了供应商支持的 BIND 8.2.3 或 9.1.0 版本的主机，并且它的 TCP/IP 协议实现也要非常稳定。（最好是基于 4.3 或 4.4 BSD Unix 的网络实现，我们是比较偏爱 Berkeley 的。）你可以根据源代码自己编译 BIND 8.2.3 或 9.1.0，这并不是太难，而且最新版本的可靠性非常好，不过你很难让你的供应商提供相应的支持。如果你并不一定非要使用 BIND 8 才有的特性，那你就可以使用供应商提供的基于较早 BIND 版本的移植（如，4.9.7），这能使你获得供应商的支持，这是很值得的。

同构性

最后需要考虑的因素是你的名字服务器的统一性。不管你多么相信“开放系统”，在各种不同的Unix版本间转来转去是很让人困惑的。如果可能的话，要尽量避免在过多不同的平台上运行名字服务器。你将会浪费大量的时间，来把你的一些程序（或者我们提供的！）从一种操作系统移植到另外一种上去，或者在三种不同的Unix版本上查找存放*nslookup*或*named.conf*的不同位置。此外，不同供应商的各种版本的Unix往往支持的是不同版本的BIND，这也会给你带来很多麻烦。例如，如果你觉得你所有的名字服务器都需要BIND 8或9的安全特性，那么你就要为你所有的名字服务器选择一个支持BIND 8或9的平台。

安全性

你当然不会希望黑客们霸占你的名字服务器来帮助他们攻击你自己的主机或是通过Internet攻击别的网络，所以在安全的主机上运行你的名字服务器是很重要的。对于一个大的多用户系统，如果你不能信任其用户，就不要在上面运行名字服务器。如果你有某种计算机用于网络服务而又不允许一般的人注册，那么它们就是运行名字服务器的很好的选择。如果你只有一台或几台真正安全的主机，考虑将主名字服务器装在上面，因为如果它出问题所带来的危害要比辅名字服务器出问题所带来的危害大得多。

虽然上面这些只是比较次要的考虑因素，在给定的子网上要有名字服务器比只在某个最佳的主机上运行服务器要重要得多，不过，在做出选择时还是要记住这些标准。

容量规划

如果你有一个很庞大的网络，或是用户需要大量的名字服务器工作，你可能会发现：你需要比我们建议的处理负载所需要的更多名字服务器。或者我们的建议在开始时还是够用的，但随着用户不断地加到网络中来或安装了需要大量名字服务器工作的程序，你可能会发现你的名字服务器陷入不停地查询之中，几乎停顿。

什么样的任务是需要大量名字服务器工作的呢？在网上冲浪就是一种。发送电子邮件，特别是向有大量用户的邮件列表发送邮件，也会大量使用域名服务的。使用大量远程过程调用的程序也是这样。甚至运行一些图形化的用户环境也会大量使用域

名服务。例如，基于 X Windows 的用户环境就会查询名字服务器来检查访问列表（还有一些其他事情）。

你们当中那些机灵的家伙可能会问“但是我怎么才能知道我的名字服务器负担过重了？我应该看些什么才会知道呢？”问得好！

内存的使用情况应该是监视名字服务器运行的最重要的方面了。当*named*是很多区的权威时，就会变得很大。如果*named*的大小加上你运行的其他进程的大小超过了主机实际内存的大小，那么主机就会频繁地交换内存块（swap），但实际上却什么都没做。即使你的主机有足够多的内存来运行所有的进程，这些巨大的名字服务器启动和生成子进程都会变得很慢（譬如，处理区传送）。另外一个问题是针对BIND 4的：因为BIND 4名字服务器会创建新的*named*进程来处理区传送，那么很有可能同时会有不止一个*named*进程在运行，一个回答查询，一个或者更多的是在为区传送服务。如果你的BIND 4主名字服务器已经消耗了5兆或者10兆内存，那么有时要乘上两或者三倍才是真正使用的内存数量。

另一个你能用来衡量名字服务器负载情况的指标就是*named*进程给主机CPU的负荷。配置正确的名字服务器并不会占用太多的CPU时间，所以过多的使用CPU往往是配置错误的表现。像*top*这样的软件可以帮助你计算你的名字服务器的平均CPU使用情况。不过，对于可接受的CPU利用率并没有绝对的标准。我们在此提供一个大概的标准：平均CPU利用率为5%是可以接受的；10%有点儿太高了，除非这台主机就是用来提供名字服务的（注1）。

下面是一个帮助你想像通常的数字会是什么样的*top*输出的例子，运行在一个相当安静的名字服务器上：

```
last pid: 14299; load averages: 0.11, 0.12, 0.12       18:19:08
68 processes: 64 sleeping, 3 running, 1 stopped
Cpu states: 11.3% usr, 0.0% nice, 15.3% sys, 73.4% idle, 0.0% intr, 0.0% ker
Memory: Real: 8208K/13168K act/tot Virtual: 16432K/30736K act/tot Free: 4224K

  PID USERNAME PRI NICE   SIZE   RES STATE  TIME   WCPU    CPU COMMAND
```

注1： *top*是一个由Bill LeFebvre开发的非常方便的软件，它可以提供给你连续的关于不同进程对于CPU的使用情况的报告。最新版本的*top*可以通过匿名FTP从*eecs.nwu.edu*上下载*/pub/top/top-3.4.tar.Z*文件。

```
  89 root       1    0  2968K 2652K sleep  5:01  0.00%  0.00% named
```

确实很安静。下面是一个非常繁忙（但还没有过载）的名字服务器上运行的 *top* 的输出情况：

```
load averages: 0.30, 0.46, 0.44                  system: relay 16:12:20
39 processes: 38 sleeping, 1 waiting
Cpu states: 4.4% user, 0.0% nice, 5.4% system, 90.2% idle, 0.0% unk5, 0.0% unk6,
0.0% unk7, 0.0% unk8
Memory: 31126K (28606K) real, 33090K (28812K) virtual, 54344K free Screen #1/ 3

   PID USERNAME PRI NICE  SIZE   RES   STATE   TIME  WCPU   CPU  COMMAND
 21910 root       1    0  2624K  2616K sleep 146:21  0.00% 1.42% /etc/named
```

另一个可参照的统计数据是名字服务器每分钟（或每秒钟，如果你有一个很忙的名字服务器的话）收到的查询数。在这里也没有绝对的标准：一台运行 NetBSD 的高速 Pentium III 每秒大约可以很轻松地处理上百个查询，而一个较老的 Unix 主机可能每秒处理几个查询都有困难。

要检查你的名字服务器收到的查询数量，最简单的方法就是查看名字服务器的内部统计数据，而这些数据是可以通过配置名字服务器使之定期写到*syslog*中的（注2）。例如，你可以配置你的名字服务器每个小时将数据写入一次文件（实际上，这就是 BIND 8 的默认配置），同时比较每个小时收到的查询总数的差别：

```
options {
        statistics-interval 60;
};
```

BIND 9 名字服务器不支持 *statistics-interval* 子语句，但你可以用 *rndc* 告诉 BIND 9 名字服务器每个小时转储一次统计数据，例如在 *crontab* 中：

```
0 * * * *  /usr/local/sbin/rndc stats
```

你要特别注意高峰时段。星期一早上通常是很忙的，因为许多人在星期一要做的第一件事就是处理他们在周末收到的邮件。

注 2： 一些老版本的BIND名字服务器需要强制将统计数据写到文件中：使用信号ABRT（对于更老的系统来说就是信号IOT）。BIND 4.9 名字服务器每个小时会自动将数据写入文件中，但是 4.9.4 和后续版本的名字服务器也需要用 ABRT 信号来强迫它们去做。

你还应该注意午饭之后开始的一段时间，也就是人们回到办公桌前，重新开始工作的时候，几乎都是在同一时刻。当然，如果你的组织分布在几个时区，你就要运用你自己的判断力来决定什么时候会是很忙的时段。

下面是一台 BIND 8.2.3 名字服务器上 *syslog* 文件中的一些片段：

```
Aug  1 11:00:49 terminator named[103]: NSTATS 965152849 959476930 A=8 NS=1
SOA=356966 PTR=2 TXT=32 IXFR=9 AXFR=204
Aug  1 11:00:49 terminator named[103]: XSTATS 965152849 959476930 RR=3243 RNXD=0
RFwdR=0 RDupR=0 RFail=20 RFErr=0 RErr=11 RAXFR=204 RLame=0 ROpts=0 SSysQ=3356

SAns=391191 SFwdQ=0 SDupQ=1236 SErr=0 RQ=458031 RIQ=25 RFwdQ=0 RDupQ=0 RTCP=101316
SFwdR=0 SFail=0 SFErr=0 SNaAns=34482 SNXD=0 RUQ=0 RURQ=0 RUXFR=10 RUUpd=34451
Aug  1 12:00:49 terminator named[103]: NSTATS 965156449 959476930 A=8 NS=1
SOA=357195 PTR=2 TXT=32 IXFR=9 AXFR=204
Aug  1 12:00:49 terminator named[103]: XSTATS 965156449 959476930 RR=3253 RNXD=0
RFwdR=0 RDupR=0 RFail=20 RFErr=0 RErr=11 RAXFR=204 RLame=0 ROpts=0 SSysQ=3360

SAns=391444 SFwdQ=0 SDupQ=1244 SErr=0 RQ=458332 RIQ=25 RFwdQ=0 RDupQ=0 RTCP=101388
SFwdR=0 SFail=0 SFErr=0 SNaAns=34506 SNXD=0 RUQ=0 RURQ=0 RUXFR=10 RUUpd=34475
```

收到查询的总数是记录在 *RQ* 字段上的（上面那些粗体字部分）。要计算每小时收到的查询数只用从第二个 *RQ* 值中减去第一个 *RQ* 值：458332 － 458031 ＝ 301。

即使你的主机速度足够快，能处理它收到的所有查询，你也应该保证DNS流量没有给你的网络带来不应有的负担。在大多数 LAN 中，DNS 流量只是网络带宽的很小一部分，可以不予考虑。但是在速度很慢的租用线路或拨号连接上，DNS流量所占的带宽就很值得考虑一下了。

要大概估计你LAN上的DNS流量，那就把一个小时内收到的查询次数（RQ）和所发送的回答总数（SAns）之和乘上800位（也就是100字节，DNS消息粗略的平均大小），再除以3600（一小时的秒数）就是所使用的网络带宽了。这就能告诉你网络的带宽有多少是用于 DNS 流量的（注 3）。

给你一个关于“正常”的概念，NSFNet最近的（1995年4月）流量报告显示DNS流量只占他们主干网总流量（按字节计算）的5%多一点。NSFNet的数字是建立在

注 3： 要想找一个能够自动分析BIND统计数据的软件的话，你可以看看DNS资源目录下工具那部分中Nigel Campbell开发的 *bindgraph*，地址是 *http://www.dns.net/dnsrd/tools.html*。

实际流量采样的基础上的，不是像我们这样，用名字服务器的统计数据算出来的（注4）。如果你还想对你名字服务器收到的流量有更正确的认识，你可以使用LAN协议分析器来自己做流量采样。

一旦发现你的名字服务器超负荷了，该怎么办呢？首先，最好检查一下，保证你的名字服务器没有受到错误程序进行的查询的干扰。为此，你需要查出所有这些查询都是从哪儿来的。

如果你运行的是BIND 4.9或者8.1.2名字服务器，那么你只需将统计信息写入文件后，分析文件内容就能知道都有哪些解析器和名字服务器在查询你的名字服务器。这些名字服务器是根据不同主机来进行统计的，这对于你要跟踪频繁使用你的名字服务器的那些用户就非常方便。BIND 8.2或更新的名字服务器默认地并不保留这些统计数据，要想让它们保留每台主机的统计数据，就在 *options* 语句中使用 *host-statistics* 子语句，如下所示（注5）：

```
options {
     host-statistics yes;
};
```

例如，看一看下面的统计数据：

```
+++ Statistics Dump +++ (829373099) Fri Apr 12 23:24:59 1996
970779    time since boot (secs)
471621    time since reset (secs)
0    Unknown query types
185108    A queries

6    NS queries
69213    PTR queries
669    MX queries
2361    ANY queries
++ Name Server Statistics ++
(Legend)
    RQ      RR      RIQ      RNXD      RFwdQ
    RFwdR   RDupQ   RDupR    RFail     RFErr
    RErr    RTCP    RAXFR    RLame     ROpts
    SSysQ   SAns    SFwdQ    SFwdR     SDupQ
    SFail   SFErr   SErr     RNotNsQ   SNaAns
```

注4： 我们不清楚这些数字对Internet的现状而言是否具有代表性，但是要想从除了NSFNet之外的其他商业主干网提供者那里骗取这些数据简直比登天还难。

注5： BIND 9不支持 *host-statistics* 子语句，BIND 9.1.0也是如此。

```
    SNXD
(Global)
    257357 20718 0 8509 19677  19939 1494 21 0 0  0 7 0 1 0
    824 236196 19677 19939 7643  33 0 0 256064 49269  155030
 [15.17.232.4]
    8736 0 0 0 717  24 0 0 0 0  0 0 0 0 0  0 8019 0 717 0
    0 0 0 8736 2141  5722
[15.17.232.5]
    115 0 0 0 8  0 21 0 0 0  0 0 0 0 0  0 86 0 1 0  0 0 0 115 0  7
[15.17.232.8]
    66215 0 0 0 6910  148 633 0 0 0  0 5 0 0 0  0 58671 0 6695 0
    15 0 0 66215 33697  6541
[15.17.232.16]
    31848 0 0 0 3593  209 74 0 0 0  0 0 0 0 0  0 28185 0 3563 0
    0 0 0 31848 8695  15359
[15.17.232.20]
    272 0 0 0 0  0 0 0 0 0  0 0 0 0 0  0 272 0 0 0  0 0 0 272 7  0
[15.17.232.21]
    316 0 0 0 52  14 3 0 0 0  0 0 0 0 0  0 261 0 51 0  0 0 0 316 30  30
[15.17.232.24]
    853 0 0 0 65  1 3 0 0 0  0 2 0 0 0  0 783 0 64 0  0 0 0 853 125  337
[15.17.232.33]
    624 0 0 0 47  1 0 0 0 0  0 0 0 0 0  0 577 0 47 0  0 0 0 624 2  217
[15.17.232.94]
    127640 0 0 0 1751  14 449 0 0 0  0 0 0 0 0  0 125440 0 1602 0
    0 0 0 127640 106  124661
[15.17.232.95]
    846 0 0 0 38  1 0 0 0 0  0 0 0 0 0  0 809 0 37 0  0 0 0 846 79  81
-- Name Server Statistics --
--- Statistics Dump --- (829373099) Fri Apr 12 23:24:59 1996
```

在 *Global* 后面，每台主机都列得很清楚了，IP 地址是用大括号括起来的。看一下 Legend 这部分，你会发现每个记录的第一项都是 RQ，就是接收到的查询。这就使得我们对主机 15.17.232.8、15.17.232.16 和 15.17.232.94 要特别注意，因为来自它们的查询几乎占了全部查询的 88%。

如果你运行的是一台老版本的名字服务器，那么找到发送这些讨厌的查询的解析器和名字服务器的惟一方法就是打开名字服务器的调试功能。（我们将在第十三章中深入讨论这一部分。）你应该感兴趣的是你的名字服务器收到的查询的源 IP 地址。当仔细阅读调试输出时，要特别注意发送重复查询的主机，特别是请求相同或者类似信息的主机。这可能说明某台主机上所运行的软件没有配置好或者有 bug，或者一台陌生的名字服务器正在对你的名字服务器进行攻击，不断地向你发送查询。

如果所有的查询看上去都是合法的，那么只能再添加一个新的名字服务器了。不过别把名字服务器随便地放在哪儿，使用调试的输出信息来帮助你决定最好在哪儿安

装一个名字服务器。在DNS流量淹没了你的以太网的情况下，随便选一台主机，在上面安装一个名字服务器是不会有任何作用的。你需要考虑哪些主机总是提出查询，然后再决定如何最好地为他们提供名字服务。这里就是一些能帮助你做出决定的提示：

- 寻找来自共享同一文件服务器的主机上的解析器的查询。你可以在该文件服务器上运行一个名字服务器。
- 寻找来自大得多的用户主机上的解析器的查询。你可以在此运行一个名字服务器。
- 寻找来自其他子网的解析器的查询。这些解析器应该配置成查询它们自己子网上的名字服务器。如果该子网上没有名字服务器，就创建一个。
- 寻找来自同一桥接器（bridge）部分的解析器的查询（如果你使用了桥接器的话）。如果你在一个桥接器部分运行了一个名字服务器，那么流量就不用桥接到网络的其他部分了。
- 寻找来自彼此以负载较轻的网络相连的主机上的查询。你可以在那个网络上运行一个名字服务器。

增加更多的名字服务器

当你需要为你的区创建新的名字服务器时，最简单的办法就是添加辅名字服务器。你已经知道如何添加辅名字服务器了，我们在第四章中讲过，而且你也建立起了一个辅名字服务器，再做一遍实在是易如反掌。但是如果你不加选择地添加辅名字服务器是会有麻烦的。

如果你为一个区运行了大量的辅名字服务器，光是应付辅名字服务器轮询检查它们的区数据是否最新，主名字服务器就够受的了。为了解决这个问题还要采取一系列的活动：

- 增加更多的主名字服务器。
- 增大刷新间隔，这样辅名字服务器就不会过于频繁地来检查了。
- 让一些辅名字服务器从其他辅名字服务器那里加载。

- 创建只缓存（caching-only）名字服务器（待会儿会讲到）。
- 创建“部分辅”（partial-slave）名字服务器（待会儿也会讲到）。

主名字服务器和辅名字服务器

创建更多的主名字服务器意味着给你添加了额外的工作，因为你不得不手工同步*/etc/named.conf*和区数据文件。无论你喜不喜欢这样，你总是希望有其他的选择。你可以使用*rdist*或*rsync*（注6）这样的工具来简化这些文件分发的过程。用来同步主服务器间文件的工具：*distfile*（注7）可以就是这么简单：

```
dup-primary:

# 拷贝named.conf文件到dup的主名字服务器

/etc/named.conf  -> wormhole
    install ;

# 拷贝/var/named（区数据文件等）的内容到dup的主名字服务器

/var/named -> wormhole
    install ;
```

对于多个主名字服务器来说：

```
dup-primary:

primaries =  ( wormhole carrie )
/etc/named.conf  -> {$primaries}
    install ;

/var/named -> {$primaries}
    install ;
```

你甚至可以用*rdist*通过添加下面这样的*special*选项来使得你的名字服务器进行重载：

```
special /var/named/* "ndc reload" ;
special /etc/named.conf "ndc reload" ;
```

注6： *rsync*是一个远程文件同步程序，它只传送文件间的差异。有关它的更多情况请浏览*http://rsync.samba.org*。

注7： 文件*rdist*用来查找哪些文件要更新。

这些选项告诉 *rdist* 在任何一个文件被更改时执行所引用的命令。

增加你的区的刷新间隔是另外一个选项。不过这延缓了新数据的传播，在某些情况下这不是问题。如果你每天凌晨 1 点（通过 *cron* 来运行）用 *h2n* 来重新生成你的区数据，然后有 6 个小时的时间来分发这些数据，那么你所有的辅名字服务器都会在 7 点时得到最新的数据（注 8）。这对于你的用户来说应该是可以接受的。具体的细节在本章后面的“ 更改其他的 SOA 值 ”一节中会提到。

你甚至可以让你的一些辅名字服务器从其他辅名字服务器那里加载数据。辅名字服务器可以从其他辅名字服务器而不是主名字服务器那里加载区数据。辅名字服务器并不知道它是从哪种名字服务器上加载数据的。惟一重要的就是提供区数据的辅服务器是该区的权威。配置这个没有什么技巧。在该辅名字服务器配置文件中不要指定一个主名字服务器的 IP 地址，而只需指定其他辅名字服务器的 IP 地址就可以了。

下面是文件 *named.conf* 的内容：

```
// 本 slave 服务器从另外一个辅服务器 wormhole 那里更新
zone "movie.edu" {
                type slave;
                masters { 192.249.249.1; };
                file "bak.movie.edu";
};
```

对于 BIND 4 服务器来说，这个文件看起来会有点不同。

下面是文件 *named.boot* 的内容：

```
; 本辅服务器从另外一个辅服务器 wormhole 那里更新
secondary   movie.edu   192.249.249.1   bak.movie.edu
```

不过，当你使用这种二级分布结构时，数据从主名字服务器传播到所有的辅名字服务器可能会需要两倍的时间。要知道辅服务器只在每隔一段刷新时间后才会去检查它们的区数据是否最新。因而要花费整个刷新间隔的时间才能让所有的一级辅服务器都从主名字服务器获得区的新拷贝。同样地，二级辅服务器也要用整个刷新间隔的时间来从一级辅服务器那里获得区的新拷贝。因此从主服务器传播到所有的辅名字服务器就需要两倍的刷新间隔时间。

注 8： 当然，如果你使用 NOTIFY，辅服务器会更快地得到新数据。

避免这个问题的一个方法是使用 BIND 8 和 9 的 NOTIFY 特性。这个特性默认是打开的，而且在主名字服务器上的数据被修改后立即开始进行区传送。不幸的是，这个特性只对于运行 BIND 8 和 9 的辅名字服务器有效（注 9）。我们会在第十章中详细讨论 NOTIFY。

如果你决定将你的网络配置成有两层（或更多层）辅名字服务器，那么就要小心避免出现更新回路。如果我们将 *wormhole* 配置成从 *diehard* 更新，而又不小心将 *diehard* 配置成从 *wormhole* 更新，那么它们俩都不会从主名字服务器获得数据。它们只是拿着自己过时了的序列号互相比较，还总是觉得它们自己的数据都是最新的。

只缓存服务器

创建只缓存（caching-only）服务器是当你需要更多服务器时的又一选择。只缓存服务器不是任何区的权威（除了 *0.0.127.in-addr.arpa*）。只缓存这个名字并不是暗示主和辅名字服务器就不使用缓存，其实它们也用。这个名字的意思是这个服务器惟一的功能就是查找数据再放入缓存。与主名字服务器和辅名字服务器一样，一个只缓存名字服务器需要一个根线索文件和 *db.127.0.0* 文件。只缓存名字服务器的 *named.conf* 文件包含下面一些内容：

```
options {
    directory "/var/named";  // 或者你的数据目录
};

zone "0.0.127.in-addr.arpa" {
     type master;
    file "db.127.0.0";
};

zone "." {
    type hint;
    file "db.cache";
};
```

在 BIND 4 名字服务器上，*named.conf* 文件的内容如下：

```
directory /var/named  ; 或者你的数据目录
primary 0.0.127.in-addr.arpa  db.127.0.0  ; 回送地址
cache   .                     db.cache
```

注 9： 附带说一下，微软的 DNS 服务器也有这个特性。

一个只缓存名字服务器既能查找你区内的域名也能查找你区外的名字，就像主和辅名字服务器一样。不同之处在于当只缓存名字服务器第一次查找你区中的名字时，它只向你的区的某个主或辅名字服务器请求回答就结束了，而主或辅名字服务器再根据它的权威数据对该问题做出回答。那么只缓存服务器会向哪个主或辅名字服务器查询呢？与你区外的名字服务器一样，它是从你的父区的某个名字服务器那里找到为你的区服务的名字服务器的。有没有办法给一个只缓存名字服务器的缓存中添加一些数据，让它知道哪些主机上运行了你的区的主和辅名字服务器呢？不行。你不能使用 *db.cache* 文件，它只能用于根名字服务器线索。而且实际上，让只缓存名字服务器只从父区的名字服务器那里获取你的权威名字服务器会更好一些：这样便于你使区的授权信息保持最新。如果在只缓存名字服务器上固定一个权威名字服务器列表，你很可能会忘了更新。

只缓存名字服务器的真实值都来自建立了它的缓存之后。每次它查询一个权威名字服务器并收到回答，它就将回答中的记录放在缓存当中。随着时间的推移，缓存就会变得包含有很多经常被解析器查询的信息。这样，你就省去了区传送的开销——一个只缓存名字服务器不需要进行区传送。

部分辅名字服务器

在只缓存名字服务器和辅名字服务器之间还有一种名字服务器：它只是某一部分本地区的辅名字服务器。我们把它叫做部分辅名字服务器（也许没有别人这样叫）。假设 *movie.edu* 有二十个 C 类网络（以及相应的二十个 *in-addr.arpa* 区）。我们不是为所有的二十一个区（所有的二十个 *in-addr.arpa* 区再加上 *movie.edu*）创建一个辅名字服务器，而是为 *movie.edu* 和主机所在的 *in-addr.arpa* 区创建一个部分辅名字服务器。如果主机有两个网络接口，那么这个名字服务器就应该是三个区的辅名字服务器：*movie. edu* 和两个 *in-addr.arpa* 区。

让我们再为一个名字服务器构造硬件环境吧。我们把新主机叫做 *zardoz.movie.edu*，它的 IP 地址是 192.249.249.9 和 192.253.253.9。我们要在 *zardoz* 上新建一个部分辅名字服务器，其 *named.conf* 文件如下：

```
options {
     directory "/var/named";
};
```

```
zone "movie.edu" {
     type slave;
     masters { 192.249.249.3; };
     file "bak.movie.edu";
};

zone "249.249.192.in-addr.arpa" {
     type slave;
     masters { 192.249.249.3; };
     file "bak.192.249.249";
};

zone "253.253.192.in-addr.arpa" {
     type slave;
     masters { 192.249.249.3; };
     file "bak.192.253.253";
};

zone "0.0.127.in-addr.arpa" {
     type master;
     file "db.127.0.0";
};

zone "." {
     type hint;
     file "db.cache";
};
```

对于 BIND 4 名字服务器来说，*named.boot* 文件如下：

```
directory   /var/named
secondary   movie.edu                192.249.249.3 bak.movie.edu
secondary   249.249.192.in-addr.arpa 192.249.249.3 bak.192.249.249
secondary   253.253.192.in-addr.arpa 192.249.249.3 bak.192.253.253
primary     0.0.127.in-addr.arpa     db.127.0.0
cache       .                        db.cache
```

这个服务器只是*movie.edu*和二十个*in-addr.arpa*区中的两个的辅名字服务器。一个“完整的”（full）辅名字服务器将会在 *named.conf* 文件中包括二十一个不同的 *zone* 语句。

部分辅名字服务器有什么作用呢？它们对管理员来说是不需要花费很大工作量的，因为它们的 *named.conf* 不会经常变动。如果是一个所有 *in-addr.arpa* 区的权威名字服务器，只要我们的网络一发生变化就得添加和删除 *in-addr.arpa* 区（及其在 *named.conf* 中相应的内容）。对一个非常庞大的网络而言，工作量会大得惊人。

部分辅名字服务器也能回答它所收到的大多数查询。大多数查询都是关于*movie.edu*和两个*in-addr.arpa*区中数据的。为什么呢？因为大多数查询该名字服务器的主机都是在它连接的这两个网络上的：192.249.249 和 192.253.253。而这些主机主要是与它们自己网络上的主机通信。这就会产生大量对应于本地网络（local network）的*in-addr.arpa*区中数据的查询。

注册名字服务器

当你建立起越来越多的名字服务器后，你可能会问一个问题，我需要将所有的主和辅名字服务器都注册到我的父域吗？不，你只用向你的父域注册你想让你区以外的名字服务器使用的那些服务器。例如，如果你为你的区建立了九个名字服务器，你可以只选择将其中的四个告诉你的父区。在你的网络中，所有九个名字服务器都可用。不过，只有五个名字服务器供解析器被配置成向它们进行查询的主机使用（例如，在*resolv.conf*中进行配置）。它们父区的名字服务器并未对它们授权，所以它们不可能被远程名字服务器查询。只有向你的父区注册过的四个服务器才能被其他名字服务器查询，包括你网络上的只缓存和部分辅名字服务器。这个配置如图 8-2 所示。

除了可以选择哪些名字服务器能被外部查询，只注册你的区的部分名字服务器还有一个技术上的原因：一个 UDP 响应消息能容纳的名字服务器数量是有一定限制的。实际上，大概能装十个名字服务器记录。根据数据的情况（有多少服务器是在同一个域中的），可多可少（注 10）。而且也没有什么必要注册多于十个服务器，如果这十个名字服务器都不可达到，目的主机也就不太可能可达到了。

如果你建立起一个新的权威名字服务器，并决定注册它，那么就先列出以它为权威的区的父区来。你要和这些父区的管理员联系。举个例子来说，我们想将刚才建立的*zardoz.movie.edu*上的名字服务器注册。要想将这个辅名字服务器在所有相应的区中都注册，我们要与*edu*和*in-addr.arpa*的管理员联系。（要想知道是谁管理你的父域，请参阅第三章）。

注 10： 这就是 Internet 的根名字服务器的域名会改变的原因。所有的根都被移到同一个域*root-servers.net*中，就是为了能充分利用域名压缩，能在一个 UDP 包中装入尽可能多的根。

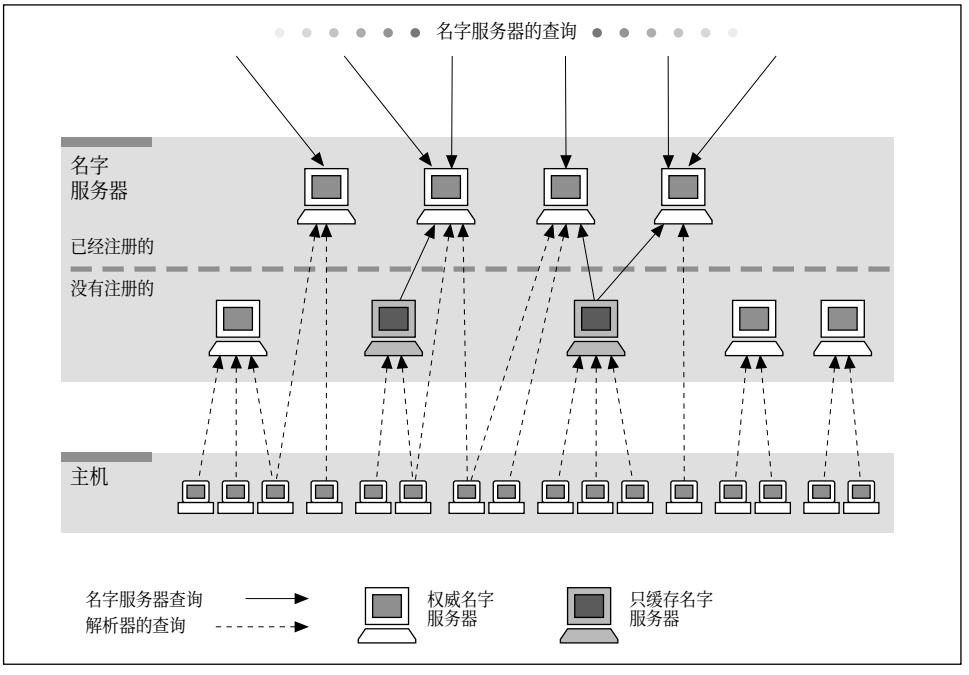


图 8-2　只注册你的一部分名字服务器

当你与父区的管理员联系时，请遵照他们的网站上指定的程序（如果有的话）。如果没有标准的更改程序，就将以该新名字服务器为权威的区的域名发给他们。如果这个新的名字服务器是在新的区当中，你还要把新名字服务器的IP地址给他们。实际上，如果没有正式的格式来提交信息，你最好以区数据文件的格式，将该区待注册的名字服务器的完整列表发给你的父区，如果需要，还要加上地址。这将避免任何可能的模糊和混淆。

由于我们的网络原来是由 InterNIC 分配的，我们按照 *http://www.arin.net/cgi-bin/amt.pl*的网上程序来更改我们的注册。（如果想手工来做的话，就把*http://www.arin.net/register/templates/modifytemplate.txt*的表格发给他们。如果他们没有模板供我们使用，就给 *in-addr.arpa* 的管理员发邮件，应该是下面这样的：

```
Howdy!

I've just set up a new slave name server on
zardoz.movie.edu for the 249.249.192.in-addr.arpa
and 253.253.192.in-addr.arpa zones.  Would you
```

```
please add NS records for this name server to the
in-addr.arpa zone?  That would make our delegation
information look like:

253.253.192.in-addr.arpa. 86400 IN NS terminator.movie.edu.
253.253.192.in-addr.arpa. 86400 IN NS wormhole.movie.edu.
253.253.192.in-addr.arpa. 86400 IN NS zardoz.movie.edu.

249.249.192.in-addr.arpa. 86400 IN NS terminator.movie.edu.
249.249.192.in-addr.arpa. 86400 IN NS wormhole.movie.edu.
249.249.192.in-addr.arpa. 86400 IN NS zardoz.movie.edu.

Thanks!

Albert LeDomaine
al@robocop.movie.edu
```

注意，我们在NS和A记录中都明确地指定了TTL吗？这是因为我们的父名字服务器对这些记录不享有权威，我们的名字服务器才是这些记录的权威。包括了这些信息，我们就指定了我们区授权的TTL。当然，我们的父区对TTL的值该为多少有它自己的想法。

在这里，每个名字服务器都该有的A记录反而并不一定需要，因为名字服务器的域名并不是在 *in-addr.arpa* 区中，它们是在 *movie.edu* 中。所以想要查找 *terminator.movie.edu*或*wormhole.movie.edu*的名字服务器仍然能通过授权给*movie.edu*的名字服务器找到它们的地址。

将部分辅名字服务器注册到父区合不合适呢？实际上，这不太合适，因为它只是你某些 *in-addr.arpa* 区的权威。从管理上来说，只注册支持所有本地区的服务器会容易一点；这样一来，你就不用总是要明了哪些名字服务器是哪些区的权威。你的所有父区都能授权给同一组名字服务器：你的主名字服务器和你的“完整的”辅名字服务器。

不过，如果你并没有太多的名字服务器，或是如果你总能记住哪些名字服务器是哪些区的权威，那就没关系，只管注册一个部分辅名字服务器。

另外，只缓存名字服务器是绝对不能被注册的。一个只缓存名字服务器绝少含有任何区的完整信息，它总是只有最近查询过的某些区的部分信息。如果父区名字服务器错误地将外部的名字服务器指向了一个只缓存名字服务器，那个外部的名字服务器会向该只缓存名字服务器发送一个非递归查询。这个只缓存名字服务器的缓存中

可能有所需要的数据，但也可能没有。如果它没有所要的数据，你会将查询者指向它所知道的最好的名字服务器（离所查询的域名最近的名字服务器）——其中可能包括该只缓存名字服务器它自己！那么外部那个可怜的名字服务器可能将永远也得不到答案。这种错误的配置被称为无用授权（lame delegation），实际上就是将一个区授权给并不是它权威的名字服务器。

更改 TTL

一个有经验的管理员需要知道如何设定区数据的TTL值来最好地为他服务。还记得吗，资源记录的TTL就是任何服务器能在缓存中保持该数据的时间长度，以秒为单位。所以如果某个资源记录的TTL是3600（秒），而你的网络之外的服务器将该记录放入缓存，1小时之后，它就要从缓存中删除该记录了。如果1小时到了之后还需要同样的数据，服务器就必须再次查询你的一个名字服务器。

当我们介绍TTL时，我们要强调的是，你对TTL值的选择表明：数据拷贝的及时性是以增加名字服务器的负载为代价的。TTL值较小就意味着，你网络外的名字服务器将不得不经常从你的名字服务器获取数据，从而保持数据的及时性。不过这样一来，你的名字服务器就要忙于应付它们的查询了。

不过，你选择的TTL并不是一成不变的。你可以定期更改TTL来适应你的需要，有经验的管理员就是这么做的。

假设我们知道我们有台主机将要移到其他网络上去。这台主机上有*movie.edu*的电影库。它收藏了大量文件，我们让Internet用户可以访问它。通常，外部的名字服务器根据$TTL控制语句中设置的默认TTL，或在BIND 8.2以前的名字服务器上根据SOA记录中的最小TTL（minimum TTL），将我们的主机地址放入缓存中。在我们的例子文件中我们将*movie.edu*的默认TTL设为3小时。一个名字服务器正好在改变前将旧的地址记录放入缓存，那么它将把这个错误的地址保存3小时。但是损失3小时的连接是不可接受的。我们怎样才能将这种连接损失减小到最低限度呢？我们可以减小TTL值，这样外部的服务器就只能将地址记录保存较短的时间。减小了TTL，我们就使得外部的服务器更频繁地更新它们的数据，这也就意味着我们在移动系统时所做的任何修改都将很快地传播出去。我们能将TTL设为多少呢？不幸的是，我们不能将TTL设为零，这就意味着“根本不要将这个记录放入缓存中”。有

些较老版本的BIND 4名字服务器就不能返回TTL为零的记录，而是返回空回答或是SERVFAIL错误。不过，将TTL设为30秒这样短就可以了。最简单的办法就是在*db.movie.edu*文件里将$TTL控制语句中的TTL改小。如果你不在区数据文件中特别声明，名字服务器就会将默认的*TTL*应用到每个资源记录。所以，如果你减小了默认的TTL，这个新的较小的默认值不仅会应用于要移动的主机的地址，还会应用于所有的区数据。这种方法的缺点就是你的名字服务器将回答比以前多许多的查询，因为查询服务器会将你区的所有数据在缓存中只存放较短的时间。另一个较好的选择就是只对受到影响的地址记录使用不同的TTL。

若想为某个资源记录添加特别声明的TTL，要在类字段中IN的前面写上TTL的值。默认地，TTL是以秒为单位的，但也可以将单位指定为*m*（分钟）、*h*（小时）、*d*（天）和*w*（周），就像在$TTL控制语句中一样。下面是一个在*db.movie.edu*中使用特定的TTL的例子：

```
cujo  1h IN  A    192.253.253.5  ; explicit TTL of 1 hour
```

如果你是个很细心的人，而且又读过RFC 1034的话，你可能就会注意到，当在一个较老的名字服务器上加载这个记录时有一个潜在的问题：*cujo*的地址中特别声明的TTL是1小时，而在BIND 8.2以前的名字服务器上该区的最小TTL，也就是SOA记录中的TTL字段还要大一些。那么会优先使用哪一个呢？

如果较老的BIND完全遵循最初的DNS RFC，那么SOA记录中的TTL真的就定义为是该区中所有资源记录的最小TTL值。那么你只能将特别声明的TTL指定为比这个最小值大。但是较老的BIND名字服务器没有这样做。换句话说，在BIND中，这个“最小值”并不是真正的最小值。BIND将SOA记录中这个最小TTL字段解释成默认的TTL。（当然，较新的BIND用$TTL专门设置默认的TTL。）如果某个记录没有特别声明TTL，那么就使用这个最小值。如果某个记录有特别声明TTL，BIND允许它比这个最小值还要小。响应时，对于这个记录就使用较小的TTL，而对其他记录都使用SOA记录中的“最小”TTL。

你还应该知道，做回答时，辅名字服务器会提供和主名字服务器同样的TTL——也就是说，如果主名字服务器对某个记录使用的TTL为1小时，那么辅名字服务器也是一样。辅名字服务器并不会根据它加载该区已经有多久了，来缩减TTL。所以，如果某个资源记录的TTL设得比默认值小，主和辅名字服务器都会对该资源记录使

用相同的、较小的 TTL。如果辅名字服务器已经过了该区数据的有效期，就会使这个区失效，而决不会只使区中某个资源记录失效。

所以，如果你知道某个数据很快会被改变，BIND 允许你为该资源记录设置一个较小的TTL。这样一来，任何将该数据放入缓存的名字服务器都只能保持较短的时间。不过，虽然名字服务器允许某些记录使用较小的TTL，但是大多数管理员都不会花时间来这样做。当一个主机的地址改变时，总是会有一段时间连接不上它的。

往往是这样的，要改变地址的主机不是本地主集线器，所以不会影响太多的人。不过，如果要移动的是一个邮件分发器（mail hub）或是一个主 Web 服务器或 FTP 存储库（比如，电影库），一天无法连接将会是不可接受的。在这样的情况中，管理员应该事先计划好，减小将要更改数据的 TTL 值。

记住，数据受影响的 TTL 要在更改发生前就修改好。减小工作站地址记录的 TTL，然后马上就改变工作站的地址几乎或根本就没有作用。可能就在你更改前几秒钟，该地址记录就已经装入缓存了，然后会保持直到旧的 TTL 超时。而且还应该在你的辅名字服务器从你的主名字服务器加载数据之前就进行更改。例如，如果你的默认 TTL 是 12 小时，而你的刷新间隔是 3 小时，一定要在 15 小时前就减小 TTL，这样在你移动主机前，所有较老的、时间较长的 TTL 都会到期了。当然，如果你的所有辅名字服务器都是能使用 NOTIFY 的 BIND 8 或 9，辅名字服务器就不用等到刷新间隔完就能实现同步了。

更改其他的 SOA 值

我们主要讲到了用增加刷新间隔来减小你主名字服务器的负载。下面我们更详细地谈谈刷新，再谈谈剩下的一些 SOA 值。

你应该还记得，刷新值用来控制一个辅名字服务器每隔多久会检查一下区数据是否已经过时。重试值会在第一次连接主名字服务器失败后变成刷新时间。期满值决定当主名字服务器总是不可达到时，区数据最多能被保存多久就要丢弃掉。最后，在 BIND 8.2 以前的名字服务器上，默认 TTL 设定的是区信息能在缓存中被保存多久。在较新的名字服务器上，最后一个 SOA 字段是否定缓存 TTL。

假设我们决定让辅名字服务器每隔1小时就获取一次数据而不是每隔3个小时。我们就将每个区数据文件中的刷新值改为*1h*（或者使用*h2n*的*—o*选项）。由于重试是同刷新相关的，所以我们也将减小重试值 —— 减小到15分钟左右。通常重试是比刷新小的，但并不一定非要这样（注11）。虽然减小刷新值会加速区数据的发布时间，但是同时也会增加名字服务器因加载而带来的负载，因为辅名字服务器会更加经常地检查。不过增加的负载也并不是很大；每个辅名字服务器在区的刷新间隔内只进行一次SOA查询，以检查它的数据和主名字服务器上的数据是否一致。所以如果有两个辅名字服务器，将刷新时间从3小时改为1小时，每3小时才会增加4个对主名字服务器的查询（每个区）。

当然，如果你的所有辅名字服务器都是BIND8或9，并且使用了NOTIFY，刷新的意思就不一样了。但是，如果你哪怕只有一个BIND 4的辅名字服务器，你的区数据就只有经过一个刷新间隔才能到达它那里。

有些较早版本的BIND的辅名字服务器在区加载时就停止回答查询。所以BIND就改成将区加载尽量分开进行，以减少不可用时间。所以，即使你设定了一个很短的刷新间隔，而你的辅名字服务器也不一定会如你所要求的那样频繁地检查。BIND 4名字服务器会在进行完一定数量的区加载后，等待15分钟，再进行下一批。另外，BIND 4.9以及其后的版本可能会比刷新间隔更频繁地进行刷新。这些较新版本的BIND会在刷新间隔一半的时间到整个刷新间隔时间之间随机地等待一段时间，就检查序列号。

期满时间设为1周很常见 —— 如果你总是很难访问到你的更新源，也可以设为1个月。期满时间总是应该比重试和刷新间隔要大得多；如果期满时间比刷新间隔还要小，你的辅名字服务器就会在试着加载新数据之前就使其数据过期。如果你设置的期满时间比更新间隔和重试时间之和要小，或者小于重试时间的两倍，或者小于7天、大于6个月的话，BIND 8就会显示警告信息。（而BIND 9.1.0不会报警。）在大多数情况下选择一个适用于所有BIND 8要求的期满时间将会是个好主意。

如果你的区数据并不会经常变动，你可以考虑提高默认的TTL。默认TTL值通常是几个小时到1天，但你可以使用更长一点的值。1周大概是使TTL有意义的最大值了。如果设得更长，你会发现就不能在比较合理的时间内清除缓存中坏的数据了。

注11： 实际上，如果刷新值比重试的10倍小的话，BIND 8会警告你的。

预防灾难

网络出现故障也是有的。硬件有故障，软件有bug，还有的错误是偶尔的人为失误造成的。有时只会带来一点点不便，比如一小部分用户不能连接。有时后果就会很严重，比如有重要的数据和有价值的工作被丢失。

因为DNS非常依赖于网络，所以它很容易受到网络故障的影响。不过还好，DNS的设计考虑了网络的不完美性：它允许有多个冗余的名字服务器，允许重发查询，还允许重试区传送，等等。

不过DNS并不能对所有可能的灾难都具有保护自己的能力。某些类型的故障，其中有的相当普通，DNS就没有或不能防护。但是只要你花费一点时间和金钱，你就能将故障的威胁降低到最小。

故障

比如停电在世界上的某些地方就很常见。在美国的某些地区，雷暴雨或龙卷风会造成相当长的一段时间停电，或只是间歇性的有电。在其他一些地方，台风、火山爆发或建筑工程也可能会中断电力服务。

当然，如果你所有的主机都停机了，也就不需要名字服务了。不过，问题往往是在电力恢复的时候出现的。遵循我们前面所提的建议，将名字服务器建在了文件服务器和大型多用户机器上。而当电力恢复的时候，这些机器自然会是最后启动的——因为首先要对所有的磁盘都进行*fsck*！这就意味着那时所有启动得快的主机都没有名字服务可用。

根据你主机的启动文件的内容，这会造成各种各样的奇怪问题。Unix主机经常会执行类似下面这样的一些命令：

```
/usr/sbin/ifconfig lan0 inet `hostname` netmask 255.255.128.0 up
/usr/sbin/route add default site-router 1
```

来启用它们的网络接口和添加默认路由。在命令中使用主机的名字（`*hostname*`是本地主机的名字，而*site-router*则是本地路由器的名字）有绝好的两个理由：

- 它使得管理员可以改变路由器的IP地址而不必修改所有的启动文件。

- 它使得管理员可以通过只改变一个文件中的 IP 地址来改变主机的 IP 地址。

不幸的是，如果没有域名服务，*route* 命令将会失败。而如果无法在 */etc/hosts* 文件中找到本地主机的名字和 IP 地址，*ifconfig* 命令也会失败，所以至少在每台主机的 */etc/hosts* 文件中保留这些数据是很有用的。

当启动过程顺序执行到*route*命令时，网络接口已经可以用了，而且主机会试着使用名字服务来将路由器的名字映射为一个 IP 地址。因为主机在执行 *route* 命令前没有默认的路由器，它只能够使用本地子网上的那些名字服务器。

如果正在启动的主机可以访问在它本地子网上的一台正在运行的名字服务器，那就可以成功地执行*route*命令。但是，经常是一台或多台它能访问的名字服务器还没有启动，那么将会发生什么就取决于 *resolv.conf* 文件的内容了。

只有在 *resolv.conf* 文件中只列出了一个名字服务器时，（或者文件中根本没有列出名字服务器，解析器默认地使用本地主机上的名字服务器，）BIND解析器会在发生问题时去使用本地的主机表。如果只配置了一台名字服务器，那么解析器就会去查询它，如果解析器每次发送查询时，网络都会返回一个错误信息，那么解析器就会回到主机表去搜索相应的信息。可能导致解析器使用主机表的原因有：

- 收到一个 ICMP port unreachable 消息
- 收到一个 ICMP network unreachable 消息
- 无法发送 UDP 包（譬如，因为本地主机上还没有启动网络功能）（注 12）

但是，如果被配置为供解析器使用的主机根本没有开机，解析器根本不会收到任何错误消息。名字服务器好像就是一个黑洞。在尝试了 75 秒以后，解析器会因为超时而向调用它的应用程序返回一个为空的答案。只有在运行名字服务器的那台主机已经启动了网络功能，但是还没有运行名字服务器时，才会使得解析器收到一个错误消息：ICMP Port unreachable 消息。

总体来说，如果每个网络上都有名字服务器的话，只使用一台名字服务器的配置会正常工作，但是不会像我们希望的那样完美。如果本地的名字服务器在一台和它同处一个网络上的主机启动时还没有运行，那么 *route* 命令就会失败。

注 12: 查看第六章中商家提供的增强功能，以及该解析器算法的各种类型。

这看起来可能很笨拙，但是使用多个名字服务器时会更糟糕。如果*resolv.conf*中列出了多台名字服务器，在使用*ifconfig*配置了主网络接口后，BIND永远也不会去使用主机表了。解析器只会简单地在这些名字服务器之间循环查询，直到得到一个答案或者时间超过 75 秒。

在启动时这特别容易造成问题。如果配置好的名字服务器没有一台可以用，解析器就会超时而不会返回一个 IP 地址，然后添加默认路由也就失败了。

建议

我们的建议听起来有点原始，就是把默认路由器的IP地址直接写在启动文件中，或者使用一个外部文件（许多系统使用*/etc/defaultrouter*这个文件）。这就确保你主机的网络功能会正确启动。

另外一个方法就是在*resolv.conf*文件中只列出你主机所处的本地网络上的一台可靠的名字服务器。这就允许你在启动文件中使用默认路由器的名字，但是需要确保这台路由器的名字会在*/etc/hosts*文件中出现（以防可靠的名字服务器在启动主机时没有运行）。当然，如果运行可靠的名字服务器的主机在你主机启动时还没有启动，那上面所说的就都没有用了。因为任何网络都没有运行，所以也不会返回任何错误信息，来让你的解析器去使用 */etc/hosts* 了。

如果你的供应商的BIND能够配置查询时使用不同服务的顺序，或者如果DNS没有返回答案时去使用*/etc/hosts*，那一定要利用这个特性！在前一种情况下，你可以配置解析器来先检查 */etc/hosts*，这就要在每台主机上保留一份 */etc/hosts* 文件，其中要包括默认路由器和本地主机的名字。在后一种情况下，只需确保有这个文件，其他额外的配置就不必了。

最后的希望是通过使用ICMP Router Discovery Messages（ICMP路由器发现消息）来手工设置默认的路由去解决这个问题。这是 RFC 1256 中描述的对于 ICMP 协议的一个扩展，在网络上使用广播或者多点发送信息来动态地发现和通知路由器。Windows NT 4.0 支持该技术，不过默认情况下是禁止的。要打开它，请参见 Knowledge Base 中的文章 Q223756。Sun 最近版本的 Solaris 上包括了有关该协议的一个实现，也就是 */usr/sbin/in.rdisc*，而且 Cisco 的互联网络操作系统（IOS）也支持这个扩展。

但是，如果你已经正确地添加了默认路由器，可名字服务器还是没有启动该怎么办呢？这就会影响*sendmail*、NFS和其他一些服务。没有DNS的话，*sendmail*无法正确地规范化那些主机名，并且你的NFS安装也会失败。

对于这个问题最好的解决方案就是在一台使用不间断电源的主机上运行名字服务器。如果你那里很少会长时间停电，备用电池也就足够了。如果你那里会长时间停电，而名字服务器又很重要，你应该考虑使用带某种发电机的不间断电源系统（UPS）。

如果你觉得这样太昂贵，你应该试着找到启动最快的主机，在上面运行一个名字服务器。使用日志文件系统的主机应该启动得特别快，这是因为它们不必执行*fsck*。规模较小的文件系统的主机也应该启动得快，因为它们没有太多的文件系统需要检查。

一旦选定了要运行名字服务器的主机，你就要让该主机的IP地址出现在每个总是需要名字服务的主机的*resolv.conf*文件中。你可能还要在最后列出后备名字服务器，因为在正常运行时，主机应该使用最靠近它的名字服务器。这样一来，停电之后，你关键的应用程序还是会有名字服务，不过是要牺牲一点性能的。

应付灾难

当故障发生时，知道该如何应对是很有帮助的。就像在地震时知道躲在坚实的桌子下能帮你免于被倒下来的显示器砸着，就像着火时知道关闭煤气能拯救你的房子。

同样地，当网络出现故障（哪怕只是很小的问题）时，知道如何应对将帮助你保持网络畅通。在此，我们有一些经验和建议。

短时故障（几小时）

如果你的网络与外界（这个“外界”可以是Internet其余的部分，也可以是你公司其余的部分）隔绝，你的名字服务器可能就会开始在名字解析中有问题了。举个例子来说，如果你的域*corp.acme.com*与Acme Internet其余的部分断开了，你可能不能访问你的父（*acme.com*）名字服务器或是根名字服务器。

你可能以为这不会影响到本地域的主机间的通信，但实际上这是会的。比如说，你在一台运行较老版本解析器的主机上输入：

```
% telnet selma.corp.acme.com
```

那么解析器首先会查找域名*selma.corp.acme.com.corp.acme.com*（假定你的主机使用默认搜索列表，我们在第六章中提到过的）。本地名字服务器如果是*corp.acme.com*的权威的话，就会告知没有合适的域名。而接下来就会查找*selma.corp.acme.com.acme.com*。这个可能的域名已经不在*corp.acme.com*区中了，所以查询就被发送到*acme.com*名字服务器。或者说是你的本地名字服务器试着发送查询，然后一直重发，直到超时。

如果你保证解析器查找的第一个域名就是正确的，那么就能避免这个问题。不要敲入：

```
% telnet selma.corp.acme.com
```

而是输入：

```
% telnet Selma
```

或者：

```
% telnet selma.corp.acme.com.
```

（注意结尾的点。）这会导致首先查找*selma.corp.acme.com*。

注意，BIND 4.9及其后续解析器没有这个问题，至少默认的配置没有这个问题。只要域名中有多于一个点的话，4.9和更新版本的解析器就会首先检查这个域名。所以，如果你输入：

```
% telnet selma.corp.acme.com
```

即使它没有结尾的点，首先查找的域名也会是*selma.corp.acme.com*。

如果你仍然使用4.8.3的BIND或更老版本的解析器，你可以利用可自定义的搜索列表，来避免查询本网络外的名字服务器。你可以使用*search*命令来定义一个不包括你父区的域名搜索列表。例如，要绕过*corp.acme.com*这个问题的话，你可以临时地把你主机的搜索列表设置为：

```
search corp.acme.com
```

现在，当用户输入：

```
% telnet selma.corp.acme.com
```

时，解析器会先查找*selma.corp.com.corp.acme.com*（这个是本地名字服务器可以回答的），然后是 *selma.corp.acme.com* 这个正确的域名。像下面这样也可以：

```
% telnet selma
```

较长时间的故障（几天）

如果你的网络已经有相当长一段时间不能连接了，你的名字服务器可能就会有其他一些问题了。如果它们与根名字服务器失去联系已经很长一段时间了，它们就会停止解析其权威区数据之外的查询了。如果辅名字服务器不能访问它们的主名字服务器，它们迟早也会使其区失效。

因为丢失连接而使得你的域名服务变得乱七八糟的话，那保持一份包括有关整个网络或者工作组信息的*/etc/hosts* 文件不失为一个好主意。在最危急的时刻，你可以把 *resolv.conf* 改名为 *resolv.bak*，然后停止运行本地的名字服务器（如果有的话），而只使用 */etc/hosts*。这种做法不怎么好看，但却是你所需要的。

至于辅名字服务器，你可以将一个无法访问其主名字服务器的辅名字服务器暂时配置成主名字服务器。只需编辑 *named.conf*，把 *zone* 语句里的 *type* 子语句中的 *slave* 改为 *master* 就可以了，然后再删除 *masters* 子语句。如果有不止一个同一区的辅名字服务器被断绝了联系，你可以临时配置其中一个作为主名字服务器，然后重新配置其他的来从这个临时的主名字服务器那里加载数据。

另外，你也可以只是增加你的所有辅名字服务器的备份区数据文件中的期满时间，然后再发送信号要求名字服务器重新加载这些文件。

很长时间的故障（几周）

如果很长时间的故障使你与 Internet 失去连接（比如说，1 周或更长）你可能需要人工地与根名字服务器恢复连接，以使一切都能重新开始工作。每个名字服务器都需要偶尔访问一下根名字服务器。这有点儿像治疗：名字服务器需要与一个根名字服务器联系，来重新获取它对外界的认识。

要想在很长的故障时间中提供根名字服务，你可以建立起你自己的根名字服务器，但是这只是暂时的。一旦你重新连接到了Internet，就必须关闭你的临时根服务器。在Internet上最讨厌又害人的就是名字服务器自认为自己是根名字服务器，但其实对大多数顶级域一无所知。其次，是被配置成可以查询的Internet名字服务器返回一组虚假的根名字服务器。

下面就是配置你的根名字服务器要做些什么。首先，你要创建一个*db.root*文件，即根区数据文件。*db.root*文件将被授权给你这个独立网络中最高层的区。比如说，如果*movie.edu*被隔绝于Internet之外，我们要为*terminator*创建一个这样的*db.root*文件：

```
$TTL 1d
. IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                1         ; 序列号
                3h        ; 刷新
                1h        ; 重试
                1w        ; 期满
                1h )      ; 否定TTL

  IN NS terminator.movie.edu. ; terminator是暂时的根

; 我们的根仅知道域movie.edu和我们的两个in-addr.arpa域

movie.edu. IN NS terminator.movie.edu.
           IN NS wormhole.movie.edu.

249.249.192.in-addr.arpa.  IN NS terminator.movie.edu.
                           IN NS wormhole.movie.edu.
253.253.192.in-addr.arpa.  IN NS terminator.movie.edu.
                           IN NS wormhole.movie.edu.

terminator.movie.edu. IN A 192.249.249.3
wormhole.movie.edu.   IN A 192.249.249.1
                      IN A 192.253.253.1
```

然后我们需要在*terminator*的*named.conf*文件中增加相应的配置：

```
// 注释掉线索区
// zone . {
//              type hint;
//                      file "db.cache";
//              };

zone "." {
                type master;
```

```
                file "db.root";
};
```

对于 BIND 4 的 *named.boot* 来说是这样的：

```
; cache    .    db.cache  (注释掉这个 cache 指令)
primary  .   db.root
```

然后，我们需要更新我们所有的名字服务器（除了这台临时增加的新的根名字服务器）上的 *db.cache* 文件，其中要包括这台临时的根名字服务器（最好把老的根线索文件保存起来，我们会在连接恢复后用到它）。

下面是 *db.cache* 文件的内容：

```
.  99999999  IN  NS  terminator.movie.edu.

terminator.movie.edu.  99999999   IN  A  192.249.249.3
```

这会使在故障期间，*movie.edu* 的名字解析仍能进行。而一旦 Internet 的连接恢复了，我们就可以从 *terminator* 的 *named.conf* 文件中删除新的 *zone* 语句，取消线索 *zone* 语句的注释，然后在其他所有的名字服务器上恢复使用原来的根线索文件。

本章内容：

- 何时成为父域
- 该建立多少子域呢？
- 给子域起什么名字
- 如何成为父域：创建子域
- in-addr.arpa 域的子域
- 做个好父域
- 管理到子域的迁移
- 父域的生命期

第九章

担当父域

黛娜是这样给她的孩子洗脸的：首先用一只爪子拎着那个小可怜的耳朵，另一只爪子来回摩擦它的脸，而且还错误地从鼻子开始：就在刚才，像我说的这样，她正在很费劲地给那只小白猫洗脸，而那只小猫很老实地躺着，有时还咕噜咕噜叫着——毫无疑问，它觉得这一切都是为了它好。

一旦你的域达到一定规模，或者是决定将你的域的某些部分的管理权分给机构中的各个组织，你就要把域分成一些子域。在名字空间中，这些子域就是你当前域的孩子，你的域就是父域。如果你将子域的权责（responsibility）授予别的组织，每个子域就从父区中分离出来成为独立的区。我们把对你子域（也就是你的孩子）的管理称为担当父域（parenting）。

做个好父域，首先要合理地划分你的域，为子域选择合适的名字，以及授权子域创建新区。负责任的父域还要很努力地维护它所在区和它孩子们所在区之间的关系，它要确保父域对子域的授权是最新的、正确的。

做个好父域对网络的畅通是至关重要的，特别是当名字服务成为站点间巡游的关键时。对一个孩子区的名字服务器的不正确授权可能导致一个站点总不可达到，而失去到父区的名字服务器的连接会使一个站点不能访问本地区以外的任何主机。

在本章中，我们将讲述我们对何时创建子域这个问题的看法，还将详细谈谈如何创建子域和对子域进行授权。我们还将讨论对父域和子域关系的管理，最后是如何管理将一个很大的域划分成较小子域的过程，使得由此引起的分裂和不便降低到最小程度。

何时成为父域

我们不能告诉你应该什么时候成为父域，但是我们将大胆地给你提供一些指导原则。除了我们列出的这些原因之外，你也许还会发现一些促使你建立子域的因素，不过下面这些是最常见的：

- 需要将域管理授权或分配给许多组织。
- 你的域规模庞大 —— 将它划分成几个子域会使管理变得容易而且会减小域的权威名字服务器的负载。
- 需要通过将主机纳入某些子域来区分它们之间的组织从属关系。

一旦你决定要建立子域，很自然地，接下来你就要问自己该建立多少个子域了。

该建立多少子域呢？

当然，你不能随便地说：“我想创建四个子域。”决定要实现多少个子域实际上就是选择这些子域的组织从属关系。例如，如果你的公司有四个分支机构，你可能就会决定创建四个子域，每一个子域对应一个分支机构。

你应该为每个地区、每个分公司、甚至每个部门都创建一个子域吗？由于DNS的可扩展性，你在选择上有很大的自由度。你能创建几个很大的子域，也能创建许多很小的子域。不过，无论如何选择，都是有利有弊。

授权给一些大的子域，对于父域而言，工作量不会太大，这是因为没有多少授权信息需要跟踪。不过，如果你真的选择较大的子域，那么就需要更大的内存和更快的名字服务器，而且管理也没有分散。例如，如果你的子域是按地区来划分的，那就会使得同一地区中一些相互独立或根本无关的组来共享同一个区和同一个管理。

而向许多较小的子域授权对管理员来说却是件很头疼的事。要保持授权数据的最新状态，就要知道哪些主机运行名字服务器，而它们又是哪些区的权威。每当子域添加新的名字服务器或者子域的名字服务器的地址改变时，这些授权数据就要变化。如果子域都是由不同的人来管理的，那么这就意味着要培训更多的管理员，父域管理员要维护更多的关系，而且组织总的开销也会增大。另一方面，较小的子域更容易管理，而管理也分布得更为广泛，允许对区数据进行更切合实际的管理。

看过这两种选择的利弊，要做出选择似乎很困难。不过实际上，在你的组织中大概会有一个很自然的划分。有的公司按照地区来管理计算机和网络；还有的公司的工作组很分散，相对独立，它们自己管理自己的事务。下面就有一些基本的原则来帮助你正确地划分你的名字空间：

- 不要把你的组织硬塞进一个古怪或不合适的域结构当中。试着将五十个独立的、无关的分公司放到四个区域性的子域中，也许会省去你（作为父区管理员）的工作，但是对你的名声可没有什么好处。分散、独立的运作需要不同的区——这就是 DNS 存在的理由。
- 域的结构应该能反映组织的结构，特别是你的组织的支持结构。如果某些部门运行网络、分配 IP 地址以及管理主机，那么它们就应该管理子域。
- 如果你不能肯定名字空间应该如何组织，那就试着提出一些准则（例如，要创建一个新子域需要有多少主机，组织中的一个组必须提供哪种级别的支持），依照这些准则来决定一个组什么时候能分离出它们自己的子域，只有在需要的时候，才有组织地扩大名字空间。

给子域起什么名字

一旦决定了要创建多少个子域，它们分别和什么相对应之后，你就该为它们选择合适的名字了。最好不要只单方面地决定你的子域的名字，和你未来子域的管理员及其使用者一起来决定会比较礼貌。实际上，如果你愿意，也可以完全由他们自己做决定。

不过，这样也会造成问题。最好是所有的子域都使用相对一致的命名机制。这会使

得无论是子域中的用户,还是你域外的用户都能很容易地猜出或记住你子域的名字,并且判断出某台主机或某个用户在哪个域中。

如果让各个地方自己决定子域的名字,就会导致命名的混乱。有的想使用地理名字,有的又坚持用组织名;有的想用简写,而有的又想用全称。

因而在选择子域名字之前,最好先确定一个命名约定。下面就是一些根据我们的经验提出的建议:

- 在一家经常发生变动的公司里,组织的名称也会频繁地变动。在这种情况下,按照组织来命名子域可能就是个灾难。头一个月 Relatively Advanced Technology 组看上去还很稳定,下个月就并入了 Questionable Computer Systems 组,下个季度它们就都卖给了一家德国的集团企业。与此同时,你就会遇到这样的问题:子域中本来很容易理解的主机名字,过不了多久就失去了意义。
- 地理名比组织名更为稳定,但是有时不如组织名那样好理解。你可能很清楚你的 Software Evangelism Business Unit 是在 Poughkeepsie 或是 Waukegan,但是你公司以外的人很可能根本就不知道它在哪儿(而且还可能不太会拼写这些名字)。
- 不要为了方便而牺牲可读性。两个字母的子域名或许很容易敲,但却很难辨认它代表的到底是什么含义。为什么要将“ Italy ”简写成“ it ”呢?这会和你的 Information Technology 组织弄混淆。而你只需要加上简单的三个字母,使用全称,就能消除这样的二义性。
- 有很多公司使用含义模糊而又不方便的域名。通常会出现这种情况:公司越大,域名就越难理解。但是,我们要尽力使子域的名字更加清晰明了。
- 不要使用现有的或保留的顶级域名作为子域的名字。对你的国际性的子域来说,使用两个字母的国家缩写或许很合情合理,或是为你的网络组织使用像*net*这样的按组织功能划分的顶级域名也很有道理,但这会造成一些非常危险的问题。比如说,将你的通信(Communications)部门的子域命名为*com*可能就会使你无法和顶级域*com*下的主机通信。假设你的*com*子域的管理员将他们新的 Sun 工作站命名为*sun*,而将新的 HP 9000 命名为*hp*(他们还不是最有想像力的人),则你域中任何主机上的用户发给*sun.com*或*hp.com*的邮件都会被送到

你的*com*子域（注1），这是因为你父区的域名可能会出现在某些主机的搜索列表中。

如何成为父域：创建子域

一旦已经决定如何起名，创建子域就很简单了。不过首先，你还是要决定给你的子域多少自治权。如果你都打算创建子域了，却还没有决定好这个问题就有点奇怪了……

到现在为止，我们都是假设如果你要创建一个子域，就是想把它授权给别的组织，也就是要使它从父域中独立出来，成为一个单独的区。总是要这样吗？不一定。

在决定是否要对一个子域授权时，要仔细考虑该子域中的计算机和网络是如何管理的。要把子域授权给一个根本就不管理自己主机或网络的组织是没有任何意义的。例如，在一个很大的公司里，人事部门很可能就不管理自己的计算机：它们是由MIS（Management Information Systems）或IT（Information Technology ——和MIS一样的组织）部门来管理的。所以，当你想为人事部门创建子域时，将该子域的管理授权给他们简直就是浪费力气。

在父区中创建子域

你也可以创建一个子域但是不对它进行授权。怎么样来做呢？只要在父区中创建指向该子域的资源记录。例如，*movie.edu*有一台保存全部雇员和学生记录的数据库主机*brazil*。要把*brazil*加入到*personnel.movie.edu*域中，我们可以在*db.movie.edu*中加入一些记录。

文件*db.movie.edu*的部分内容是：

```
brazil.personnel      IN  A      192.253.253.10
                      IN  MX     10 brazil.personnel.movie.edu.
```

注1： 实际上，不是所有的电子邮件软件都有这个问题，但是一些流行的*sendmail*版本有这个问题。正如我们在第六章中“电子邮件”一节里提到的那样，这完全取决于它的规范化采用的是哪种方式。

```
                        IN  MX     100 postmanrings2x.movie.edu.
employeedb.personnel    IN  CNAME  brazil.personnel.movie.edu.
db.personnel            IN  CNAME  brazil.personnel.movie.edu.
```

现在，用户可以登录到*db.personnel.movie.edu*中来访问雇员数据库了。我们可以把*personel.movie.edu*添加到人事部雇员的PC或工作站的搜索列表中，更加方便他们的使用，这样他们只需要输入 *telnet db* 就可以访问到相应的主机了。

我们还可以使用$ORIGIN控制语句把起始地址改为*personnel.movie.edu*，这样我们自己就可以使用更短的名字，从而使操作更加方便。

文件 *db.movie.edu* 的部分内容是：

```
$ORIGIN personnel.movie.edu.
brazil     IN A     192.253.253.10
           IN MX    10 brazil.personnel.movie.edu.
           IN MX    100 postmanrings2x.movie.edu.
employeedb IN CNAME brazil.personnel.movie.edu.
db         IN CNAME brazil.personnel.movie.edu.
```

如果我们有更多的记录，也可以为它们单独创建一个文件，然后使用$INCLUDE把它包含在 *db.movie.edu* 中，同时也改变起点（origin）。

你有没有注意到，*personnel.movie.edu*没有SOA记录？其实没有必要用SOA记录，因为 *movie.edu* 的SOA 记录指出整个 *movie.edu* 区的权威的起点（start of authority）。由于没有对*personnel.movie.edu*进行授权，它就是*movie.edu*区的一部分。

创建并授权一个子域

如果你决定要授权你的子域 —— 把你的孩子们送出去，让他们到应该去的地方 —— 你所要做的就和上面有些不一样了。现在我们就开始，你可以跟着我们一起做。

要为我们的特效实验室创建一个 *movie.edu* 的新子域。我们为它选择了一个名字 *fx.movie.edu* —— 简短、易懂，也无二义性。因为我们要将 *fx.movie.edu* 授权给实验室的管理员，所以它就应该是一个独立的区。特效实验室中的两台主机 *bladerunner* 和 *outland* 将作为该区的名字服务器（bladerunner 是主名字服务器）。我们选择运行两台名字服务器作为冗余 —— 如果只用一个*fx.movie.edu*名字服务器的话，它就成为单一失败点，一旦故障，整个特效实验室就会与外界隔离。而实验室里也没有多少主机，所以我们认为两个名字服务器也就足够了。

特效实验室是在 *movie.edu* 的新子网 192.253.254/24 上。

文件 */etc/hosts* 的一部分内容是：

```
192.253.254.1 movie-gw.movie.edu movie-gw
# fx primary
192.253.254.2 bladerunner.fx.movie.edu bladerunner br
# fx secondary
192.253.254.3 outland.fx.movie.edu outland
192.253.254.4 starwars.fx.movie.edu starwars
192.253.254.5 empire.fx.movie.edu empire
192.253.254.6 jedi.fx.movie.edu jedi
```

首先，我们创建一个区数据文件，它包括 *fx.movie.edu* 中所有主机的记录。

文件 *db.fx.movie.edu* 的内容如下：

```
$TTL 1d
@  IN  SOA  bladerunner.fx.movie.edu. hostmaster.fx.movie.edu. (
            1       ; 序列号
            3h      ; 刷新
            1h      ; 重试
            1w      ; 期满
            1h )    ; 否定缓存TTL

    IN  NS  bladerunner
    IN  NS  outland

; fx.movie.edu的MX记录
    IN  MX  10 starwars
    IN  MX  100 wormhole.movie.edu.

; starwars处理bladerunner的邮件
; wormhole是movie.edu的邮件分发器

bladerunner  IN  A   192.253.254.2
             IN  MX  10 starwars
             IN  MX  100 wormhole.movie.edu.

br           IN    CNAME    bladerunner

outland      IN  A   192.253.254.3
             IN  MX  10 starwars
             IN  MX  100 wormhole.movie.edu.

starwars     IN  A   192.253.254.4
             IN  MX  10 starwars
             IN  MX  100 wormhole.movie.edu.
```

```
empire       IN  A   192.253.254.5
             IN  MX  10 starwars
             IN  MX  100 wormhole.movie.edu.

jedi         IN  A   192.253.254.6
             IN  MX  10 starwars
             IN  MX  100 wormhole.movie.edu.
```

然后我们创建 *db.192.253.254* 文件：

```
$TTL 1d
@  IN  SOA  bladerunner.fx.movie.edu. hostmaster.fx.movie.edu. (
            1       ; 序列号
            3h      ; 刷新
            1h      ; 重试
            1w      ; 期满
            1h )    ; 否定缓存 TTL

       IN    NS    bladerunner.fx.movie.edu.
       IN    NS    outland.fx.movie.edu.

1      IN    PTR   movie-gw.movie.edu.
2      IN    PTR   bladerunner.fx.movie.edu.
3      IN    PTR   outland.fx.movie.edu.
4      IN    PTR   starwars.fx.movie.edu.
5      IN    PTR   empire.fx.movie.edu.
6      IN    PTR   jedi.fx.movie.edu.
```

请注意，*1.254.253.192.in-addr.arpa* 的 PTR 记录是指向 *movie-gw.movie.edu* 的。我们是特意这样做的。192.253.254.1 是一个路由器，它还连接到其他 *movie.edu* 网络，所以它并不真正属于 *fx.movie.edu* 域，而且没有必要使 *254.253.192.in-addr.arpa* 中所有的 PTR 记录都要映射到同一个区，不过它们应该与其主机的规范名相对应。

接下来，我们为主名字服务器创建一个适当的 *named.conf* 文件：

```
options {
                directory "/var/named";
};

zone "0.0.127.in-addr.arpa" {
                type master;
                file "db.127.0.0";
};

zone "fx.movie.edu" {
                type master;
                file "db.fx.movie.edu";
};
```

```
zone "254.253.192.in-addr.arpa" {
                type master;
                file "db.192.253.254";
};

zone "." {
                type hint;
                file "db.cache";
};
```

下面是相应的 BIND 4 的 *named.boot* 文件的内容：

```
directory     /var/named

primary       0.0.127.in-addr.arpa      db.127.0.0  ; 回送地址
primary       fx.movie.edu              db.fx.movie.edu
primary       254.253.192.in-addr.arpa  db.192.253.254

cache         .                         db.cache
```

当然，如果我们使用 *h2n* 的话，只需运行：

```
%   h2n -d fx.movie.edu -n 192.253.254 -s bladerunner -s outland \
-u hostmaster.fx.movie.edu -m 10:starwars -m 100:wormhole.movie.edu
```

这样就能减少一些输入。*h2n* 会创建基本上一样的 *db.fx.movie.edu*、*db.192.253.254* 和 *named.boot* 文件。

现在，我们需要配置 *bladerunner* 的解析器了。实际上，这可能都不需要创建 *resolv.conf* 文件。如果我们把 *bladerunner* 的主机名设置为它的新域名 *bladerunner.fx.movie.edu*，解析器就可以从这个全称域名得出本地域名。

然后我们启动 *bladerunner* 上的 *named* 进程，同时检查 *syslog* 中的错误。如果 *named* 启动成功，而且 *syslog* 中没有需要特别注意的错误，我们就要使用 *nslookup* 查找几个位于 *fx.movie.edu* 和 *254.253.192.in-addr.arpa* 中的主机：

```
Default Server:  bladerunner.fx.movie.edu
Address:  192.253.254.2

> jedi
Server:  bladerunner.fx.movie.edu
Address:  192.253.254.2

Name:    jedi.fx.movie.edu
Address:  192.253.253.6
```

```
>  set type=mx
>  empire
Server:  bladerunner.fx.movie.edu
Address:  192.253.254.2

empire.fx.movie.edu     preference = 10,
                        mail exchanger = starwars.fx.movie.edu
empire.fx.movie.edu     preference = 100,
                        mail exchanger = wormhole.movie.edu
fx.movie.edu    nameserver = outland.fx.movie.edu
fx.movie.edu    nameserver = bladerunner.fx.movie.edu
starwars.fx.movie.edu   internet address = 192.253.254.4
wormhole.movie.edu      internet address = 192.249.249.1
wormhole.movie.edu      internet address = 192.253.253.1
bladerunner.fx.movie.edu        internet address = 192.253.254.2
outland.fx.movie.edu    internet address = 192.253.254.3
>  ls -d fx.movie.edu
[bladerunner.fx.movie.edu]
$ORIGIN fx.movie.edu.
@                       1D IN SOA       bladerunner hostmaster (
                                        1               ; 序列号
                                        3H              ; 刷新
                                        1H              ; 重试
                                        1W              ; 期满
                                        1H )            ; 最小值

                        1D IN NS        bladerunner
                        1D IN NS        outland
                        1D IN MX        10 starwars
                        1D IN MX        100 wormhole.movie.edu.
bladerunner             1D IN A         192.253.254.2
                        1D IN MX        10 starwars
                        1D IN MX        100 wormhole.movie.edu.
br                      1D IN CNAME     bladerunner
empire                  1D IN A         192.253.254.5
                        1D IN MX        10 starwars
                        1D IN MX        100 wormhole.movie.edu.
jedi                    1D IN A         192.253.254.6
                        1D IN MX        10 starwars
                        1D IN MX        100 wormhole.movie.edu.
outland                 1D IN A         192.253.254.3
                        1D IN MX        10 starwars
                        1D IN MX        100 wormhole.movie.edu.
starwars                1D IN A         192.253.254.4
                        1D IN MX        10 starwars
                        1D IN MX        100 wormhole.movie.edu.
@                       1D IN SOA       bladerunner hostmaster (
                                        1               ; 序列号
                                        3H              ; 刷新
                                        1H              ; 重试
                                        1W              ; 期满
                                        1H )            ; 最小值
```

```
>   set type=ptr
>   192.253.254.3
Server:  bladerunner.fx.movie.edu
Address:  192.253.254.2

3.254.253.192.in-addr.arpa      name = outland.fx.movie.edu

>   ls -d 254.253.192.in-addr.arpa.
[bladerunner.fx.movie.edu]
$ORIGIN 254.253.192.in-addr.arpa.
@            1D IN SOA       bladerunner.fx.movie.edu. hostmaster.fx.movie.edu. (
                                1                       ; 序列号
                                3H                      ; 刷新
                                1H                      ; 重试
                                1W                      ; 期满
                                1H )                    ; 最小值

              1D IN NS          bladerunner.fx.movie.edu.
              1D IN NS          outland.fx.movie.edu.
1             1D IN PTR         movie-gw.movie.edu.
2             1D IN PTR         bladerunner.fx.movie.edu.
3             1D IN PTR         outland.fx.movie.edu.
4             1D IN PTR         starwars.fx.movie.edu.
5             1D IN PTR         empire.fx.movie.edu.
6             1D IN PTR         jedi.fx.movie.edu.
@            1D IN SOA        bladerunner.fx.movie.edu. hostmaster.fx.movie.edu. (
                                1                       ; 序列号
                                3H                      ; 刷新
                                1H                      ; 重试
                                1W                      ; 期满
                                1H )                    ; 最小值


  >  exit
```

结果看上去很合理，所以现在我们就可以很放心地为 *fx.movie.edu* 建立一个辅名字服务器，然后将 *fx.movie.edu* 从 *movie.edu* 中授权出去了。

fx.movie.edu 的一个辅名字服务器

设置 *fx.movie.edu* 的辅名字服务器很简单：从 *bladerunner* 那里拷贝一份 *named.conf*、*db.127.0.0*和*db.cache*的副本，然后再根据第四章中的说明去编辑*named.conf* 和 *db.127.0.0*。

文件 *named.conf* 的内容是：

```
options {
                directory "/var/named";
};

zone "0.0.127.in-addr.arpa" {
                type master;
                file "db.127.0.0";
};

zone "fx.movie.edu" {
                type slave;
                masters { 192.253.254.2; };
                file "bak.fx.movie.edu";
};

zone "254.253.192.in-addr.arpa" {
                type slave;
                masters { 192.253.254.2; };
                file "bak.192.253.254";
};

zone "." {
                type hint;
                file "db.cache";
};
```

等价的 *named.boot* 文件是：

```
directory  /var/named

primary    0.0.127.in-addr.arpa      db.127.0.0
secondary  fx.movie.edu              192.253.254.2  bak.fx.movie.edu
secondary  254.253.192.in-addr.arpa  192.253.254.2  bak.192.253.254
cache      .                         db.cache
```

像 *bladerunner* 那样，只要 *outland* 的 *hostname* 设置为 *outland.fx.movie.edu*，它就也不需要 *resolv.conf* 文件了。

我们再次启动 *named* 并检查 *syslog* 文件中是否有出错信息。如果没有从 *syslog* 中发现什么问题，我们将查找一些位于 *fx.movie.edu* 中的记录。

在 movie.edu 的主名字服务器上

现在所要做的就是将 *fx.movie.edu* 子域授权给 *bladerunner* 和 *outland* 上的 *fx.movie.edu* 的新名字服务器。我们给 *db.movie.edu* 添加合适的 NS 记录。

文件 *db.movie.edu* 的部分内容如下：

```
fx      86400    IN    NS    bladerunner.fx.movie.edu.
        86400    IN    NS    outland.fx.movie.edu.
```

按照 RFC 1034，这两行中的资源记录专用的部分中的域名（*bladerunner.fx.movie.edu* 和 *outland.fx.movie.edu*）都必须使用名字服务器的规范域名。远程的名字服务器希望能沿着授权信息找到该域名对应的一个或多个地址记录，而不是别名（CNAME）记录。实际上，RFC将这个限制扩展到所有包括以域名作为值的资源记录中 —— 所有这些记录都必须指定规范域名。

但是，只添加这两个记录还不够。你发现问题在哪儿了吗？*fx.movie.edu* 之外的名字服务器怎么才能查找 *fx.movie.edu* 中的信息呢？*movie.edu* 的名字服务器会将它指向 *fx.movie.edu* 的权威名字服务器，对吗？是这样的，但是 *db.movie.edu* 中的 NS 记录只给出了 *fx.movie.edu* 名字服务器的名字。外部的名字服务器需要 *fx.movie. edu* 名字服务器的 IP 地址才能向它们发送查询。谁能给它这些地址呢？只有 *fx.movie.edu* 的名字服务器才行。真是个先有鸡还是先有蛋的问题！

解决的方法就是在 *movie.edu* 区数据文件中包括 *fx.movie.edu* 名字服务器的地址。严格来说，这并不是 *movie.edu* 区的一部分，但是为了 *fx.movie.edu* 的授权能正常工作，这是必需的。当然，如果 *fx.movie.edu* 的名字服务器并不在 *fx.movie.edu* 中，这些地址（被称为 glue 记录）就不必要了。外部的名字服务器能通过向其他的名字服务器查询找到它所需要的地址。

所以，加上 glue 记录，我们所添加的记录就如同下面显示的文件 *db.movie.edu* 的部分内容：

```
fx      86400    IN    NS    bladerunner.fx.movie.edu.
        86400    IN    NS    outland.fx.movie.edu.
bladerunner.fx.movie.edu.  86400  IN  A  192.253.254.2
outland.fx.movie.edu.      86400  IN  A  192.253.254.3
```

要确保你没有在文件中添加不必要的 glue 记录。老版本的 BIND（4.9 以前）会把这些记录加载到缓存中，然后将这些地址作为到其他名字服务器的引用。如果这些地址记录中引出的名字服务器改变了 IP 地址，而你又忘记了更新这些 glue 记录，那么你的名字服务器就会继续把这些过时的信息提供给其他名字服务器使用，从而使得名字服务器在查找该授权区中的数据时性能很差，甚至会使它们无法解析这些授权区中的名字。

BIND 4.9及其后续版本的名字服务器会自动忽略任何你已经包括进来的、但是不必要的glue记录，而且还会在主名字服务器的*syslog*中或辅名字服务器的区数据副本中记录它忽略的这些不必要的记录。例如，如果我们为*movie.edu*添加了一个NS记录，它指向一台不在本网络上的名字服务器*ns-1.isp.net*，而且我们误把它的地址包括在*movie.edu*的主名字服务器上的*db.movie.edu*文件中了，在*named*的*syslog*文件中我们会看到如下信息：

```
Aug  9 14:23:41 terminator named[19626]: dns_master_load: db.movie.edu:55:
ignoring out-of-zone data
```

如果我们的主名字服务器是4.9以前版本的名字服务器，在一个新名字服务器的区传送中包含了不必要的glue记录，在备份区数据文件中我们将看到如下信息：

```
; Ignoring info about ns-1.isp.net, not in zone movie.edu
; ns-1.isp.net 258983  IN      A       10.1.2.3
```

注意，那条无关的A记录已经被注释掉了。

还要记住，要保持这些glue记录的及时性。如果*bladerunner*增加了一个新的网络接口，因而也就增加了一个IP地址，那么你就应该在glue数据中再添加一个A记录。

我们可能还想为从*movie.edu*移到*fx.movie.edu*中的主机保留一些别名。比如说，如果我们要把*plan9.movie.edu*移到*fx.movie.edu*中，这个服务器装有重要的公共域特效算法库。我们应该在*movie.edu*下创建一个别名，将以前的名字指向新的名字：

```
plan9           IN      CNAME   plan9.fx.movie.edu.
```

这就使得*movie.edu*之外的人即使使用以前的域名*plan9.movie.edu*也可以访问到这台机器。

你不应该把任何有关*fx.movie.edu*中域名的信息写在*db.movie.edu*文件中。*plan9*这个别名实际上是在*movie.edu*区中的（这个记录的拥有者是*plan9.fx.movie.edu*），所以它属于*db.movie.edu*的。从另一方面来说，一个把*p9.fx.movie.edu*指向*plan9.fx.movie.edu*的别名，是在*fx.movie.edu*区的并且属于*db.fx.movie.edu*。如果你想把一个不属于某一区数据文件的记录加入到该区的区数据文件中，那么一台BIND 4.9或者后续版本的名字服务器就会忽略它，就像前面那个关于不必要的glue记录的例子

中所描述的那样。更老版本的名字服务器可能会把该记录加载到缓存中，甚至还会加载到它的权威数据中，但是因为这些行为的后果是无法预测的，而又在新版本的BIND中被去掉了，所以即使软件没有要求你这么做，你也最好按照正确的方法来处理这些问题。

授权一个in-addr.arpa区

我们差点儿忘了授权*254.253.192.in-addr.arpa*区了！比起授权*fx.movie.edu*来，这似乎有点欺骗的性质，因为我们并不管理这个父区。

首先，我们需要知道*254.253.192.in-addr.arpa*的父区是谁，而谁又在管理它。要想知道这个就要进行一些查找；在第三章中，我们讲过如何来做。

事实上，*in-addr.arpa*区就是*254.253.192.in-addr.arpa*的父区。而你仔细想想，这还是很有道理的。*in-addr.arpa*的管理员没有理由将*253.192.in-addr.arpa*或*192.in-addr.arpa*授权给独立的权威，因为除非192/8或192.253/16是整个的一个CIDR块，否则像192.253.253/24和192.253.254/24这样的网络并不会有什么联系。它们可以由完全无关的组织来管理。

你可能还记得（在第三章中讲到过），*in-addr.arpa*区是由ARIN（the American Registry of Internet Numbers）来管理的。（当然，即使你不记得了，你也可以用nslookup来找到*in-addr.arpa*的SOA记录中的联系地址，就像我们在第三章中所讲的那样。）接下来我们要做的就只是用基于Web的“Modify Tool”（可以从*http://www.arin.net/cgi-bin/amt.pl*获得）去请求注册我们的反向映射区。

添加一个movie.edu的辅名字服务器

如果特效实验室足够大，那么就有必要在192.253.254/24网络的某个地方设置一个*movie.edu*的辅名字服务器。这样一来，来自*fx.movie.edu*中主机的DNS查询大部分都能在本地得到回答。将某个现有的*fx.movie.edu*的名字服务器变成*movie.edu*的辅名字服务器也很合情理，因为这样，我们就能更好地利用现有的名字服务器，而不必再新建一个名字服务器了。

我们决定将*bladerunner*作为*movie.edu*的辅名字服务器。这不会和*bladerunner*作

为 *fx.movie. edu* 的主名字服务器的主要任务相冲突。在理论上，一个名字服务器，只要有足够的内存，就可以是数千个区的权威。而一个名字服务器既可以是某些区的主名字服务器，同时又是其他区的辅名字服务器（注 2）。

对配置的修改很简单：我们在 *bladerunner* 的 *named.conf* 文件中添加一行语句来告诉 *named* 从 *movie.edu* 的主名字服务器 *terminator.movie.edu* 的 IP 地址那里加载 *movie.edu* 区的数据。

文件 *named.conf* 的内容是：

```
options {
     directory "/var/named";
};

zone "0.0.127.in-addr.arpa" {
     type master;
     file "db.127.0.0";
};

zone "fx.movie.edu" {
     type master;
     file "db.fx.movie.edu";
};

zone "254.253.192.in-addr.arpa" {
     type master;
     file "db.192.253.254";
};

zone "movie.edu" {
     type slave;
     masters { 192.249.249.3; };
     file "bak.movie.edu";
};

zone "." {
     type hint;
     file "db.cache";
};
```

注 2： 显然，一个名字服务器不能同时是某一个区的主名字服务器和辅名字服务器。即，对于一个给定的区，名字服务器要么从一个本地区数据文件加载数据（该名字服务器是给定区的主名字服务器），要么从其他名字服务器加载数据（该名字服务器是给定区的辅名字服务器）。

或者，如果你使用BIND 4名字服务器的话，那么*named.boot*文件的内容是：

```
directory     /var/named

primary       0.0.127.in-addr.arpa      db.127.0.0  ; 回送地址
primary       fx.movie.edu              db.fx.movie.edu
primary       254.253.192.in-addr.arpa  db.192.253.254
secondary     movie.edu                 192.249.249.3     bak.movie.edu
cache         .                         db.cache
```

in-addr.arpa域的子域

你不仅能把正向映射域划分成多个子域并且对各个子域授权，而且，如果你的*in-addr.arpa*名字空间足够大，你也可以把它划分成多个子域。通常，你把与你网络相对应的*in-addr.arpa*划分成与你子网相对应的子域。如何划分完全依赖于你拥有的网络类型和你网络的子网掩码。

按字节划分子网

因为电影大学恰好有三个/24（C类）网络，每一个网络对应一个网段，因此也就不需要把这些网络划分成多个子网。但是，我们的姊妹大学：Altered州立大学，有一个B类网络172.20/16。他们的网络用IP地址的第三个和第四个字节来划分子网，即子网掩码是255.255.255.0。他们已经把他们的域*altered.edu*划分成了一组子域，包括*fx.altered.edu*、*makeup.altered.edu*和*foley.altered.edu*。因为每一个部门运行自己的子网（特效部门运行172.20.2/24，化妆部门运行172.20.15/24，Foley部门运行172.20.25/24），他们也打算据此划分*in-addr.arpa*名字空间。

对*in-addr.arpa*子域的授权与对正向映射域子域的授权是一样的。在它们的*db.172.20*区数据文件中，要添加下面这样的NS记录，把这些对应于各个子网的子域授权给各个子域中相应的名字服务器：

```
2     86400    IN    NS    gump.fx.altered.edu.
2     86400    IN    NS    toystory.fx.altered.edu.
15    86400    IN    NS    prettywoman.makeup.altered.edu.
15    86400    IN    NS    priscilla.makeup.altered.edu.
25    86400    IN    NS    blowup.foley.altered.edu.
25    86400    IN    NS    muppetmovie.foley.altered.edu.
```

有几点很重要，请注意：首先，Altered州立大学的管理者只能在所有者名字字段中使用子网的第三个字节，这是由于该文件的默认起点（origin）是 *20.172.in-addr.arpa*。其次，他们在 NS 记录里需要使用名字服务器的全称域名，从而避免附加起点。再者，它们不需要glue地址记录，因为被授权的名字服务器的名字不是以该区的域名结尾的。

不按字节划分子网

对于那些不能按字节来划分子网的网络（如，对/24（C 类）网络划分子网），该怎么办呢？在这样的例子里，你不能根据子网来授权。这就有两种情况：每一个 *in-addr.arpa* 区有多个子网，或每一个子网有多个 *in-addr.arpa* 区。任何一种情况对我们来讲都不那么令人满意。

A 类和 B 类网络

让我们考虑一个例子：/8（A 类）网络 15/8 以子网掩码 255.255.248.0（13 位的子网字段和 11 位的主机字段，或者说 8192 个子网，每个子网有 2048 个主机）划分子网。在这个例子中，子网 15.1.200.0 代表的地址空间是从 15.1.200.0 到 15.1.207.255。因此，在 *15.in-addr.arpa* 的区数据文件 *db.15* 中对这个子域的授权如下所示：

```
200.1.15.in-addr.arpa.    86400    IN    NS    ns-1.cns.hp.com.
200.1.15.in-addr.arpa.    86400    IN    NS    ns-2.cns.hp.com.
201.1.15.in-addr.arpa.    86400    IN    NS    ns-1.cns.hp.com.
201.1.15.in-addr.arpa.    86400    IN    NS    ns-2.cns.hp.com.
202.1.15.in-addr.arpa.    86400    IN    NS    ns-1.cns.hp.com.
202.1.15.in-addr.arpa.    86400    IN    NS    ns-2.cns.hp.com.
203.1.15.in-addr.arpa.    86400    IN    NS    ns-1.cns.hp.com.
203.1.15.in-addr.arpa.    86400    IN    NS    ns-2.cns.hp.com.
204.1.15.in-addr.arpa.    86400    IN    NS    ns-1.cns.hp.com.
204.1.15.in-addr.arpa.    86400    IN    NS    ns-2.cns.hp.com.
205.1.15.in-addr.arpa.    86400    IN    NS    ns-1.cns.hp.com.
205.1.15.in-addr.arpa.    86400    IN    NS    ns-2.cns.hp.com.
206.1.15.in-addr.arpa.    86400    IN    NS    ns-1.cns.hp.com.
206.1.15.in-addr.arpa.    86400    IN    NS    ns-2.cns.hp.com.
207.1.15.in-addr.arpa.    86400    IN    NS    ns-1.cns.hp.com.
207.1.15.in-addr.arpa.    86400    IN    NS    ns-2.cns.hp.com.
```

一个子网中居然有这么多授权信息！

幸运的是，BIND 8.2、BIND 9.1.0 以及后续版本的名字服务器支持一个叫做

$GENERATE的控制语句。该语句可以让我们创建一组只有个别数字不同的资源记录。例如，你只用下面两个 $GENERATE 控制语句就能创建上面我们列出的 16 个 NS 记录：

```
$GENERATE 200-207 $.1.15.in-addr.arpa.  86400  IN  NS  ns-1.cns.hp.com.
$GENERATE 200-207 $.1.15.in-addr.arpa.  86400  IN  NS  ns-2.cns.hp.com.
```

这个语法非常简单：当名字服务器读到这个控制语句时，它就在控制命令第一个参数指定的范围内迭代，用当前迭代体替换跟在第一个参数后面的模板中的每一个美元符号（$）。

C 类网络

下面再举一个 /24（C类）网络的例子，如 192.253.254/24，以子网掩码255.255.255.192 来划分子网。现在，你有一个 *in-addr.arpa* 区 *254.253.192.in-addr.arpa*，它对应多个子网：192.253.254.0/26、192.253.254.64/26、192.253.254.128/26和192.253.254.192/26。如果你想要让不同的组织管理每个子网对应的反向映射信息，那么这将是一个问题。你可用三种方法来解决这个问题，不过每种方法都不是很完美。

方案 1. 第一种方法是将 *254.253.192.in-addr.arpa* 区作为一个单个实体（entity），而不是试图去授权。这种方法要求四个子网管理员之间的协调，或者使用 Webmin（*http://www.webmin.com/webmin*）这样的工具，让四个管理员每个人负责管理自己的数据。

方案 2. 第二种方法是对 IP 地址的第四个字节授权。它比我们前面所示的 /8 授权还糟糕。在文件 *db.192.253.254* 中，对每一个 IP 地址你将需要至少两个 NS 记录，如下所示：

```
1.254.253.192.in-addr.arpa.    86400    IN    NS    ns1.foo.com.
1.254.253.192.in-addr.arpa.    86400    IN    NS    ns2.foo.com.

2.254.253.192.in-addr.arpa.    86400    IN    NS    ns1.foo.com.
2.254.253.192.in-addr.arpa.    86400    IN    NS    ns2.foo.com.

...

65.254.253.192.in-addr.arpa.    86400    IN    NS    relay.bar.com.
65.254.253.192.in-addr.arpa.    86400    IN    NS    gw.bar.com.

66.254.253.192.in-addr.arpa.    86400    IN    NS    relay.bar.com.
```

```
66.254.253.192.in-addr.arpa.    86400    IN    NS    gw.bar.com.

...

129.254.253.192.in-addr.arpa.    86400    IN    NS    mail.baz.com.
129.254.253.192.in-addr.arpa.    86400    IN    NS    www.baz.com.

130.254.253.192.in-addr.arpa.    86400    IN    NS    mail.baz.com.
130.254.253.192.in-addr.arpa.    86400    IN    NS    www.baz.com.
```

与之类似的，一直到 *254.254.253.192.in-addr.arpa*。

实际上，你可以通过使用 $GENERATE 来大量减少你的工作量：

```
$GENERATE 0-63 $.254.253.192.in-addr.arpa  86400  IN  NS  ns1.foo.com.
$GENERATE 0-63 $.254.253.192.in-addr.arpa  86400  IN  NS  ns2.foo.com.

$GENERATE 64-127 $.254.253.192.in-addr.arpa.  86400  IN  NS  relay.bar.com.
$GENERATE 64-127 $.254.253.192.in-addr.arpa.  86400  IN  NS  gw.bar.com.

$GENERATE 128-191 $.254.253.192.in-addr.arpa.  86400  IN  NS  mail.baz.com.
$GENERATE 128-191 $.254.253.192.in-addr.arpa.  86400  IN  NS  www.baz.com.
```

当然，在 *ns1.foo.com* 的 *named.conf* 中，你还会看到：

```
zone "1.254.253.192.in-addr.arpa" {
                type master;
                file "db.192.253.254.1";
};

zone "2.254.253.192.in-addr.arpa" {
                type master;
                file "db.192.253.254.2";
};
```

或者，如果 *ns1.foo.com* 运行的是 BIND 4，那么你会在 *named.boot* 中看到这样一些指令：

```
primary    1.254.253.192.in-addr.arpa    db.192.253.254.1
primary    2.254.253.192.in-addr.arpa    db.192.253.254.2
```

而且在 *db.192.253.254.1* 中只有这样一个 PTR 记录：

```
$TTL 1d
@    IN    SOA    ns1.foo.com.    root.ns1.foo.com.    (
                        1        ; 序列号
                        3h       ; 刷新
                        1h       ; 重试
```

```
                        1w      ; 期满
                        1h )    ; 否定缓存TTL

        IN    NS    ns1.foo.com.
        IN    NS    ns2.foo.com.

        IN    PTR    thereitis.foo.com.
```

注意，PTR记录被连到区的域名，因为区的域名只对应一个IP地址。现在，当某个*254.253.192.in-addr.arpa*名字服务器接收到对*1.254.253.192.in-addr.arpa*的PTR记录的查询时，它将查询者指到*ns1.foo.com*和*ns2.foo.com*，它们将用区中的这条PTR记录来响应。

方案 3. 最后，有一种聪明的技巧不需要对每一个IP地址单独维护一个区数据文件（注3）。负责/24网络的组织为区中每一个域名生成一个CNAME记录，指向新子域中的域名，这些新的子域都被授权到正确的服务器。新子域只能以如下形式表示，如*0-63*、*64-127*、*128-191*、*192-255*，它们表示每个子域反向映射的地址范围。每个子域仅包含新子域名所示地址范围内的PTR记录。

文件*db.192.253.254*的部分内容：

```
1.254.253.192.in-addr.arpa.  IN  CNAME  1.0-63.254.253.192.in-addr.arpa.
2.254.253.192.in-addr.arpa.  IN  CNAME  2.0-63.254.253.192.in-addr.arpa.

...

0-63.254.253.192.in-addr.arpa.    86400    IN    NS    ns1.foo.com.
0-63.254.253.192.in-addr.arpa.    86400    IN    NS    ns2.foo.com.

65.254.253.192.in-addr.arpa. IN  CNAME 65.64-127.254.253.192.in-addr.arpa.
66.254.253.192.in-addr.arpa. IN  CNAME 66.64-127.254.253.192.in-addr.arpa.

...

64-127.254.253.192.in-addr.arpa.    86400    IN    NS    relay.bar.com.
64-127.254.253.192.in-addr.arpa.    86400    IN    NS    gw.bar.com.

129.254.253.192.in-addr.arpa.  IN  CNAME  129.128-191.254.253.192.in-addr. arpa.
130.254.253.192.in-addr.arpa.  IN  CNAME  130.128-191.254.253.192.in-addr. arpa.

...
```

注 3： 我们最早见到这种方法是在CalTech的Glen Herrmansfeldt在新闻组*comp.protocols.tcp-ip.domains*中提出的。现在这已经被纳入RFC 2317中了。

```
128-191.254.253.192.in-addr.arpa.    86400    IN    NS    mail.baz.com.
128-191.254.253.192.in-addr.arpa.    86400    IN    NS    www.baz.com.
```

同样，你也可以用命令 $GENERATE 来简化你的操作：

```
$GENERATE 1-63 $ IN CNAME $.0-63.254.253.192.in-addr.arpa.

0-63.254.253.192.in-addr.arpa.    86400    IN    NS    ns1.foo.com.
0-63.254.253.192.in-addr.arpa.    86400    IN    NS    ns2.foo.com.

$GENERATE 65-127 $ IN CNAME $.64-127.254.253.192.in-addr.arpa.

64-127.254.253.192.in-addr.arpa.    86400    IN    NS    relay.bar.com.
64-127.254.253.192.in-addr.arpa.    86400    IN    NS    gw.bar.com.
```

0-63.254.253.192.in-addr.arpa 的区数据文件 *db.192.253.254.0-63* 可以只包含从 192.253.254.1 到 192.253.254.63 的 IP 地址的 PTR 记录。

文件 *db.192.253.254.0-63* 的部分内容：

```
$TTL 1d
@    IN    SOA    ns1.foo.com.    root.ns1.foo.com.    (
                          1       ; 序列号
                          3h      ; 刷新
                          1h      ; 重试
                          1w      ; 期满
                          1h )    ; 否定缓存 TTL

     IN    NS    ns1.foo.com.
     IN    NS    ns2.foo.com.

1    IN    PTR   thereitis.foo.com.
2    IN    PTR   setter.foo.com.
3    IN    PTR   mouse.foo.com.
...
```

这个设置工作的方式有点欺骗的性质，让我们重新回忆一遍。一个解析器请求 *1.254.253.192.in-addr.arpa* 的 PTR 记录时，它的本地名字服务器就会查询该记录。本地名字服务器会查询一个 *254.253.192.in-arpa* 的名字服务器，并返回一个 CNAME 记录，该记录表明 *1.254.253.192.in-addr.arpa* 实际上是 *1.0-63.254.253.192.in-addr.arpa* 的别名并且有 PTR 记录对应该名字。该响应也包含 NS 记录，它告诉本地名字服务器，*0-63.254.253.192.in-addr.arpa* 的权威名字服务器是 *ns1.foo.com* 和 *ns2.foo.com*。然后，该本地名字服务器就向 *ns1.foo.com* 或者 *ns2.foo.com* 查询 *1.0-63.254.253.192.in-addr.arpa* 的 PTR 记录并得到该 PTR 记录。

做个好父域

既然对*fx.movie.edu*的授权工作已经完成了，我们作为负责任的父域，应该使用*host*来检查这些授权信息。哦，我们还没有给你*host*吗？Unix的*host*版本可以通过匿名FTP从*ftp.nikhef.nl/pub/network/host.tar.z*得到。

要安装一个*host*首先对它解压：

```
%  zcat host.tar.Z | tar -xvf -
```

然后再在你的系统上编译：

```
% make
```

*host*使得检查授权变得很容易。利用*host*，我们可以在你的父区的名字服务器上查询你所在区的NS记录。如果这些看起来都很好的话，你还可以用*host*向列出来的每个名字服务器查询该区的SOA记录。不过这种查询是非递归的，所以被查询的名字服务器不会查询其他名字服务器以找到那个SOA记录。如果名字服务器回答了，那么*host*就会检查回答消息中的*aa*（权威答案）位是否被置位了。如果是的话，那么名字服务器就会检查数据包以确保其中包括有答案。如果这两个条件都满足了，这个名字服务器就被标志为该区的权威。否则，这个名字服务器就不是权威，*host*就会报告一个错误。

为什么要这么紧张不正确的授权信息呢？不正确的授权信息会导致域名解析变得很慢，或是传播那些过时的和错误的根名字服务器信息。当一个名字服务器收到对于它不是权威的区中的数据的查询时，它会尽其所能地为查询者提供有用的信息。这些“有用的信息”是指该名字服务器所知道的距离最近的祖先区的NS记录。（在第八章中，当我们讨论为什么你不应该注册只缓存名字服务器时，曾简单地提到过这个问题。）

举个例子来说，比如*fx.movie.edu*的一个名字服务器错误地收到一个反复查询，请求*carrie.horror.movie.edu*的地址。而它对*horror.movie.edu*区一无所知（除了它可能会放入缓存中的一点内容），但是它的缓存中可能会有*movie.edu*的NS记录，因为那些是它的父名字服务器。因此它就会把这些NS记录返回给查询者。

在这种情况下，这些NS记录能帮助提出查询的名字服务器获取答案。但是实际生

活中，在 Internet 上并不是所有的管理员都能将他们的根线索文件保持最新。如果你的一个名字服务器沿着一个坏的授权，向一个远程的名字服务器查询它根本就没有的记录，看看会发生什么：

```
% nslookup
Default Server:  terminator.movie.edu
Address:  192.249.249.3

> set type=ns
> .
Server:  terminator.movie.edu
Address:  192.249.249.3

Non-authoritative answer:
(root)  nameserver = D.ROOT-SERVERS.NET
(root)  nameserver = E.ROOT-SERVERS.NET
(root)  nameserver = I.ROOT-SERVERS.NET
(root)  nameserver = F.ROOT-SERVERS.NET
(root)  nameserver = G.ROOT-SERVERS.NET
(root)  nameserver = A.ROOT-SERVERS.NET
(root)  nameserver = H.ROOT-SERVERS.NET
(root)  nameserver = B.ROOT-SERVERS.NET
(root)  nameserver = C.ROOT-SERVERS.NET
(root)  nameserver = A.ISI.EDU              - 这三个
(root)  nameserver = SRI-NIC.ARPA           - 名字服务器
(root)  nameserver = GUNTER-ADAM.ARPA       - 不再是根了
```

远程的名字服务器试图帮助我们本地的名字服务器，发送给它当前的根列表。不幸的是，这个远程的名字服务器有问题，返回的 NS 记录是不正确的。而我们的本地名字服务器对此一无所知，将这些数据放入缓存。

注意：在 BIND4.9 以及后续版本的名字服务器中对此有所防范。

向配置错误的 *in-addr.arpa* 名字服务器查询，通常会导致坏的根 NS 记录，这是因为 *in-addr.arpa* 和 *arpa* 域是大多数 *in-addr.arpa* 子域最近的祖先，而名字服务器很少会将 *in-addr.arpa* 或 *arpa* 的 NS 记录放入缓存。（根也很少给出它们，因为根已经直接向低层的子域授权了。）一旦你的名字服务器将坏的根 NS 记录放入缓存，你的名字解析可能就会有问题了。

这些根 NS 记录可能会使你的名字服务器向不再是那个 IP 地址的根名字服务器，或根本就不再存在的根名字服务器查询。如果有一天你的运气很不好的话，那个坏的

NS记录可能会将你指向一个真实存在的，却不是根的名字服务器，而它离你的网络又很近。虽然它返回的不是权威根数据，但是因为它离你的网络很近，你的名字服务器还是会相信它。

使用 host

如果我们上面所说的能使你明白保持授权正确的重要性，你可能就急着要学习如何使用 *host* 来确信你没有出现错误的授权。

第一步是利用 *host* 在你的父区的名字服务器上查询你所在区的 NS 记录，并且保证它们的正确。下面是我们应该如何在 *movie.edu* 的一个名字服务器上检查 *fx.movie.edu* 的 NS 记录：

```
%  host -t ns fx.movie.edu. terminator.movie.edu.
```

如果 NS 记录都是好的，我们将会在输出结果中看到如下显示：

```
fx.movie.edu   NS   bladerunner.fx.movie.edu
fx.movie.edu   NS   outland.fx.movie.edu
```

这说明所有 *terminator.movie.edu* 授权给 *fx.movie.edu* 的 NS 记录都是正确的。

下一步我们将利用 *host* 的“ SOA check ”模式向这些 NS 记录中的每个名字服务器查询 *fx.movie.edu* 区的 SOA 记录。这也会检查响应是不是权威的：

```
%  host -C fx.movie.edu.
```

如果正常的话，结果将会是上面列出的NS 记录，同时列出 *fx.movie.edu* 区的SOA 记录的内容：

```
fx.movie.edu      NS      bladerunner.fx.movie.edu
bladerunner.fx.movie.edu     hostmaster.fx.movie.edu   (1 10800 3600 608400 3600)
fx.movie.edu      NS      outland.fx.movie.edu
bladerunner.fx.movie.edu     hostmaster.fx.movie.edu   (1 10800 3600 608400 3600)
```

如果 *fx.movie.edu* 的某个名字服务器（比如，*outland*）的配置有误，我们将看到如下的显示：

```
fx.movie.edu   NS   bladerunner.fx.movie.edu
fx.movie.edu   NS   outland.fx.movie.edu
```

```
fx.movie.edu SOA record currently not present at outland.fx.movie.edu
fx.movie.edu has lame delegation to outland.fx.movie.edu
```

这表明 *outland* 上的名字服务器正在运行，但是它不是 *fx.movie.edu* 的权威。

如果 *fx.movie.edu* 的某个名字服务器根本没有运行，我们将会看到：

```
fx.movie.edu   NS   bladerunner.fx.movie.edu
bladerunner.fx.movie.edu     hostmaster.fx.movie.edu   (1 10800 3600 608400 3600)
fx.movie.edu   NS   outland.fx.movie.edu
fx.movie.edu SOA record not found at outland.fx.movie.edu, try again
```

在这种情况下，*try again* 信息表明 *host* 向 *outland* 发送了一个查询信息，但是在规定的可接受的时间段内没收到响应。

虽然也能用 *nslookup* 检查 *fx.movie.edu* 的授权，但 *host* 强大的命令行选项能使这项任务变得非常简单。

管理授权

如果特效实验室变大了，我们可能会发觉还需要增加名字服务器。我们在第八章中讨论了如何建立新的名字服务器，而且还谈到了要向父区的管理员发送什么样的信息。但我们没有解释父区需要做些什么。

事实是父区的工作要相对容易一些，特别是子域的管理员发送给它了完整的信息。假设特效实验室要扩展成一个新的网络192.254.20/24，该网络有一批新的高性能图形工作站。其中有一个 *alien.fx.movie.edu*，将作为这个新网络的名字服务器。

fx.movie.edu 的管理员（我们将该域授权给了实验室的人）发给他们父区的管理员（就是我们）一个简短的通知：

```
Hi!

We've just set up alien.fx.movie.edu (192.254.20.3) as a name
server for fx.movie.edu.  Would you please update your
delegation information?  I've attached the NS records you'll
need to add.

Thanks,

Arty Segue
```

```
ajs@fx.movie.edu

----- cut here -----

fx.movie.edu.  86400  IN  NS  bladerunner.fx.movie.edu.
fx.movie.edu.  86400  IN  NS  outland.fx.movie.edu.
fx.movie.edu.  86400  IN  NS  alien.fx.movie.edu.

bladerunner.fx.movie.edu.  86400  IN  A  192.253.254.2
outland.fx.movie.edu.      86400  IN  A  192.253.254.3
alien.fx.movie.edu.        86400  IN  A  192.254.20.3
```

作为*movie.edu*的管理员，我们的工作很简单：在*db.move.edu*中添加NS和A记录。

如果用*h2n*来生成我们的名字服务器数据会怎么样？我们可以把这些授权信息保存在文件*spcl.movie*中，而*h2n*会在*db.movie*文件的最后使用$INCLUDE把这个文件包括进去。

*fx.movie.edu*的管理员最后要做的就是给*noc@netsol.com*（*in-addr.arpa*区的管理员）也发送一条类似的消息，要求将*20.254.192.in-addr.arpa*子域授权给*alien.fx.movie.edu*、*bladerunner.fx.movie.edu*和*outland .fx.movie.edu*。

另外一种管理授权的方法：存根

如果你运行的是BIND 4.9或后续版本的名字服务器，你就不必手工管理授权信息了。BIND 4.9和后续版本的名字服务器支持一个处于试验阶段的特性，称为存根区（stub zone），它使得一台名字服务器可以自动识别有关授权信息的更改。

名字服务器可以作为一个区的存根，定期地对该区的SOA记录、NS记录以及必要的glue A记录进行分散查询。名字服务器利用NS记录对区进行授权，并且SOA记录决定名字服务器查询的频率。现在当一个子域的管理员更改了该子域的名字服务器，他们只用更新自己的NS记录。而其父区的名字服务器将会在刷新周期规定的时间内得到这些更新后的记录。

在*movie.edu*的名字服务器上，我们在*named.conf*中会增加如下内容：

```
zone "fx.movie.edu" {
                type stub;
                masters { 192.253.254.2; };
                file "stub.fx.movie.edu";
};
```

在 BIND 4.9 名字服务器上，我们使用下面这样的指令：

```
stub     fx.movie.edu    192.253.254.2     stub.fx.movie.edu
```

注意，我们需要将*movie.edu*所有的名字服务器都配置成*fx.movie.edu*的存根，这是因为如果*fx.movie.edu*的授权信息改变了的话，它是不会改变*movie.edu*区的序列号的（注 4）。使 *movie.edu* 的所有名字服务器都成为这个子域的存根可以使它们保持同步。

管理到子域的迁移

我们不想骗你 —— 出于某些原因，我们举的这个*fx.movie.edu*的例子并不实际。其中最主要的一个原因就是特效实验室中的主机情况并不会像我们描述的这样。在现实生活中，实验室在开始的时候只会有几台主机，就放在 *movie.edu* 区中。如果有了某个慷慨的捐助，比如 NSF 的赠款或是某个公司的赠品，实验室就会扩大一点，买更多的计算机。迟早，实验室的主机数量就到了要求创建新子域的程度了。不过到这时，许多原来主机在 *movie.edu* 下的名字已经为人们所熟知了。

我们简要地谈到过在父区中用 CNAME 记录（在*plan9.movie.edu*的例子中）来帮助人们适应域中主机的改变。但是如果你要将整个网络或子网移到新的子域中时，又该怎么办呢？

我们建议的策略同样是使用 CNAME 记录，不过要在更大的范围内使用。使用 *h2n* 这样的工具，你能为全体主机都创建 CNAME。这能使用户继续使用被移动主机的原来的域名。而当他们 telnet 或 FTP（或其他什么）到这些主机时，这些命令会报告他们连接到的是 *fx.movie.edu* 中的主机：

```
%   telnet plan9
Trying...
Connected to plan9.fx.movie.edu.
Escape character is '^]'.

HP-UX plan9.fx.movie.edu A.09.05 C 9000/735 (ttyu1)

login:
```

注 4： BIND 9 名字服务器也没有把 NS 记录放入父区，所以它们也不包括在区传送中。

当然，有些用户不会注意到这样细小的变化，所以你应该在公共关系上做些工作，让人们了解到这些改变。

在 *fx.movie.edu* 的主机上运行着老版本的 *sendmail*，我们可能也需要配置邮件系统接受发送到这些新域名的邮件。最近的 *sendmail* 版本在发送邮件之前会使用名字服务器对消息首部中地址里的主机名进行规范化。这就会把一个 *movie.edu* 的别名转换为一个在 *fx.movie.edu* 中的规范域名。但是，如果收信人那边使用的 *sendmail* 还是老版本，不会改变本地主机的域名，我们就必须手工地把地址中的老名字改为新的。这通常会要求改变 *sendmail.cf* 文件中的 *w* 类或 *w* 文件类，参见第五章中“ MX 算法 ”一节。

你要怎么来创建所有这些别名呢？你只需简单地告诉 *h2n* 让它为 *fx.movie.edu* 网络（192.253.254/24 和 192.254.20/24）上的主机创建别名，然后指明（在 */etc/hosts* 文件中）那些主机的新域名。例如，使用 *fx.movie.edu* 主机表，我们可以很容易为 *fx.movie.edu* 中所有的主机都生成 *movie.edu* 中的别名。

文件 */etc/hosts* 中的部分内容如下：

```
192.253.254.1 movie-gw.movie.edu movie-gw
# fx primary
192.253.254.2 bladerunner.fx.movie.edu bladerunner br
# fx secondary
192.253.254.3 outland.fx.movie.edu outland
192.253.254.4 starwars.fx.movie.edu starwars
192.253.254.5 empire.fx.movie.edu empire
192.253.254.6 jedi.fx.movie.edu jedi
192.254.20.3  alien.fx.movie.edu alien
```

h2n 的 *–c* 选项把一个区的域名作为参数。当 *h2n* 在它需要创建数据的网络上找到该区中的任何主机时，它就会为它们在当前区（用 *–d* 选项指定的）中创建别名。所以通过运行：

```
%  h2n -d movie.edu -n 192.253.254 -n 192.254.20 \
-c fx.movie.edu -f options
```

（这里 *options* 包括用于从其他 *movie.edu* 网络创建数据的其他命令行选项），我们就可以为所有 *fx.movie.edu* 中的主机创建位于 *movie.edu* 下的别名。

删除父区中的别名

虽然使用父区中的别名能减小移动主机所带来的影响，但是就像拐杖，它们也会限制你的自由。每当你想要实现一个子域的时候，你原打算会使你的父区变小，然而，现实情况却是使你父区的名字空间变得很混乱。而且这会使得你父区中的主机不能和子域中的主机同名。

一段时间之后（这应该通知你的用户）你就该删除所有的别名，当然，如果有的Internet主机已经为人所熟知了，也可以保留一些。在这段时间之内，用户应该适应新的域名，修改脚本、*.rhost* 文件以及类似的东西。但是不要总是将所有的别名都留在父区中，它们违反了DNS的初衷，因为这使得你和你子域的管理员不能自主地为主机命名。

为了避免无法连接所带来的影响，你可能想要保持那些为人所熟知的Internet主机或重要的网络资源的CNAME记录。另一方面，你或许根本就不用把它们放到子域中，最好就留在父区里。

*h2n*的*-c*选项能使你非常方便地删除那些你创建的别名，即使子域中主机的记录混在主机表中，或者在同一网络上但是作为其他区的主机上。而*-e*选项以一个区的域名作为参数，然后告诉*h2n*要排除网络上所有包含该域名的记录，否则它就会为它们创建数据。例如，下面这个命令行就会删除早先创建的所有*fx.movie.edu*主机的CNAME记录，同时也会为*movie-gw.movie.edu*（在网络192.253.254上）创建一个A记录：

```
%  h2n -d movie.edu -n 192.253.254 -n 192.254.20 \
-e fx.movie.edu -f options
```

父域的生命期

作为父域，其实有许多东西要理解，所以让我们再来扼要讲述一下已经讲过的内容中的一些要点。通常，作为一个父域的生命周期是这样的：

1. 你只有一个区，所有的主机都在其中。
2. 你将你的区分成一些子域，如果有必要，有的子域还可以放在一起作为父区。你为迁移到子域中的为人所熟知的主机在父区中添加一些CNAME记录。

3. 一段时间之后，你要删除保留的 CNAME 记录。

4. 你要手工或使用存根区来处理子域授权的更新工作，还要定期检查授权。

好了，现在你已经了解了如何做一个父域。下面就让我们来谈一些更高级的名字服务器特性。你可能还要利用它们来使你的孩子们都能正常地工作。

第十章

高级特性

本章内容：

- 地址匹配列表和 ACL
- DNS 动态更新
- DNS NOTIFY
- 增量区传送（IXFR）
- 转发
- 视图
- 循环分配
- 名字服务器地址排序
- 更喜欢使用特定的名字服务器
- 非递归名字服务器
- 避免使用伪装的名字服务器
- 系统优化
- 兼容性
- IPv6

小蚊子说：“ 如果你叫他们的名字，
他们却不回答你，那要名字
还有什么用呢？ ”

最新的BIND 名字服务器版本 8.1.2 和 9.1.0 有大量的新特性，其中最突出的就是支持DNS 动态更新、异步区变动通知（简称NOTIFY）以及增量区传送。剩下的最重要的是和安全有关的：它们允许你配置名字服务器可以回答谁的查询，可以向谁提供区传送，以及允许从哪里获得动态更新。对于公司内部网络来说，大部分安全特性都没有必要，但是其他机制对于任何名字服务器的管理员来说都很有用。

在本章中，我们会讨论这些特性，并且对于如何在DNS基础设施中很方便地使用它们提出些建议。（我们把有关防火墙的内容放在下一章讨论。）

地址匹配列表和 ACL

不过，在我们介绍新的特性之前，最好说说地址匹配列表。在 BIND 8 和 9 中，几乎每个安全特性，甚至一些与安全毫无关系的特性都使用到了地址匹配列表。

地址匹配列表的每一项指定一个或多个IP地址。该列表中的元素可以是独立的IP地址、IP前缀或是一个已命名的地址匹配列表（待会儿我们还会更详细地讨论它）（注1）。IP 前缀的格式如下：

```
network in dotted-octet format/bits in netmask
```

例如，网络 15.0.0.0 的网络掩码是 255.0.0.0（连续 8 位 1），可以写成 15/8。从传统角度来说，这是一个可称为 15 的“ A 类 ”网络。另外，包括 IP 地址从 192.169.1.192 到 192.168.1.255 的网络则可以写成 192.168.1.192/26（也就是说，192.168.1.192 这个网络的网络掩码为 255.255.255.192，是连续 26 位 1）。下面是包括这两个网络的地址匹配列表：

```
15/8; 192.168.1.192/26;
```

已命名的地址匹配列表就是有名字的地址匹配列表。要想在另一个地址匹配列表中使用已命名的地址匹配列表，必须在被引用之前用一个 *acl* 语句在 *named.conf* 中对它进行定义。*acl* 语句的语法非常简单，如下所示：

```
acl name { address_match_list; };
```

有了这个定义以后，这个名字就等价于对应的地址匹配列表。虽然该语句的名字是 *acl*（access control list，访问控制列表），但还是可以在任何能接受地址匹配列表的地方使用已命名的地址匹配列表，包括一些和访问控制无关的地方。

任何时候当你想在几个访问列表中使用一个或多个相同的项时，用 *acl* 语句给它们起个名字使之相关联是个不错的主意。然后，你就可以在地址匹配列表中引用这个名字了。例如，让我们把15/8称为“ HP-NET ”，而把192.168.1.192/26称为“ internal ”：

```
acl "HP-NET" { 15/8; };

acl "internal" { 192.168.1.192/26; };
```

现在，我们就可以在其他地址匹配列表中用名字来引用这些列表了。这不仅可减少输入量，而且可增加文件 *named.conf* 的可读性。

注 1： 如果你运行的是 BIND 9，那么地址匹配列表可以包括 IPv6 地址和 IPv6 前缀。这部分将在本章后面进行描述。

我们非常小心地把ACL的名字封装在引号中，以避免和BIND的保留字发生冲突。如果你确信ACL名字不会和保留字发生冲突，那就不必使用引号。

下面是四个预定义名字的地址匹配列表：

none
: 没有任何IP地址

any
: 所有的IP地址

localhost
: 本地主机（也就是运行名字服务器的主机）的任一IP地址

localnets
: 本地主机任一网络接口所在的网络(用网络掩码屏蔽掉每个网络接口的IP地址中的主机位后得到的)

DNS动态更新

Internet世界（通常指TCP/IP网络）已经逐渐变成一个越来越动态的空间。许多大的公司都利用DHCP动态分配IP地址。几乎所有的ISP也是利用DHCP来给拨号用户分配IP地址。为了保持这种特性，就要求DNS也能够动态地添加和删除记录。在RFC 2136中引入了这种机制，称为DNS动态更新（DNS Dynamic Update)。

BIND 8和9也支持RFC 2136中说明的动态更新机制。这就允许被授权的更新者添加和删除以某个名字服务器为权威的区中的资源记录。更新者可以通过获取一个区的NS记录来发现该区的权威名字服务器。如果该名字服务器收到了一个已授权的更新消息，但是它不是该区的主名字服务器，它就会把这个消息向上转发给它的主名字服务器，这个过程称为“更新转发”。依次，如果下一个名字服务器仍然是该区的辅名字服务器，它继续向它的主名字服务器转发这个消息。毕竟，只有该区的主名字服务器才具有对区数据可写的权限；所有的辅名字服务器直接或者间接（通过其他辅名字服务器）地从主名字服务器得到区数据文件的副本。一旦主名字服务器完成了动态更新并修改了区数据文件，那么辅名字服务器就可以通过区传送得到一个新的副本。

动态更新不仅仅允许简单地添加和删除记录。更新者还能添加或删除单独的资源记录，删除 Rrsets［指一群有着相同域名、类（class）和类型（type）的资源记录，例如 *www.movie.edu* 的所有 Internet 地址］，或者甚至是删除所有与指定域名有关的记录。更新还可以以区中特定的记录是否存在为先决条件，条件满足更新才会生效。例如，只在 *armageddon.fx.movie.edu* 这个域名当前未被使用或 *armageddon.fx.movie.edu* 当前还没有地址记录时，更新才添加下面的地址记录：

```
armageddon.fx.movie.edu.  300  IN  A  192.253.253.15
```

注意：有关更新转发有几点需要注意的地方：9.1.0以前版本的BIND名字服务器并没有实现更新转发，所以，如果使用 9.1.0 以前版本的 BIND 名字服务器，在试图更新时一定要保证更新消息是发向该区主名字服务器的，这一点非常重要。可以通过检查该区 SOA 记录的MNAME字段来确信这个名字服务器就是该区的主名字服务器。大多数动态更新例程也是利用 MNAME 字段来暗示要把该更新消息送往哪个权威名字服务器。

对大部分情况来说，动态更新功能是由那些像 DHCP 服务器这样的程序使用的，它们要自动为计算机分配IP地址，并且注册由此引起的从地址到名字和从名字到地址的映射。一些程序使用新的 *ns_update()* 解析器例程来创建更新消息，并发送给包含该域名的区的权威服务器。

不过，也可以使用命令行程序 *nsupdate* 手工创建更新，该程序是包含在标准 BIND 软件包中的。*nsupdate* 读入一行命令，然后将其转换为一个更新消息。该命令可以从标准输入（默认方式）读入，也可以从一个文件中读入，如果是后者，那么文件的名字必须作为 *nsupdate* 的参数。只要有空间，那些没有用空行隔开的命令会被合并成一个更新消息。

下面是 *nsupdate* 能理解的命令：

prereq yxrrset domain name type [rdata]

把由 *domain name* 指定的某个域名中存在由 *type* 指定的特定类型的 RRset 作为执行后续 *update* 命令的先决条件。如果指定了 *radata*，那么它必须存在。

prereq nxrrset

把由 *domain name* 指定的某个域名中不存在由 *type* 指定的特定类型的 RRset 作为执行指定 *update* 命令的先决条件。

prereq yxdomain domain name

把存在指定的域名作为进行更新的先决条件。

prereq nxdomain

把不存在指定的域名作为进行更新的必需条件。

update delete domain name [type] [rdata]

删除指定的域名，或者如果指定了 *type* 的话就删除指定的 RRset，或者如果也指定了 rdata 的话，那就删除匹配指定的 *domain name*、*type* 和 *rdata* 的记录。

update add domain name ttl [class] type rdata

把指定的记录加入到区中。注意，这里除类型和与资源记录有关的数据之外，TTL 也必须包括在内，但是类（class）是可选的，默认值是 IN。

所以，例如命令：

```
%  nsupdate
>   prereq nxdomain mib.fx.movie.edu.
>   update add mib.fx.movie.edu. 300 A 192.253.253.16
>
```

就告诉服务器只有在该域名已不存在时才为 *mib.fx.movie.edu* 添加一个地址。注意，最后的空行是暗示 *nsupdate* 要发送这个更新。狡猾吗？

下面的命令：

```
%  nsupdate
>   prereq yxrrset mib.fx.movie.edu. MX
>   update delete mib.fx.movie.edu. MX
>   update add mib.fx.movie.edu. 600 MX 10 mib.fx.movie.edu.
>   update add mib.fx.movie.edu. 600 MX 50 postmanrings2x.movie.edu.
>
```

是检查 *mib.fx.movie.edu* 是否已经有了 MX 记录，如果有的话就删除这些 MX 记录，再重新添加两个取而代之。

在进行动态更新时仍然有一些限制：不能完全删除一个区（虽然能删除除了 SOA 记录和 NS 记录以外的所有东西），并且不能添加新的区。

动态更新和序列号

当名字服务器进行动态更新时，它对区进行修改并且必须增加区序列号以通知该区的辅名字服务器做相应更新。当然这会自动实现。不过，并不是每一次动态更新都需要增加序列号。

在 BIND 8 名字服务器中，它都会使增加序列号推迟 5 分钟或者发生了 100 次更新才增加序列号。推迟增加序列号的目的是为了解决名字服务器处理动态更新的能力和处理区传送的能力之间不匹配的问题：对于较大的区，进行区传送可能需要较长的时间。当名字服务器最终增加序列号的时候，它会发出一个 NOTIFY 声明（在本章后面将予以详细叙述），通知该区的辅名字服务器序列号已经改变。

在 BIND 9 中，一旦进行了动态更新就会增加序列号。

动态更新和区数据文件

由于动态更新对区数据文件实施永久性的改变，这就需要在磁盘上进行记录。但是，每次重写一个区数据文件都需要从区中添加或者删除一个记录，这对于名字服务器来讲相当费力。写区数据文件要花费时间，而我们可以想像一个名字服务器每秒将会收到数十个甚至上百个动态更新。

所以，当 BIND 8 和 9 名字服务器收到动态更新时，它们只是在日志文件（log file）中添加一个简短的更新记录（注 2）。当然，这种更新会立刻在位于内存的区数据文件的副本中生效。但是只有等到一定的时间间隔（通常是 1 小时）后，名字服务器才会将整个区数据文件备份到磁盘上。这时，BIND 8 名字服务器会立即删除日志文件，因为不再需要它了。（这时内存中的区数据文件的副本和磁盘上的区数据文件是一样的。）而在 BIND 9 名字服务器中，会保留这些日志文件，因为在增量区传送中仍会用到，我们会在本章的后面讲述这个问题。（BIND 8 名字服务器将增量区传送信息保存在另一个文件里。）

在 BIND 8 名字服务器中，日志文件的名字是在区数据文件名字后加上 *.log*。而在 BIND 9 名字服务器中，日志文件（也叫 *journal* 文件）的名字是区数据文件名字加上 *.jnl*。所以首次使用动态更新的时候，看见这些文件是很正常的，不要感到奇怪。

注 2： 这个办法对那些使用 journal 文件系统的人来说是相当熟悉的。

在 BIND 8 名字服务器中，正常退出名字服务器或者每隔 1 小时，日志文件会消失（虽然它们可能还会很快再次出现，如果名字服务器收到许多更新的话）。在 BIND 9 名字服务器中，这些日志文件根本不会消失。如果日志文件存在的话，当名字服务器启动时，这两种名字服务器都会把其中改变的记录加入到区中。

要是你感兴趣，BIND 8 的日志文件是人工可读的，包括下面这些条目：

```
;BIND LOG V8
[DYNAMIC_UPDATE] id 8761 from [192.249.249.3].1148 at 971389102 (named pid 17602):
zone:   origin movie.edu class IN serial 2000010957
update: {add} almostfamous.movie.edu. 600 IN A 192.249.249.215
```

不幸的是，BIND 9 的日志文件无论如何也不是我们能读懂的。

更新访问控制列表

很明显，动态更新给了更新者对区的可怕的控制权利，如果要使用它们的话，无疑需要限制它们。默认情况下，BIND 8 和 9 名字服务器都不允许对权威区进行动态更新。为了使用它们，要在希望允许动态更新的区的 *zone* 语句中添加 *allow-update* 或 *update-policy* 子语句。

allow-update 以一个地址匹配列表作为参数。只有匹配该列表的地址才会允许更新这个区。要很谨慎，使这个匹配列表尽可能限制严格：

```
zone "fx.movie.edu" {
    type master;
    file "db.fx.movie.edu";
    allow-update { 192.253.253.100; }; // 就是我们的 DHCP 服务器
};
```

带 TSIG 签名的更新

因为 BIND 9.1.0 及其后续版本的辅名字服务器能够转发更新，那么基于 IP 地址的访问控制列表又有什么用呢？如果主名字服务器允许从它的辅名字服务器地址更新，那么不管更新的源发送者是谁，转发更新都是允许的。这并不好（注 3）。

注 3： BIND 9.1.0 还提醒你使用基于 IP 地址的访问控制列表是不安全的。

首先，你能够控制哪一个更新被转发。*allow-update-forwarding* 子语句以地址匹配列表为参数。只有从那些和地址匹配列表匹配的IP 地址发来的更新才被转发。下面的 *zone* 语句只是转发来自特效部门子网的更新：

```
zone "fx.movie.edu" {
    type slave;
    file "bak.fx.movie.edu";
    allow-update-forwarding { 192.253.254/24; };
};
```

尽管如此，在使用更新转发的时候你还是要用带 TSIG 签名的动态更新。现在你需要知道的就是带 TSIG 签名的动态更新具有签名者的加密签名，在第十一章中我们将更深入地介绍TSIG。如果这些更新被转发，其中的签名也随之被转发。一旦签名被认证，就可以知道签署该更新的密钥的名字。这些密钥的名字看上去像一个域名，并且它们经常就是密钥所在主机的域名。

在 BIND 8.2 及其后续版本的名字服务器中，地址匹配列表可以包含一个或者多个 TSIG 密钥的名字：

```
zone "fx.movie.edu"
    type master;
    file "db.fx.movie.edu";
   allow-update { key dhcp-server.fx.movie.edu.; }; // 仅允许由DHCP 服务器的TSIG 密钥签名的更新
};
```

这就允许用TSIG密钥*dhcp-server.fx.movie.edu*签署的更新对*fx.movie.edu*进行任何的更改。不幸的是，我们没有办法进一步限制拥有该TSIG 密钥的更新者的源IP 地址。

BIND 9支持一个同样基于TSIG签名的粒度更好的访问控制机制，该机制使用新的 *update-policy zone*语句。*update-policy*可以让你具体设定哪些密钥允许更新区中的哪些记录。由于辅名字服务器只是转发更新，所以该机制只是对主区有意义。

更新是由签署它的密钥的名字和它试图更新的记录的域名与类型决定的。*update-policy* 的语法如下所示：

```
(grant | deny) identity nametype name [types]
```

*grant*和 *deny*的含义很明显：允许或不允许指定的动态更新。*identity* 指的是用来对更新签名的密钥的名字。*nametype* 是下面所述的选项中的一个：

name

当被更新的域名跟 *name* 字段所指定的域名相同时用该选项。

subdomain

当被更新的域名是 *name* 字段中名字的子域（也就是被更新的域名以 *name* 字段中的名字结尾）时使用该选项。

wildcard

当被更新的域名和 *name* 字段所指定的域名的通配符表达式相匹配时使用该选项。

self

当被更新的域名和 *identity*（而非 *name*）字段所指的名字相同时，也就是和用来签署该更新的密钥的名字一致时，使用该选项。如果 *nametype* 选择 *self*，那么 *name* 字段将被忽略。即使这样看上去有些冗余（后面例子中将会看到），但是当 *nametype* 为 *self* 时，我们仍然必须使用 *name* 字段。

很显然，*name* 项是符合指定的 *nametype* 的域名。例如，如果 *nametype* 为 *wildcard*，那么 *name* 字段还应该包括一个 *wildcard* 标号。

types 字段是一个可选项，除了 NXT 以外，它可以是任何有效的记录类型（如果是多个类型，就用空格分开）。（ANY 一个非常方便的缩写，指除了 NXT 以外的所有类型。）如果你不写 *types*，那么它将匹配除了 SOA、NS、SIG 和 NXT 以外的所有类型。

注意：有关 *update-policy* 规则的优先次序：在 *update-policy* 子语句中，应用于动态更新的是第一个匹配，而非最接近的匹配。

所以，如果 *mummy.fx.movie.edu* 使用一个名为 *mummy.fx.movie.edu* 的密钥签署它的动态更新，我们就可以利用下面的语句限制 *mummy.fx.movie.edu* 只更新自己的记录：

```
zone "fx.movie.edu" {
     type master;
     file "db.fx.movie.edu";
     update-policy { grant mummy.fx.movie.edu. self mummy.fx.movie.edu.;
};
```

```
};
```

或者用下面的语句来限制它只更新它自己的地址记录：

```
zone "fx.movie.edu" {
     type master;
     file "db.fx.movie.edu";
     update-policy { grant mummy.fx.movie.edu. self mummy.fx.movie.edu. A; };
};
```

通常我们可以用下面的语句来限制我们所有的用户只能更新它们自己的地址记录：

```
zone "fx.movie.edu" {
    type master;
    file "db.fx.movie.edu";
    update-policy { grant *.fx.movie.edu. self fx.movie.edu. A; };
};
```

下面是一个更复杂的例子：该操作允许所有的用户改变除SRV记录以外的任何记录（条件是这些记录的名字和密钥的名字相同），不过还允许 *matrix.fx.movie.edu* 更新和 Windows 2000 相关的 SRV 记录（在 *_udp.fx.movie.edu.cn*、*_tcp.fx.movie.edu*、*_sites.fx.movie.edu* 和 *_msdcs.fx.movie.edu* 中），具体使用的语句如下：

```
zone "fx.movie.edu" {
     type master;
     file "db.fx.movie.edu";
     update-policy {
          deny *.fx.movie.edu. self *.fx.movie.edu. SRV;
          grant *.fx.movie.edu. self *.fx.movie.edu. ANY;
          grant matrix.fx.movie.edu. subdomain _udp.fx.movie.edu. SRV;
          grant matrix.fx.movie.edu. subdomain _tcp.fx.movie.edu. SRV;
          grant matrix.fx.movie.edu. subdomain _sites.fx.movie.edu. SRV;
          grant matrix.fx.movie.edu. subdomain _msdcs.fx.movie.edu. SRV;
     };
};
```

因为*update-policy*子语句中的规则是根据它们出现的次序来衡量的，所以用户不能更改它们的 SRV 记录，不过它们可以更新其他任何它们自己的记录。

```
grant identity subdomain fx.movie.edu
```

与

```
grant identity wildcard *.fx.movie.edu
```

的区别是：前者允许由 *identity* 指定的密钥修改与 *fx.movie.edu* 有关的记录（例如，区的 NS 记录），而后者不允许。

如果你想利用带 TSIG 签名的动态更新，而又没有能发送这种更新消息的软件，那么你可以使用较新版本的 *nsupdate*。欲了解更多的信息，请参阅下一章。

DNS NOTIFY（区变动通知）

传统上，BIND 辅名字服务器采用轮询（polling）机制来决定何时需要区传送。轮询间隔也被称为刷新间隔（refresh interval）。区的 SOA 记录中的其他参数控制轮询机制的其他方面。

但是使用这种轮询机制，辅名字服务器只有在刷新间隔到了以后才会检查它的主名字服务器，并传送新的区数据。在动态更新环境中，这种时延可能会引起混乱。如果主名字服务器在区中信息改变之后能告诉辅名字服务器，那该多好啊！毕竟主名字服务器总是会知道数据已被改变：有人重新装载了数据，服务器可以通过检查所有区数据文件的 *mtime* 值（Unix 文件最后被修改的时间），以了解哪些文件被修改了（注 4）；或者它收到并处理了一个动态更新。处理了重载和更新以后，主名字服务器就会发出通知，而不必等到刷新间隔到期。

RFC 1996 建议了一种机制，它允许主名字服务器通知它的辅区数据已被改变。BIND 8 和 9 就实现了这一机制，简称为 DNS NOTIFY。

DNS NOTIFY 是这样工作的：当一个主名字服务器发现区的序列号已经改变后，它就会向该区所有的辅名字服务器发出一个特殊声明。它确定哪些名字服务器是该区的辅名字服务器的方法是：查看该区所有的 NS 记录，然后删除区的 SOA 记录中 MNAME 字段指向的名字服务器（译注 1）以及本地主机，剩下的就是辅名字服务器。

什么时候名字服务器才会发现一个更改？重启主名字服务器会导致向该区当前序列

注 4： 除了只有一个区重载的情况，当名字服务器检查数据文件的 *mtime* 时，只有这个区被重载。

译注 1： 通常这就是该区的主名字服务器。

号的全部辅名字服务器发送通知，因为主名字服务器在重启之前无法知道它的区数据文件是否编辑了。用新的序列号重载一个或多个区会引发名字服务器向那些区的辅名字服务器发送通知。并且，导致区的序列号增加的动态更新也会引发通知的发送。

这个特殊的NOTIFY声明是通过DNS首部中的操作码来标识的。大多数查询的操作码是QUERY。NOTIFY消息，包括声明和响应，有一个特殊的操作码NOTIFY。除此之外，NOTIFY消息看上去就非常像对区SOA记录的查询消息：它指定序列号已经改变的区的域名、类和SOA类型。权威回答位也置1了。

当辅名字服务器收到来自它的主名字服务器的某个区的NOTIFY声明时，它就发送一个NOTIFY响应。这个响应告诉主名字服务器这个辅名字服务器已经收到了NOTIFY声明，这样主名字服务器就可以停止发送该区的NOTIFY声明了。然后辅名字服务器就像该区的刷新计数器到期一样执行下列操作：它向主服务器查询，请求那个主名字服务器声称已经改变了的区的SOA记录。如果序列号较高，辅名字服务器就传送该区。

为什么辅名字服务器不只是简单地相信主名字服务器，认为该区已被改变呢？这是因为有可能有些人给我们的辅名字服务器发送假冒的NOTIFY，造成许多不必要的区传送，从而导致对我们的主名字服务器的拒绝服务（denial of service）攻击。

RFC 1996认为，如果辅名字服务器确实传送了区，它就应该向区中其他的权威名字服务器发送它自己的NOTIFY声明。这样做是由于主名字服务器也许不能够亲自通知该区所有的辅名字服务器，因为有的辅名字服务器不能直接和主名字服务器通信（这样的辅名字服务器以其他辅名字服务器作为自己的主名字服务器）。不过，与BIND 8.3.2和BIND 9不同，较早版本的BIND 8没有这样做。较早的BIND 8的辅名字服务器不会发送NOTIFY消息，除非被明确地配置成要这样做。

在实际情况中是如下这样工作的：在我们的网络上，*terminator.movie.edu*是*movie.edu*的主名字服务器，而*wormhole.movie.edu*和*zardoz.movie.edu*是辅名字服务器，如图10-1所示。

当我们在*terminator.movie.edu*上编辑、重载或者更新*movie.edu*时，*terminator.movie.edu*会向*wormhole.movie.edu*和*zardoz.movie.edu*发送NOTIFY声明。两个辅名字服务器都会响应*terminator.movie.edu*并告诉它已经收到通知。然后它们就检

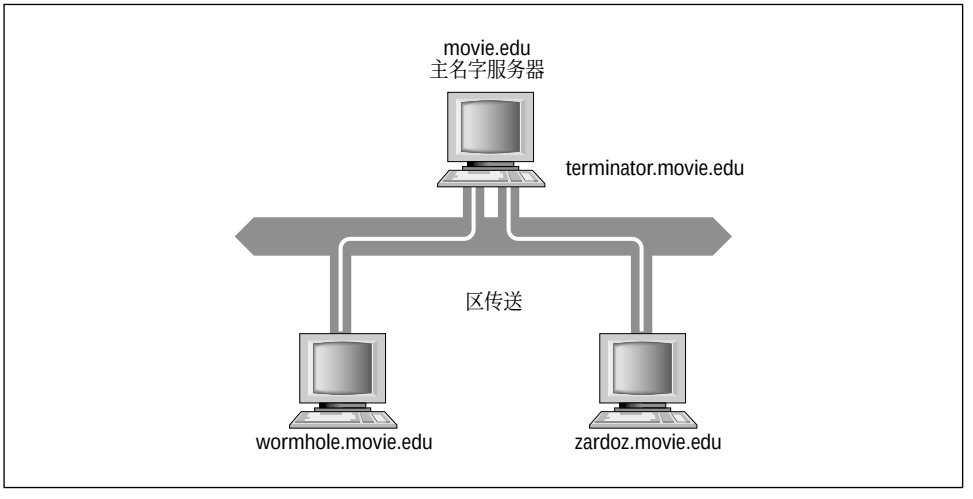


图 10-1 movie.edu 区传送示例

查*movie.edu*的序列号是否已被增加，如果确实如此，就进行区传送。如果*wormhole.movie.edu* 和 *zardoz.movie.edu* 运行的是 BIND 8.2.3 或者 BIND 9，在传送区的新版本之后也会彼此发送 NOTIFY 声明，告诉对方发生了变化。但是如果对于*movie.edu*来讲*wormhole.movie.edu*不是*zardoz.movie.edu*的主名字服务器，反之亦然，这两个辅名字服务器就会忽略对方的 NOTIFY 声明。

BIND 8 会把 NOTIFY 消息写到系统日志文件（syslog）中。下面是我们重载了 *movie.edu* 以后 *terminator.movie.edu* 记录的日志文件：

```
Oct 14 22:56:34 terminator named[18764]: Sent NOTIFY for "movie.edu IN SOA
2000010958" (movie.edu); 2 NS, 2 A
Oct 14 22:56:34 terminator named[18764]: Received NOTIFY answer (AA) from 192.249.
249.1 for "movie.edu IN SOA"
Oct 14 22:56:34 terminator named[18764]: Received NOTIFY answer (AA) from 192.249.
249.9 for "movie.edu IN SOA"
```

第一个消息显示的是 *terminator.movie.edu* 发送的 NOTIFY 声明，通知两个辅名字服务器（2 NS）现在 *movie.edu*的序列号是 2000010958。下面两行表示两个辅名字服务器已经确认收到通知。（BIND 9 通常并不自动记录 NOTIFY 动作。）

让我们再来看一个更为复杂的区传送的例子。*a* 是某个区的主名字服务器，同时也是 *b* 的主名字服务器，而 *b* 是 *c* 的主名字服务器，而且 *b* 有两个网络接口，如图 10-2 所示。

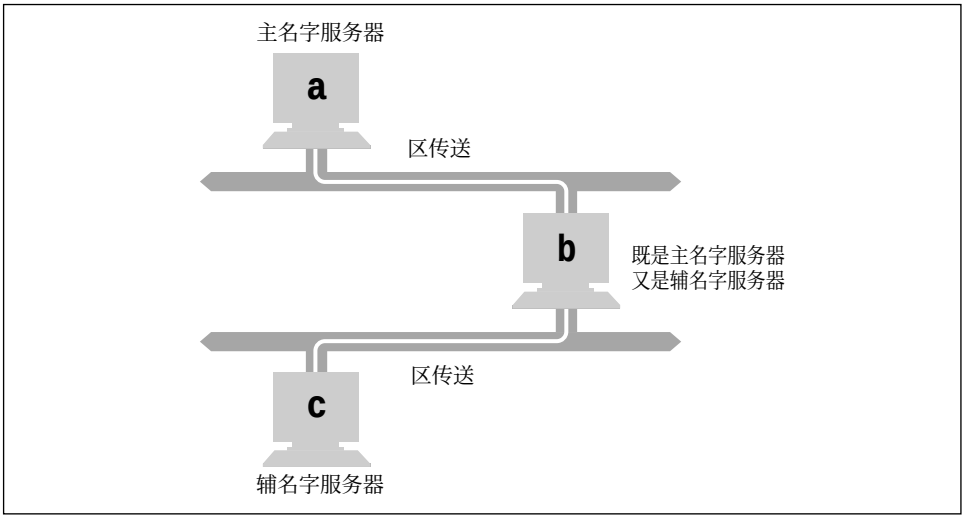


图 10-2　复杂的区传送示例

在这种情况下，当区被更新之后，*a* 通知 *b* 和 *c*。*b* 检查区的序列号是否增加了，如果是的，它就进行区传送。而因为 *c* 配置中的主名字服务器并不是 *a*（而是 *b*），所以 *c* 忽略 *a* 的 NOTIFY 消息。如果 *b* 运行的是 BIND 8.2.3 或 BIND 9 或者被明确地配置成要通知 *c*，那么当 *b* 的区传送完成之后，它就向 *c* 发送一个 NOTIFY 消息，提示 *c* 检查 *b* 中该区的序列号。如果 *c* 也运行的是 BIND 8.2.3 或 BIND 9，在完成区传送以后它也会给 *b* 发送一个 NOTIFY 声明，当然 *b* 会忽略掉该声明。

还要注意，如果有任何可能会导致 *c* 从 *b* 的另外一个网络接口那里收到 NOTIFY 声明，*c* 就必须在区的 *masters* 子语句中配置两个网络接口，否则 *c* 会忽略掉来自未知接口的 NOTIFY 声明。

BIND 4 的辅名字服务器和其他一些不支持 NOTIFY 的名字服务器，会返回一个 Not Implemented（NOTIMP，没有实现）错误。注意，微软的 DNS Server 支持 DNS NOTIFY。

在 BIND 8 和 9 中，DNS NOTIFY 默认是打开的，但是可以使用下面的语句全部关闭该功能：

```
options {
      notify no;
};
```

也可以针对某个区打开或关闭该功能。例如，假设我们知道 *fx.movie.edu* 区所有的辅名字服务器都运行的是 BIND 4，根本不支持 NOTIFY 声明。那么下面的 *zone* 语句就不会向 *fx.movie.edu* 的辅名字服务器发送没有意义的 NOTIFY 声明：

```
zone "fx.movie.edu" {
     type master;
     file "db.fx.movie.edu";
     notify no;
};
```

针对特定区的 NOTIFY 设置会覆盖任何有关该区的全局设置。不幸的是，BIND 8 和 9 都不允许关掉服务器对服务器的 NOTIFY 声明。

BIND 8 和 9 甚至提供了一个功能可以允许你在“ NOTIFY 列表 ”中添加那些除了区的 NS 记录中规定的服务器之外的其他服务器。例如，你可能有一台或多台没有注册的辅名字服务器(第八章中提到过的)，而你仍然希望它们能很快地获得改变后的新数据。或者你所配置的服务器是较早版本的 BIND 8，它是一个区的辅名字服务器，但又是另一个辅名字服务器的主名字服务器，所以它还需要给那个辅名字服务器发送 NOTIFY 消息。

可以使用 *zone* 语句的 *also-notify* 子语句在你的 NOTIFY 列表中添加一个服务器：

```
zone "fx.movie.edu" {
    type slave;
    file "bak.fx.movie.edu";
    notify yes;
    also-notify { 15.255.152.4; }; // 这是一个 BIND 8 辅名字服务器，它必须被
                                   // 显式地配置以通知它的辅名字服务器
};
```

在 BIND 8.2.2 及其后续版本的名字服务器中，你也可以把 *also-notify* 作为 *options* 子语句。这应用到所有 NOTIFY 处于开状态（而又没有自己的 *also-notify* 子语句）的区。

从 BIND 9.1.0 开始，可以把 *explicit* 作为 *notify* 子语句的一个参数，这可以限制 NOTIFY 消息只传送给 *also-notify* 列表中的名字服务器。也可以使用 *allow-notify* 子语句通知你的名字服务器接受来自除了该区配置的主名字服务器以外的名字服务器的 NOTIFY 消息：

```
options {
```

```
        allow-notify { 192.249.249.17; }; // 使192.249.249.17发送NOTIFY消息
    };
```

作为一个 *options* 子语句，*allow-notify* 可以对所有的辅区起作用。当作为 *zone* 子语句时，它只覆盖该区所有的全局 *allow-notify*。

增量区传送（IXFR）

有了动态更新和 NOTIFY，我们的区就可以随着网络状态的改变而不断地更新，并把这种改变迅速地传播到这些区的所有权威服务器。这是不是就足够了呢？

并不完全。设想一下，你运行着一个很大的区，它以非常快的速度进行着更新。可以想像，一开始你就有一个很大的区，包括上千个用户，突然一下想运行 Windows 2000 和 DHCP，那会是个什么样子。现在每个用户都要更新它在区中的地址记录，同时域控制器（Domain Controller）也要更新记录，告诉用户它们运行着什么服务。（在第十六章中，我们将详细介绍 Windows 2000。）

每次主名字服务器收到更新并增加区的序列号时，它都会给它的辅名字服务器发送 NOTIFY 声明。而每次收到 NOTIFY 声明的时候，辅名字服务器都会检查它们的主名字服务器上该区的序列号，如果可能的话就进行区传送。如果区很大的话，区传送就需要一定的时间，在这段时间间隔内可能会有其他的更新到达；这样的话你的辅名字服务器就可能永远处于区传送状态！至少，即使区的改变非常小（比如，只是添加一个用户的地址记录），你的名字服务器仍然需要很长的时间对整个区进行传送。

增量区传送（Incremental Zone Transfer，简称 IXFR）可以解决这个问题。它允许辅名字服务器告诉其主名字服务器它当前使用的区的版本，并仅仅请求传送它使用的版本和当前版本之间改动过的部分。这样就大大减少了区传送的大小和所需的时间。

增量区传送请求的查询类型为 IXFR，而不是 AXFR（进行整个区传送的查询类型），并且在信息的权威段包含了辅名字服务器所维护的该区的当前 SOA 记录。当主名字服务器收到一个增量区传送请求的时候，它就查找该区的辅名字服务器和主名字服务器所使用的区版本之间发生改变的记录。如果该记录丢失，就传送整个区，否则就只是传送它们使用的区版本之间的变动。

IXFR 的局限性

上面所讲的听起来的确很好，但是 IXFR 也存在一些局限性你需要知道。首先，在 BIND 8.2.3 以前的版本，IXFR 运行不是很正常。所有的 BIND 9 都支持 IXFR 并运行得很好，同时也可以和 BIND 8.2.3 进行互操作。

其次，在你只用动态更新修改你的区数据文件的情况下，IXFR 运行得最好。动态更新保留区的改变记录以及相应的序列号的改变 —— 这也就是主名字服务器要向请求 IXFR 的辅名字服务器发送的内容。但是一个重载过所有区数据文件的 BIND 主名字服务器不能得出当前区和原来区之间的差异。得到完整的区传送的辅名字服务器也不能得出当前区和以前区之间的差异。

这意味着，如果你想最大限度地利用 IXFR，你最好只是用动态更新来修改你的区，而不是手工编辑区数据文件。

IXFR 文件

BIND 8 名字服务器除了保留动态更新日志文件以外，还单独保留一个区变动的 IXFR 日志文件。与动态更新日志文件一样，每次名字服务器收到更新 IXFR 日志文件都会被更新。不同的是，虽然名字服务器能被配置成当 IXFR 日志文件超过一定大小时对它进行删减，但从不会删除它。在 BIND 8 中 IXFR 日志文件的默认名字是区数据文件的名字后面加上扩展名 *.ixfr*。

BIND 9 名字服务器使用动态更新日志文件（log file 或者 journal file），来汇集 IXFR 响应并维护区的完整性。因为主名字服务器不知道自己何时会用到某一个区的变更记录，所以它不会删除日志文件。如果 BIND 9 的辅名字服务器收到一个区的 AXFR 请求，它就会删除日志文件，因为收到 AXFR 以后会进行一个新的完整区传送，就不再需要保留对上一个完整区传送进行改动的情况了。

BIND 8 的 IXFR 配置

在 BIND 8 上配置 IXFR 非常简单。首先，在你的主名字服务器上，要有一个叫做 *maintain-ixfr-base* 的 *options* 子语句，它告诉服务器为所有的区维护 IXFR 日志文

件，甚至包括以该名字服务器作为辅名字服务器的区，这是因为这些区可能也有辅名字服务器会要求 IXFR：

```
options {
    directory "/var/named";
    maintain-ixfr-base yes;
};
```

然后，应该告诉你的辅名字服务器向主名字服务器请求IXFR。可以用新的*server*子语句 *support-ixfr* 实现这个功能：

```
server 192.249.249.3 {
     support-ixfr yes;
};
```

如果你想对主名字服务器上的IXFR日志文件重命名，可以使用一个新的*zone*子语句 *ixfr-base*:

```
zone "movie.edu" {
     type master;
     file "db.movie.edu";
     ixfr-base "ixfr.movie.edu";
};
```

另外，你还可以配置名字服务器在IXFR日志文件超过一定的大小时对它进行修整（注5）：

```
options {
    directory "/var/named";
    maintain-ixfr-base yes;
    max-ixfr-log-size 1M;     // 修整IXFR到1MB
};
```

一旦IXFR日志文件超过限定的大小100KB，名字服务器就会修整它到指定大小以下。这个100KB的缓冲防止日志文件在达到限定大小后每次连续的更新都需要进行修整。

使用*many-answers*区传送格式可以使区传送更有效。详细情况请看后面“ 更有效的区传送 ”一节。

注 5： 在 BIND 8.2.3 之前，你需要指定字节数，但并不是“ 1M ”，这是由于有一个 bug。

BIND 9 的 IXFR 配置

在 BIND 9 的主名字服务器上配置 IXFR 更简单，因为你不必做任何事情，一切都是默认的。如果想对某个特殊的辅名字服务器关掉 IXFR 功能（你可能不会真的那么做，因为辅名字服务器必须请求增量区传送），可以使用 *provide-ixfr* 这个 *server* 子语句，它的默认值是 *yes*:

```
server 192.249.249.1 {
     provide-ixfr no;
};
```

也可以使用 *provide-ixfr* 作为 *options* 子语句，这样就可以把它应用到所有 *server* 语句中没有自己明确说明的 *provide-ixfr* 子语句的辅名字服务器上。

BIND 9的主名字服务器默认情况下就是发送 *many-answers* 区传送的，所以不必进行特殊的 *transfer-format* 配置。

BIND 9 中更有用的是 *request-ixfr* 子语句，它可以在 *options* 或者 *server* 语句中使用。如果你有一个混合使用 IXFR-capable 和 IXFR-impaired 的主名字服务器，你就可以使你的辅名字服务器的区传送请求和主名字服务器相匹配：

```
options {
    directory "/var/named";
    request-ixfr no;
};

server 192.249.249.3 {
    request-ixfr yes;     // 在我们的主名字服务器中，只有 terminator
                          // 支持 IXFR
};
```

BIND 9 不支持 *max-ixfr-log-size* 子语句。

转发

某些网络连接不鼓励向本地以外发送很大的数据流量，这要么是因为网络连接是按流量计费的，要么是因为网络连接是低速、高延迟的，就像远程办公室使用卫星连接到公司的网络一样。在这些情况下，你可能想将发往外部的DNS流量限制到尽可能小。BIND 提供了一种处理这种问题的机制：转发器（forwarder）。

如果你想避免到某一特定的名字服务器的名字解析，转发器也是非常有用的。例如，如果你的网络上只有一台主机可以连接到Internet，你在这台主机上运行了名字服务器，那么你就可以把它配置成其他名字服务器的转发器使它们也可以查找 Internet 域名。（转发器的这种应用将在第十一章讨论防火墙的时候给予更多的阐述。）

如果你指定本地的一个或多个服务器作为转发器，那么你的名字服务器将会把所有发往外部的查询都先送往转发器。这就意味着由转发器来处理所有本地产生的发往外部的查询，因而可以建造一个丰富的信息的缓存。对于任何指定的对远程区的查询，很可能转发器就能从它的缓存中找到答案，避免了其他服务器再向外部发送查询。如果要将一个名字服务器作为转发器，不用对它本身做什么修改，你只用修改本地其他的服务器，将它们的查询指向转发器即可。

当主或辅名字服务器被配置成使用转发器时，它的操作方式有一点细微的变化。如果解析器请求的记录已经在名字服务器的权威数据或缓存数据中了，它就用该信息作为回答，这一部分操作没有变化。不过，如果记录不在数据库中，名字服务器将向转发器发送查询，在继续正常操作之前，它会等待一段很短的时间，如果没有答案，就自己同远程名字服务器联系。这里名字服务器所做的不同之处在于它向转发器发送的是一个递归查询，期待它找到答案。而在其他时候，名字服务器向其他名字服务器发送的都是非递归查询，并处理收到的响应，这些响应指引它到另外的名字服务器。

例如，下面是 *movie.edu* 中 BIND 8 和 9 名字服务器的 *forwarders* 子语句，它等价于 BIND 4 启动文件中的指令。其中 *wormhole.movie.edu* 和 *terminator.movie.edu* 都是该网络的转发器。这个 *forwarders* 语句要被添加到除了转发器本身所在服务器以外的其余每个名字服务器的配置文件中去：

```
options {
    forwarders { 192.249.249.1; 192.249.249.3; };
};
```

等价的 BIND 4 指令为：

```
forwarders 192.249.249.1 192.249.249.3
```

使用转发器的时候，要尽量使本地的配置保持简单。配置太复杂会把你自己弄晕的。

警告：避免使转发器形成一条链。不要将服务器A配置成转发给服务器B，而将服务器B配置成转发给服务器C（或者更糟糕，让它转发回服务器A）。这种配置将引起很长的解析延迟并且很脆弱，链中任何一个转发器出错都会损坏或者中断名字解析。

一种更受限制的名字服务器

你可能想进一步限制你的名字服务器，甚至要阻止它们在转发器关机或没有响应时去联系其他的外部服务器。可以通过把服务器配置成“只转发”（forward-only）模式来实现。只转发模式的名字服务器是使用转发器的名字服务器的一个变种。它仍然根据它的权威数据和缓存中的数据来响应查询。但是，它完全依赖于它的转发器；它不会在转发器无法响应时试图去联系其他服务器以获得所需数据。下面是一个只转发模式的名字服务器的配置文件需要包括的语句：

```
options {
    forwarders { 192.249.249.1; 192.249.249.3; };
    forward only;
};
```

在BIND 4名字服务器上看起来应该是这样的：

```
forwarders 192.249.249.1 192.249.249.3
options forward-only
```

BIND 4.9之前的名字服务器提供了相同的功能，但是使用的是*slave*指令而不是*options forward-only*指令：

```
forwarders 192.249.249.1 192.249.249.3
slave
```

不要把这里“slave”的老用法和现在的用法混淆了。在BIND 4名字服务器中“slave”和“forward-only”是同义的。而现在它意味着一个从主名字服务器那里通过区传送来获得区数据的名字服务器。

要使用只转发模式，就要配置转发器，否则是没有意义的。如果你将一个服务器配置成只转发模式，并且运行的是BIND 8.2.3以前版本的BIND，你可能要把转发器的IP地址多写几遍。在BIND 8服务器上看起来就像下面这样：

```
options {
```

```
        forwarders { 192.249.249.1; 192.249.249.3;
                192.249.249.1; 192.249.249.3; };
        forward only;
};
```

在 BIND 4 服务器上则是：

```
forwarders 192.249.249.1 192.249.249.3 192.249.249.1 192.249.249.3
options forward-only
```

这个名字服务器同每个转发器只联系一次，然后等待一段很短的时间以获得响应。把转发器重复列出几次会告诉只转发服务器重复发送查询给转发器，这样就增加了这个只转发服务器等待从转发器来的响应的总的时间。

不过，你必须要问你自己是否真的需要使用只转发模式的服务器。这样的服务器完全依赖于它的转发器。其实，根本不用这样的服务器，你也完全能获得同样的效果；你可以创建*resolv.conf*文件，其中包含*nameserver*指令，它指向你正在使用的转发器。这样的话你仍然是依赖转发器，但是现在你的应用程序是直接查询转发器，而不用让只转发名字服务器来代表应用程序查询它们。你可能不会有只转发服务器才有的本地缓存内容，以及地址排序，但是你可以通过运行较少的名字服务器来减少本地配置的整体复杂度。

转发区

传统上，使用转发器是一个全有或者全无的方法：使用转发器解析你的名字服务器自己不能回答的所有查询，或者根本不使用转发器。但是更多的情况下，你希望对转发有更多的控制。例如，或许你希望用特定的转发器解析特定的域名，而对其他域名的解析使用迭代的方式。

BIND 8.2 引入了一种新的特性 —— 转发区（forward zone），它允许你把名字服务器配置成只有查找特定的域名时才使用转发器。（BIND 9 从 9.1.0 才开始有转发区的功能。）例如，你可以使你的服务器将所有对以*pixar.com*结尾的域名的查询都转发给 Pixar 的两个名字服务器：

```
zone "pixar.com" {
    type forward;
    forwarders { 138.72.10.20; 138.72.30.28; };
};
```

为什么这样配置而不是直接让你的名字服务器按授权依次去访问 *com* 和 *pixar.com* 呢？假设你和Pixar之间有一个私有连接，并且你可以利用一组只能从你的网络到达的特殊的名字服务器来解析所有 *pixar.com* 的域名，你就该像上面那样配置。

即使转发规则是在*zone*语句中指定的，它们也将会应用到所有以指定域名结尾的域名上。也就是说，不管你要查找的域名 *foo.bar.pixar.com* 是不是在 *pixar.com* 区中，因为它们是以 *pixar.com* 结尾的（或者说是在 *pixar.com* 域里的），所以都会应用该规则。

还有一种转发区设置，和我们刚才讲的刚好相反。它允许你指定什么样的查询将不被转发。因此该特性只适用于在*options*语句中指定了转发器的名字服务器，通常该 *options* 语句将应用于所有的查询。

使用zone语句配置这些转发区，而不是使用*forward*类型。使用*forwarders*子语句，这些区都是正常的区 —— master、slave 或 stub。为取消在 *options* 语句中的转发配置，我们指定一个空的转发器列表：

```
options {
    directory "/var/named";
    forwarders { 192.249.249.3; 192.249.249.1; };
};

zone "movie.edu" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.movie.edu";
    forwarders {};
};
```

等一下 —— 为何你需要在你是权威的区里禁止转发？你难道不是自己回答查询而不使用转发器吗？

记住，转发规则对以该区域名结尾的所有域名查询都有效。因此该转发规则只对在 *movie.edu* 中被授权了的子域（如，*fx.movie.edu*）中的域名查询有效。没有该转发规则的话，该名字服务器将向在 192.249.249.3 和 192.249.249.1 上的名字服务器转发请求 *matrix.fx.movie.edu* 的查询。有了该规则，名字服务器会使用来自 *movie.edu* 区的子域的 NS 记录，并直接查询 *fx.movie.edu* 的名字服务器。

转发区在处理 Internet 防火墙时是非常有用的，我们将在下一章看到这一点。

转发器选择

在 BIND 8.2.3 名字服务器上，不需要多次列出转发器。这些名字服务器不需要按照列出来的顺序查询转发器；它们将在列表中的名字服务器解释成“候选”转发器，并且按照往返时间来选择首先查询的转发器，其中往返时间是转发器应答上次查询的时间。

如果转发器，特别是列表中的第一个转发器出现了故障，这就会非常有用了。老版本的BIND将继续盲目地查询出现故障的转发器，并且在查询列表中的下一个转发器前一直等待。BIND 8.2.3很快就认识到转发器未应答，并尝试另一个。

不幸的是，尽管BIND 9可以在需要时重传查询，它也未实现这个更加智能的转发器选择方式。

视图

BIND 9 引入了另一个在防火墙环境中非常有用的机制 —— 视图（view）。视图允许你为一组主机提供一种名字服务器配置，而为另一组主机提供另一种不同的配置。如果运行你的名字服务器的主机收到的查询既来自内部主机，也来自 Internet 上的主机（我们将在下一章介绍），这会是特别方便的。

如果不配置任何视图的话，BIND 9 会自动创建一个单一的、隐含的视图，所有访问它的主机看到的都是这个视图。要明确地创建一个视图，要使用 *view* 语句，该语句用视图名作为参数：

```
view "internal" {
};
```

虽然视图名可以是任何表述，不过最好还是用一个描述性的名称。尽管不需要引用视图名，但是最好尽量避免与 BIND 自己使用的保留字（例如，“internal”）冲突。尽管 *view* 语句不需要紧跟在 *options* 语句后，但必须放在 *options* 语句后面。

利用 *match-clients* *view* 子语句，可以选择哪些主机能看到某个视图，这个子语句用

地址匹配列表作为参数。如果不用 *match-clients* 指定一组主机，那么该视图将应用于所有主机。

比如说，在我们的名字服务器上创建 *fx.movie.edu* 的一个特殊视图，我们只想让特效部门看到这个视图。我们能够创建一个只让我们子网上的主机看到的视图：

```
view "internal" {
     match-clients { 192.253.254/24; };
};
```

如果想让语句更有可读性，可以使用 *acl* 语句：

```
acl "fx-subnet" { 192.253.254/24; };

view "internal" {
     match-clients { "fx-subnet"; };
};
```

因为不能在视图内使用 *acl* 语句，必须确信在视图之外定义了 ACL。

能将什么放在 *view* 语句之内呢？几乎任何除了 ACL 以外的东西。可以用 *zone* 语句定义区，用 *server* 语句描述远程名字服务器，以及用 *key* 语句配置 TSIG 密钥。在 *view* 语句中，能够使用大多数 *options* 子语句，但是如果使用 *options* 子语句，不要将那些子语句封装在 *options* 语句中；在 *view* 语句中，要独立使用它们：

```
acl "fx-subnet" { 192.253.254/24; };
view "internal" {
    match-clients { "fx-subnet"; };
    recursion yes;  // 打开本视图的递归设置
                    // 在 options 语句中，全局是关闭的
};
```

对于匹配 *match-clients* 的主机，在视图里你所指定的任意配置都将覆盖同名的全局选项（例如，在 *options* 语句中的选项）。

要想了解你所运行的 BIND 9 版本的 *view* 语句中都支持哪些东西（因为会随版本而变化），请查看 BIND 发布中的 *doc/misc/options* 文件。

下面是特效实验室完整的 *named.conf* 文件，可帮助你理解视图的作用：

```
options {
    directory "/var/named";
```

```
};

acl "fx-subnet" { 192.253.254/24; };

view "internal" { // 我们区的内部视图

    match-clients { "fx-subnet"; };

    zone "fx.movie.edu" {

        type master;
        file "db.fx.movie.edu";
    };
    zone "254.253.192.in-addr.arpa" {
        type master;
        file "db.192.253.254";
    };
};

view "external" {  // 相应于世界的其余部分，我们区的视图

    match-clients { any; };  // 隐式地
    recursion no;            // 在我们的子网外面，它们不应该请求递归查询

    zone "fx.movie.edu" {
        type master;
        file "db.fx.movie.edu.external";   // 外部区数据文件
    };

    zone "254.254.192.in-addr.arpa" {
        type master;
        file "db.192.253.254.external";   // 外部区数据文件
    };
};
```

注意，每个视图都有一个 *fx.movie.edu* 区和一个 *254.253.192.in-addr-arpa* 区，但是区数据文件在内部和外部视图里是不同的。这允许我们向外部展示与内部看到的不同的数据。

因为主机 IP 地址匹配的第一个视图决定了该主机所能看见的视图，所以 *view* 语句的次序很重要。如果在配置文件中首先列出的是外部视图，因为外部视图匹配所有地址，所以它将使内部视图不发生作用。

关于视图，最后要注意的是（至少在下一章我们使用视图之前要注意的最后一点）：哪怕只配置了一个 *view* 语句，你的所有 *zone* 语句都必须出现在显示的视图里。

循环分配

BIND 4.9以后发布的名字服务器规范早已存在于以前BIND的一些补丁中的负载分配功能中。Bryan Beecher 在他给 BIND 4.8.3 写的补丁中实现了他称为“混洗（shuffle）地址记录”的记录。这是些特殊类型的地址记录，名字服务器响应时会轮转地给出。例如，如果域名*foo.bar.baz*有三个“混洗的”IP地址：192.168.1.1、192.168.1.2 和 192.168.1.3，一个打了适当补丁的名字服务器就会首先以下面的顺序列出：

```
192.168.1.1 192.168.1.2 192.168.1.3
```

然后是：

```
192.168.1.2 192.168.1.3 192.168.1.1
```

然后是：

```
192.168.1.3 192.168.1.1 192.168.1.2
```

然后再是第一种，不停地重复这样的循环。

如果你有许多相当的网络资源，例如，镜像FTP服务器、Web服务器或终端服务器，而你又希望让负载分布于它们之间，那么这个功能非常有用。你可以建立一个指向一组资源的域名，将客户机配置成访问这个域名，名字服务器将请求分布在列出的这些 IP 地址中。

BIND 4.9 及其后续版本将混洗地址记录作为一种单独的记录类型，服从特殊的处理。与之相反，如果一个域名有不止一个 A 记录，现代的名字服务器就会轮转提供这些地址。（事实上，只要给定的域名有多个某种类型的记录，名字服务器都会轮转使用，注 6）所以下面的记录：

```
foo.bar.baz.    60    IN    A    192.168.1.1
foo.bar.baz.    60    IN    A    192.168.1.2
foo.bar.baz.    60    IN    A    192.168.1.3
```

注 6： 实际上，在 BIND 9 之前，PTR 记录是不会轮转给出的，BIND 9 则会轮转所有类型的记录。

在4.9及其后续版本的名字服务器上与在打了补丁的4.8.3服务器上的混洗地址记录所实现的功能是一样的。BIND 的文档称之为循环法（round robin）。

减小记录的生存时间也是一个好办法，就像我们在这个例子中所做的那样。这就保证了如果地址保存在一个不支持循环法的中间名字服务器的缓存中，它们也会很快地在内存中超时。如果中间名字服务器需再次查找这个名字，你的权威名字服务器就会再次对地址进行循环。

注意，这实际上只是负载分配，而不是负载平衡，因为名字服务器按照完全确定的方式来给出地址，而不考虑实际的负载或服务器服务请求的能力。在我们的例子中，地址 192.168.1.3 的服务器是一个运行 Linux 的 486DX33，而其他两个服务器是 HP9000 Superdomes，可是运行 Linux 的服务器还是会得到三分之一的查询量。即使把性能好的服务器地址多列出来几次也没有任何作用，因为 BIND 会自动除去那些重复的记录。

多 CNAME 记录

在 BIND 4 名字服务器鼎盛的时期，有些人用多 CNMAE 记录而不是多地址记录来实现循环法:

```
foo1.bar.baz.   60   IN   A   192.168.1.1
foo2.bar.baz.   60   IN   A   192.168.1.2
foo3.bar.baz.   60   IN   A   192.168.1.3
foo.bar.baz.    60   IN   CNAME   foo1.bar.baz.
foo.bar.baz.    60   IN   CNAME   foo2.bar.baz.
foo.bar.baz.    60   IN   CNAME   foo3.bar.baz.
```

对那些已经习惯我们反复强调不要把任何东西和 CNAME 记录混淆的人来讲，这看上去可能很奇怪。但是 BIND 4 名字服务器并不认为这是配置错误，而只是以循环的顺序返回 *foo.bar.baz* 的 CNAME 记录。

另一方面，BIND 8 名字服务器更敏感一些，能够发现这个错误。然而，你可以明确地配置 BIND 8，使它允许一个域名有多个 CNAME 记录：

```
options {
    multiple-cnames yes;
};
```

直到BIND 9.1.0，BIND 9才注意到多CNAME的问题。BIND 9.1.0能检测到这个问题，但并不支持*multiple-cnames*语句。

rrset-order子语句

某些时候我们是不希望让名字服务器使用循环法的。例如，你希望把一个Web服务器作为另一个的备份，这就要求名字服务器只有在返回主Web服务器之后才返回备份服务器的地址。但是，这时就不能使用循环法了，因为它只是在连续的响应中轮转地址的顺序。

BIND 8.2及后续版本的名字服务器（但是不包括BIND 9名字服务器，例如9.1.0）允许你对特定的域名和记录类型关闭循环法。例如，如果我们想确认*www.movie.edu*的地址记录总是按同样的次序返回，就可以使用下面的*rrset-order*子语句：

```
options {
    rrset-order {
        class IN type A name "www.movie.edu" order fixed;
    };
};
```

我们应该减小*www.movie.edu*的地址记录的TTL，这可以使得缓存这些记录的名字服务器不会长时间地轮转着给出它们。

class、*type*和*name*的设置决定指定的顺序会应用到哪些记录上。默认的类为IN，类型为ANY，名字是*，也就是任何记录。所以下面语句是指以随机的次序返回名字服务器的所有记录：

```
options {
    rrset-order {
        order random;
    };
};
```

名字设置中的最左边可以包括一个通配符，如下所示：

```
options {
    rrset-order {
        type A name "*.movie.edu" order cyclic;
    };
};
```

虽然只允许使用一个*rrset-order*子语句，但是一个子语句中可以包含多个次序说明。一个记录集合使用它匹配的第一个次序说明。

rrset-order 支持三种不同的次序：

fixed
: 总是以同样的次序返回匹配的记录

random
: 以随机的次序返回匹配记录

cyclic
: 以循环的次序返回匹配记录

默认的操作是：

```
options {
    rrset-order {
        class IN type ANY name "*" order cyclic;
    };
};
```

不幸的是，配置*rrset-order*并不是很完美的解决方法，因为解析器和名字服务器的缓存会干扰它的工作。更好更长期的解决办法是SRV记录，我们将在第十六章讨论这方面的内容。

名字服务器地址排序

有时候，不管是循环法还是其他配置规则都不是你想要的。当你和一台有多个网络接口，也就是有多个IP地址的主机联系时，根据你的主机地址选择其中某个特定的接口可能会给你更好的性能。这不是 *rrset-order* 子语句能够做到的。

如果多宿主主机是在本地，与你的主机共享一个网络或者子网，那么这个多宿主主机的某个地址会更“近”一些。如果该多宿主主机在远程网络里，使用其中一个网络接口可能会比使用另一个有更好的性能，不过很可能使用哪个地址差别不会太大。很久以前，网络 10（以前 ARPAnet 的主干网）总是比其他远程地址要近一些。由

于现在 Internet 发展迅猛，所以对于远程多宿主主机，使用这个网络而不用那个性能往往不会有显著的改进，不过我们还是要讨论那种情况。

在我们讨论名字服务器进行的地址排序前，你首先应该看看由解析器进行的地址排序是否能更好地满足你的要求。（参见第六章中“ sortlist 指令 ”一节。）因为你的解析器和名字服务器可能处于不同的网络上，让解析器为它自己所处的主机进行优化地址排序会更有意义一些。在名字服务器端进行地址排序会工作得相当好，但是要想使它对于所服务的每个解析器都优化是很困难的。解析器地址排序这个特性是从 BIND 4.9 才开始有的，所以，如果你的解析器（不是你的名字服务器）比 4.9 还要老或者根本就不是 BIND，你就太不走运了。你将必须在名字服务器端进行地址排序，这个特性是从 BIND 4.8.3 开始有的。

最初的几版 BIND 8 中没有名字服务器的地址排序特性，主要是因为开发人员认为它在服务器里没什么作用。但在 BIND 8.2 中又恢复了这一特性 —— 实际上是增强了。BIND 9.1.0 是第一个支持地址排序特性的 BIND 9 版本。

BIND 4 地址排序

尽管 BIND 4 的地址排序在配置上比 BIND 8 简单，但是，因为它的配置有相当多是自动生成的，所以描述起来更加复杂。让我们首先来介绍一下 BIND 4 的地址排序。

本地多宿主主机

让我们首先来看一下本地多宿主主机。假定你有一个源主机（也就是一台保存了主名字服务器源数据的主机）位于两个网络上，不妨称为网络 A 和网络 B，这台主机使用 NFS 把文件系统导出到两个网络上。网络 A 上的主机如果使用源主机网络 A 上的接口的话，性能就会好一些。同样网络 B 上的主机将会因为使用源主机网络 B 上的接口而获益。

在第四章中，我们提到过 BIND 会返回一台多宿主主机所有的地址。但是无法保证 DNS 服务器会以何种顺序返回这些地址，所以我们为每个接口都分配一个别名（*wormhole.movie.edu* 的是 *wh249.movie.edu* 和 *wh253.movie.edu*）。如果使用其中一个接口会更好，你（或者更严格一点来说，一个 DNS 客户端）就可以使用适当的别

名来获得正确的地址。你可以用别名来选择那个“更近”的接口（例如，为了建立NFS安装），但是因为有了地址排序，所以并不总是需要它们。

默认地，BIND 4服务器在此种情况下会进行地址排序：如果给名字服务器发送查询的主机和这个名字服务器处于同一网络中（例如，都在网络A上），BIND就会对响应中的地址进行排序。那么BIND怎么知道它是和查询它的主机位于同一网络中的呢？这是因为BIND在启动时会找出它所运行的主机的所有接口地址。BIND从这些地址中计算出网络号以创建默认排序列表。当收到一个查询时，BIND会检查发送者的地址是否位于默认排序列表中的某个网络上。如果是的，那么查询就是来自于本地的，BIND就会对答案中的地址进行排序。

在图10-3中，假设*notorious*上是BIND 4名字服务器。名字服务器的默认排序列表包括网络A和网络B。当*spellbound*向*notorious*发送查询查找*notorious*的地址时，它获得的回答中*notorious*的网络A的地址会在最前面，这是因为*notorious*和*spellbound*共享网络A。当*charade*查找*notorious*的地址时，它获得的回答中*notorious*的网络B的地址会在最前面，这是因为两台主机都在网络B上。在这两个例子中，名字服务器会在响应中对地址排序，因为发送查询的主机与名字服务器主机共享一个网络。排序后的地址列表是较“近”的接口在最前面。

让我们来做一点小小的改动。假设名字服务器是运行在*gaslight*上的。当*spellbound*向*gaslight*查询*notorious*的地址时，*spellbound*将看到的响应和上面例子中的一样，这是因为*spellbound*和*gaslight*共享网络A，这就意味着名字服务器会对响应排序。不过，*charade*将会看到不一样顺序的响应，这是由于它和*gaslight*不共享一个网络。在对*charade*的响应中，*notorious*较近的地址仍然是在前面，但这只是因为运气，而不是名字服务器对地址进行了排序。在这种情况下，要想*charade*也能从BIND 4的默认地址排序中受益，你就不得不在网络B上再运行一个名字服务器。

正如你所看到的，你能得益于在每个网络上都运行一个名字服务器；不仅当你的路由器关闭时你的名字服务器仍然可用，而且它还能对多宿主主机地址进行排序。由于名字服务器会对地址排序，你就不需要为获得最好的响应而为NFS安装或网络登录指定别名了。

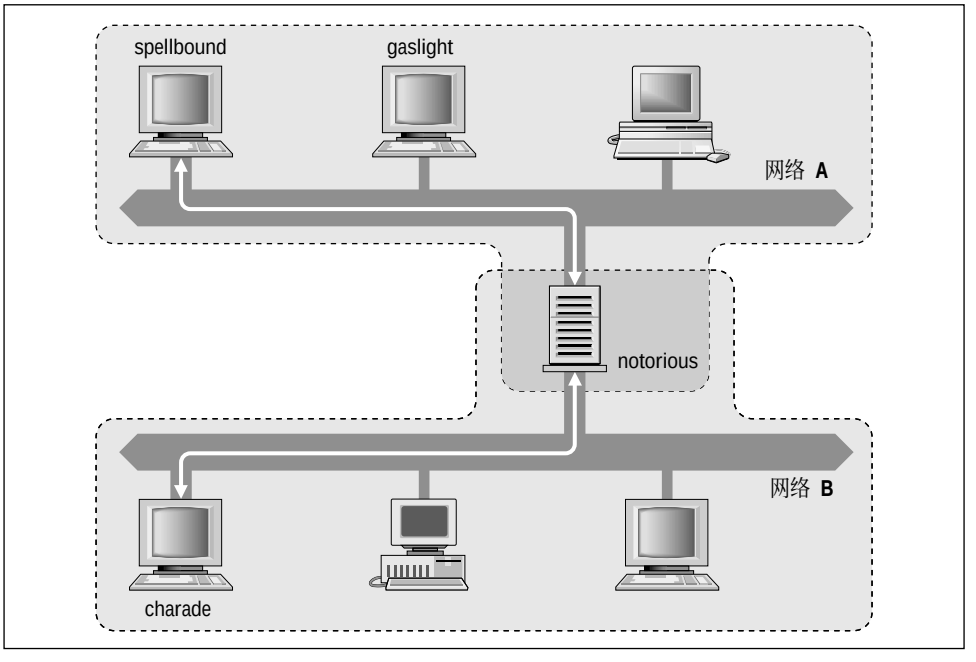


图 10-3　同本地多宿主主机通信

远程多宿主主机

假定你的网络经常和某个特定的远程网络或一个“远方”的本地网络联系，你可以通过使用远程主机在某个网络上的地址而获得更好的性能。例如，*movie.edu*区包括网络 192.249.249/24 和 192.253.253/24。让我们添加一个到网络 10/8（以前的 ARPAnet）的连接。要联系的远程主机有到两个网络的连接，一个到网络 10/8，一个到网络 26/8。这台主机没有到网络 26/8 的路由，但是因为某些特殊原因，它有一个到网络 26/8 的连接。因为到网络 26/8 的路由器总是超负荷运行，你可以通过使用远程主机在网络 10/8 上的地址获得更好的性能。图 10-4 显示了这种情况。

如果*terminator.movie.edu*上的一个用户要联系*reanimator.movie.edu*，最好使用网络 10/8 的地址，因为通过网关 B 到网络 26/8 的地址会比直接路由慢。不幸的是，运行在*terminator.movie.edu*上的名字服务器不会在查找*reanimator.movie.edu*的地址时有意识地把网络10/8的地址放在前面；*terminator.movie.edu*惟一连接的网络就是 192.249.249/24，所以它并不知道网络 10/8 比网络 26/8 更近。现在就是 *sortlist* 指令发挥作用的时候了。为了指明网络 10/8 中的地址是更好的选择，在 *named.boot* 文件中添加下面一行：

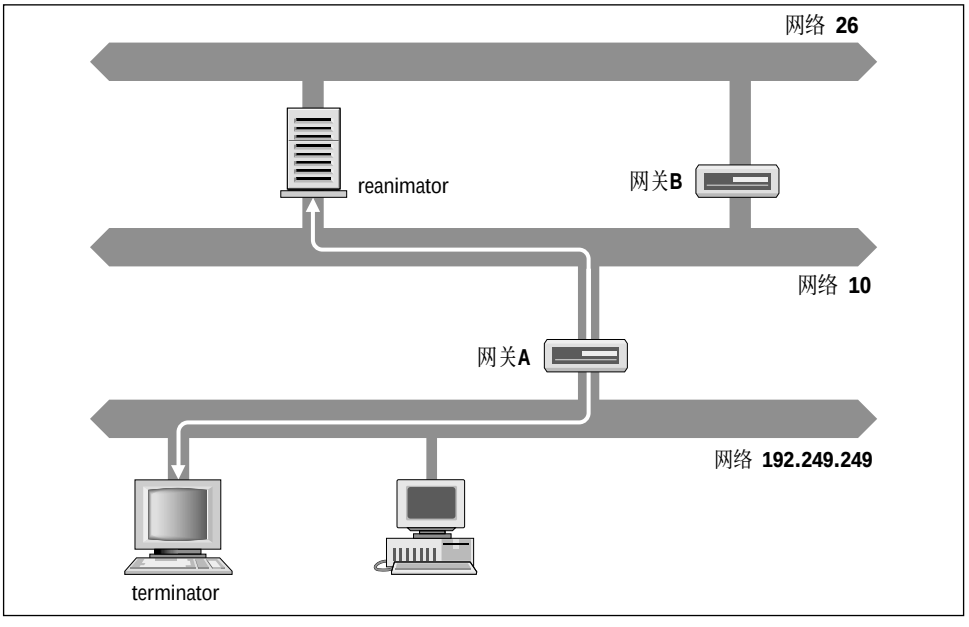


图 10-4　同远程多宿主主机通信

```
sortlist 10.0.0.0
```

*sortlist*的参数会被附加到默认排序列表后面。有了这个指令，*terminator.movie.edu*上的排序列表就包括有网络 192.249.249/24 和 10/8 了。现在，当 *terminator.movie.edu* 上的用户查询 *terminator.movie.edu* 上的名字服务器时，因为这是个本地查询，名字服务器会对得到的响应进行排序，检查位于192.249.249/24网络上的地址并将其放在响应的最前面。如果没有网络 192.249.249/24 上的地址，它就会检查是否有网络10/8上的地址，并把它们放在响应的最前面。这就解决了我们前面提到的问题；现在，当查询 *reanimator.movie.edu* 时，它的网络 10/8 上的地址就会被放在响应的前面了。

已划分子网的网络上的地址排序

划分了子网的网络对地址排序的影响是很轻微的。当名字服务器创建它默认的排序列表时，会把子网号和网络号都加进去。和以前一样，当查询是本地的时，名字服务器就会对响应中的地址进行排序，公有的子网地址会被放在前面。不幸的是，不是所有的事情都很完美：你无法为网络中的其他子网添加*sortlist*指令。原因是：名

字服务器总是假定所有的*sortlist*配置的都是网络号（而不是子网号），而你的网络号早已经在排序列表中了。既然你的网络号已经在列表中，所以子网的*sortlist*也就被丢弃了。

多个 sortlist 条目

最后一件事情，如果你想添加多个*sortlist*条目，就必须把它们都写在一行中，如下所示：

```
sortlist 10.0.0.0 26.0.0.0
```

BIND 8 和 BIND 9 上的地址排序

BIND 8.2 以及后续版本（9.1.0 以及后续版本也是一样）的名字服务器也可以进行地址排序。不幸的是，具体的实现不是自动的，也不是很容易配置。其中的关键是称为 *sortlist* 的 *options* 子语句。

*sortlist*子语句以地址匹配列表作为参数。不过，与作为访问控制列表的地址匹配列表不同，对*sortlist*中的地址匹配列表有特殊的解释。地址匹配列表中每个条目本身就是一个包括一个元素或两个元素的地址匹配列表。

只含有一个元素的条目是用来检查查询者的 IP 地址的。如果查询者的 IP 地址和该元素匹配，那么名字服务器就在对它的响应中对地址排序，使和该元素匹配的地址总排在最前面。有点迷糊？下面我们给出一个例子：

```
options {
    sortlist {
        { 192.249.249/24; };
    };
};
```

这个排序列表中惟一的条目只有一个元素。在对来自网络192.249.249/24的查询的响应中，它会把在该网络上的地址放在前面。

如果一个条目含有两个元素，那么第一个元素用来匹配查询者的IP地址。如果查询者的地址和第一个元素匹配，那么名字服务器会在对它的响应中进行地址排序，把和第二个元素匹配的地址放在最前面。条目中的第二个元素其实也可以是一个完整

的包括几个元素的地址匹配列表，在这种情况下响应中的第一个地址会是第一个匹配该列表的地址。这里是个简单的例子：

```
options {
    sortlist {
        { 192.249.249/24; { 192.249.249/24; 192.253.253/24; }; };
    };
};
```

这个排序列表适用于192.249.249/24上的查询者，会把它们自身网络上的地址放在最前面，然后是192.253.253/24上的地址。

排序列表规范中的元素可以是子网，甚至是单个主机：

```
options {
    sortlist {
        { 15.1.200/21;          // 如果请求者在15.1.200/21上,
            { 15.1.200/21;      //那么最好是那个子网上的地址,
            15/8; };            // 或至少是15/8上的地址
       };
    };
};
```

更喜欢使用特定网络上的名字服务器

BIND 8的拓扑特性有点和*sortlist*类似，但是它只适用于选择名字服务器的过程。（BIND 9不支持9.1.0中的拓扑。）我们在本书前面描述过BIND是如何在同一区的多个权威名字服务器中进行选择的，它会选择往返时间（roundtrip time，RTT）最短的名字服务器。但是我们还是撒了一点小谎。BIND 8在比较RTT时，实际上会把远程名字服务器按照64毫秒为一段分成许多段。第一段实际上只有32毫秒宽（这里，我们再重复一次！），从0到32毫秒。下一组是从33毫秒到96毫秒，依此类推。这样设计这些段是为了让位于不同大陆上的名字服务器总是位于不同的段里。

这种做法是为了迎合带宽较低的名字服务器，但是对位于同一段中的名字服务器则一视同仁。如果名字服务器比较两个远程名字服务器的RTT时发现其中一个位于较低的带宽中，名字服务器就会选择向较低带宽中的那个发送查询。但是如果远程的名字服务器位于同一带宽中，名字服务器就会检查哪一个在拓扑上更近一些。

拓扑给你一个在选择要查询的名字服务器时可以考虑的因素。它使你可以优先选择

位于特定网络上的名字服务器。拓扑以地址匹配列表作为参数，其中的条目是网络号，按照本地名字服务器愿意使用它们的程度列出（从高到低）。因此：

```
topology {
    15/8;
    172.88/16;
};
```

会告诉本地名字服务器先去使用网络15/8上的名字服务器而不是其他名字服务器，或者在不考虑网络15/8上的名字服务器时先使用网络172.88/16上的而不是其他的。所以，如果名字服务器有几个选择，一个位于网络15/8上，一个位于172.88/16上，还有一个位于192.168.1/24上，假定它们的RTT值都在同一段中，它就会选择去查询位于15/8上的名字服务器。

你也可以否定拓扑地址匹配列表中的某些条目以避免使用特定网络上的名字服务器。地址匹配列表中被否定的条目位置越靠前，越要避免使用。例如，你可以使用这个功能避免名字服务器向网络上比较脆弱的远程名字服务器提出查询。

非递归名字服务器

默认情况下，BIND解析器发送递归查询，而名字服务器也会做必要的工作来回答递归查询。（如果你不记得递归查询是如何工作的，请参阅第二章）在查找递归查询的答案时，名字服务器的缓存中存放了一些关于其他区的非权威信息。

在有些情况下，我们不希望名字服务器做额外工作来回答递归查询或建立一个数据缓存。根名字服务器就是这样的。根名字服务器非常繁忙，不能花费额外的精力来为递归查询查找答案。所以它们只是根据它们的权威数据来做出响应。这个响应中可能会有答案，但更有可能的是这个响应是指向其他名字服务器的。而且由于根服务器不支持递归查询，它们就不用建立起非权威数据的缓存，这样做很有好处，否则它们的缓存会非常大（注7）。

注7： 注意，根名字服务器通常不会接收递归查询，除非某个名字服务器的管理者把根名字服务器配置成该名字服务器的转发器，或者某个主机的管理者把它的解析器配置成使用根名字服务器作为主机的名字服务器，或者某个用户把*nslookup*指向根名字服务器。然而，这些情况并不常见。

可以用下面的语句使 BIND 名字服务器以非递归模式运行：

```
options {
    recursion no;
};
```

在 BIND 4.9 服务器上，要使用指令：

```
options no-recursion
```

现在服务器会把递归查询当做非递归查询来响应。

如果你想不让服务器建立起缓存，那还需要将一个配置选项和 *recursion no* 一起使用：

```
options {
    fetch-glue no;
};
```

在 BIND 4.9 上是这样的：

```
options no-fetch-glue
```

这就使得服务器在生成一个响应的附加数据段时不会去获取丢失的glue记录。由于 BIND 9 名字服务器并不会去获取 glue 记录，所以在 BIND 9 中 *fetch-glue* 子语句已经过时了。

如果你决定将服务器变成非递归的，那么在任何主机的 *resolv.conf* 文件中都不要列出这个名字服务器。你可以使名字服务器成为非递归的，但是你没有办法让你的解析器来使用一个非递归的名字服务器（注8）。如果你的名字服务器仍然需要为一个或多个解析器服务的话，你可以使用 *allow-recursion* 子语句，BIND 8.2.1 以及后续版本（包括 BIND 9）提供了这项功能。*allow-recursion* 子语句以一个地址匹配列表作为参数；只有和地址匹配列表匹配的查询者才能发送递归查询，而对待其他查询者就好像关闭了递归功能一样：

注 8： 通常是这样的。当然，被设计成发送非递归查询的程序或者可被配置成发送非递归查询的程序（如，*nslookup*）仍然能工作。

```
options {
    allow-recursion { 192.253.254/24; }; // 只有FX子网上的解析器能够发送递归
                                         // 查询
};
```

allow-recursion 默认的设置是对所有的IP地址提供递归查询服务。

当然，也不要将非递归名字服务器设为转发器。当一个名字服务器将另一个服务器作为转发器时，它将递归查询转发给转发器。不过，可以通过设置*allow-recursion*，只允许被授权了的名字服务器使用你的转发器。

可以将一个非递归名字服务器作为你的区数据的一个权威（也就是你能告诉父名字服务器将关于你的区的查询指向这个服务器）。能够这样是因为名字服务器之间发送的是非递归查询。

避免使用伪装的名字服务器

在你作为名字服务器管理员的工作期间，可能会发现一些远程的名字服务器发回一些错误的信息作为响应，这些信息可能是过时的、错误的、格式不对的，或者根本就是蓄意欺骗的。你可以尝试联络其管理员以解决这个问题。或者你可以自己来配置你的名字服务器不要使用这些服务器，BIND 4.9、BIND 8、BIND 9.1.0及其后续版本可以做到这一点。下面是配置文件中的语句：

```
server 10.0.0.2 {
    bogus yes;
};
```

或者在BIND 4.9服务器上是：

```
bogusns 10.0.0.2
```

当然，你需要填上正确的IP地址。

如果你告诉你的名字服务器不要再联系某个区的惟一的名字服务器，那就不要再指望可以查询该区中的名字。还好，还有其他的名字服务器能够提供有关该区的正确信息。

把一个远程的名字服务器排除在外的更有效的方法是把它放在你的*blackhole*列表

中。你的名字服务器不会查询该列表中的名字服务器，而且也不会响应它们的查询（注 9）。*blackhole* 是一个以地址匹配列表为参数的 *options* 子语句：

```
options {

    /* 不要浪费你的时间试图响应来自 RFC 1918 私有地址的查询 */

    blackhole {
        10/8;
        172.16/12;
        192.168/16;
    };
};
```

这避免了名字服务器试图对有关 RFC 1918 中的私有地址查询做出响应。在 Intenet 上没有到这些地址的路由，所以试图对它们做出响应只是浪费 CPU 时间和带宽。

8.2 以后的 BIND 8 和 9.1.0 以后的 BIND 9 支持 *blackhole* 子语句。

系统优化

对大多数名字服务器来说，默认配置值就能很好地工作，而你的服务器可能还需要进一步地优化。

区传送

区传送可以给名字服务器带来很大的负荷。在 BIND 4 名字服务器上，对外的区传送（传送某个以它为主名字服务器的区），要用 *fork()* 来创建新的 *named* 进程，因此会使用大量额外的内存。BIND 4.9 引入了新的机制来限制你的名字服务器给它的主名字服务器所带来的区传送负荷。BIND 8 和 9 除此之外还有其他机制。

限制每个名字服务器请求的传送数量

使用 BIND 4.9 及其后续版本，你可以限制名字服务器从远程名字服务器那里所能

注 9： 我们的意思是，真的不会做任何响应。由 *allow-query* 访问控制列表禁止的查询者会收到一个响应，告知查询被拒绝，但是在 *blackhole* 列表中的查询者将不会得到任何响应。

请求的区的数量。这会使该远程名字服务器的管理员非常高兴，因为这样即使所有的区数据都发生了改变，它的主机也不会因为区传送而崩溃了，如果该主机涉及到上百个区，这就会非常重要了。

在 BIND 8 和 BIND 9 中，配置文件中的语句如下：

```
options {
    transfers-per-ns 2;
};
```

与此等价的 BIND 4 的启动文件中的指令是：

```
limit transfers-per-ns 2
```

在BIND 9中，你也能够设置针对每个不同服务器的限制，而不是针对全部服务器。为了实现这个目的，你要在 *server* 语句中使用 *transfers* 子语句，这里的服务器指的就是你希望加以限制的服务器：

```
server 192.168.1.2 {
    transfers 2;
};
```

这将覆盖*options*语句中所有的全局限制。默认的限制是每个名字服务器同时可以进行两个区数据传送。这个限制看起来很小，但是确实有效。让我们看一下它具体是怎么实现的：假定你的名字服务器需要从一个远程的名字服务器那里加载四个区的数据，你的名字服务器会首先开始传送前两个区的数据，同时等待进行后两个区数据的传送。当前两个区传送中的一个完成后，名字服务器就开始传送第三个区。当另一个区数据传送完成后，名字服务器就开始传送第四个区。最终结果和没有这些限制时完全一样，所有区的数据传送都完成了，但是工作是分开进行的。

什么时候你可能会考虑增加这个限制呢？你可能会注意到，和远程名字服务器同步数据需要花费太多时间，而且你知道原因在于要这样顺序地进行传输，而不是因为主机之间网络速度太慢，那么你可能想增加这个限制了。这可能只有在你维护着上百或上千个区时才会有问题。而且你要确保远程名字服务器和两者之间的网络能够应付同时进行更多的区传送所造成的新增的工作负荷。

限制区请求传送的总数

刚才讲的是限制对单个远程名字服务器区传送请求数量。现在要讲的则是限制多个远程名字服务器。BIND 4.9及其后续版本允许你限制你的服务器能同时请求的区传送的总数。默认的限制是10。就像我们前面解释过的那样，默认情况下，你的服务器同时只能从指定的任何单个远程名字服务器那里加载两个区。如果你的服务器从五台服务器中的每一台那里加载两个区，那么你的服务器就达到这个限制了，于是它就会在当前任何一个正在进行的区传送结束前推迟其他任何的传送。

BIND 8 和 9 的 *named.conf* 文件中的语句为：

```
options {
    transfers-in 10;
};
```

等价的 BIND 4 的启动文件中的指令是：

```
limit transfers-in 10
```

如果你的网络或主机无法同时处理10个区传送，你就应该考虑减少这个数值。如果你管理着一个支持上百或上千个区的名字服务器，而且你的主机和网络能够支持这些负荷，那么你可能就会想增加这个数值。如果你增加了这个限制，可能也需要增加每个名字服务器能够同时进行的传送的数目。（例如，如果你的名字服务器只从4个远程名字服务器那里加载区，而从每个远程服务器那里同时只能加载2个区，那么你的服务器最多也就同时会进行8个区传送。除非也增加从每个服务器那里同时进行区传送的数目，否则只增加同时允许进行的区传送的总数不会有任何影响。）

限制区传送的总数

BIND 9 也允许你限制本服务器能同时服务的区传送的总数。可以证明这比限制区传送请求的数量更有效，因为如果没有这个限制，主名字服务器的负载大小就完全取决于维护辅名字服务器的管理员。BIND 9 中该语句是：

```
options {
    transfers-out 10;
};
```

默认值是 10。

限制区传送的持续时间

BIND 8和BIND 9还会允许你限制一个向内的区传送的持续时间。默认情况下，区传送被限制在120分钟，即2个小时以内。这样定是考虑到，如果一个区传送花费的时间比2个小时还要长就很有可能是停在那里无法完成了，而该进程只是在毫无必要地占用资源。如果你知道你的服务器是某个区的辅名字服务器，一般来说需要多于120分钟才能完成传送，所以你想用一个更短或更长的时间限制，可以使用这样的语句：

```
options {
    max-transfer-time-in 180;
};
```

你甚至可以在*zone*语句中使用*max-transfer-time-in*子语句来限制传输某个特定区允许花费的时间。例如，你知道因为*rinkydink.com*区的大小或是因为到其主名字服务器的线路速度很慢，这个区总是需要很长时间进行传输（比如说，3个小时），但是你仍然希望对于其他区的传输使用更短一些的限制（可能是1小时），你可以使用：

```
options {
    max-transfer-time-in 60;
};

zone "rinkydink.com" {
    type slave;
    file "bak.rinkydink.com";
    masters { 192.168.1.2; };
    max-transfer-time-in 180;
};
```

在BIND 9中也有一个*max-transfer-time-out*子语句，使用方法和上面说的相同（可以在*options*语句中，也可以在*zone*语句中）。它控制向外的区传送（也就是到辅名字服务器的传送）的时间，并且有和*max-transfer-time-in*一样的默认值（120分钟）。

BIND 9名字服务器甚至允许你限制区传送的空闲时间，即区传送没有任何进展持续的时间。有两个配置子语句：*max-transfer-idle-in*和*max-transfer-idle-out*，分别控制向内和向外的区传送空闲时间。同传送时间限制一样，二者既能够作为*options*的子语句也能作为*zone*的子语句。默认空闲时间为60分钟。

限制区传送的频度

如果把区传送的刷新间隔设置得太短，可能会导致辅名字服务器不能正常工作。比如，你的名字服务器是上千个区的辅名字服务器，而其中某些区的管理员将相应区的刷新时间间隔设置得非常短，那么你的名字服务器也许就不能及时完成其所需要的所有刷新。（如果你正在维护一个名字服务器，它是许多区的辅名字服务器，请仔细阅读后面一节“限制SOA查询的数量”，你可能还需要调整所允许的SOA查询的数量。）另一方面，没经验的管理员也可能会将区的刷新间隔设置得太高，导致主和辅名字服务器之间长时间地数据不一致。

BIND 9.1.0及其后续版本允许通过使用*max-refresh-time*和*min-transfer-time*来限制刷新间隔。如果这些子语句用在*options*语句中，它们就指定了所有master、slave和stub区的刷新间隔；而如果它们用在*zone*语句中，就仅仅指定了某个特定区的刷新间隔。二者都以秒数作为参数：

```
options {
    max-refresh-time 86400;     // 刷新间隔最大不应该超过1天
    min-refresh-time 1800;      // 最小不应小于30分钟
};
```

BIND 9.1.0以及后续版本的名字服务器允许用*max-retry-time*和*min-retry-time*子语句限制重试间隔，语法类似。

更有效的区传送

我们前面提到的区传送是通过一条TCP连接端到端地发送，由多个DNS消息组成。传统上的区传送只会在每个DNS消息中放入一个资源记录。这很浪费空间：每个DNS消息都需要一个完整的首部信息，即使你只传送一个记录。这就有点像在一辆大旅行车中只有你一个人。一个基于TCP的DNS消息其实可以放下更多的记录：最大值是64KB!

BIND 8和BIND 9服务器理解一种新的区传送格式，被称为*many-answers*（多答案）。这个*many-answers*格式在一个DNS消息中尽可能多地放入记录，从而使区传送使用的带宽更少，因为这样的开销更少，而所用的CPU时间也更少，因为花费在排列DNS消息上的时间更少。

transfer-format 子语句能够控制以该名字服务器为主名字服务器的区的传送格式。也就是说，它决定着你的名字服务器向它的辅名字服务器传输区数据时所使用的格式。这个 *transfer-format* 既是一个 *options* 的子语句，也是 *server* 的子语句：作为 *options* 子语句，*transfer-format* 控制着这个服务器的所有区传送的格式。BIND 8 默认地使用旧的 *one-answer* 区传送格式，以便和 BIND 4 名字服务器兼容。BIND 9 默认地使用 *many-answers* 格式。下面这条语句：

```
options {
    transfer-format many-answers;
};
```

把这个服务器的设置改为对所有的辅名字服务器的区传送使用*many-answers*格式，除非在 *server* 语句中特别指明不要使用这种格式，如下所示：

```
server 192.168.1.2 {
    transfer-format one-answer;
};
```

使用*many-answers*格式的一个缺点是，因为要使用新格式，区传送实际上可能会花费更长的时间，尽管从带宽和 CPU 使用率的角度来说是效率更高了。

如果你喜欢使用新的效率更高的区传送，可以采用下面的方法：

- 如果你的大部分辅名字服务器运行的是能够理解 *many-answers* 格式的 BIND 8、BIND 9 或微软的 DNS Server，就将你的名字服务器的全局区传送格式设置为 *many-answers*（如果使用的是 BIND 9，你根本就不需要修改）。
- 如果你的大部分辅名字服务器运行的是 BIND 4，那就将你的名字服务器的全局区传送格式设置为 *one-answer*。然后对某些特殊的服务器再使用 *transfer-format* 的 *server* 子语句来调整全局设置。

记住，如果你使用的是 BIND 9，你需要针对所有的 BIND 4 辅名字服务器添加一条 *server* 语句，把它们的传输格式改为 *one-answer*。

资源限制

有时你想告诉服务器不要太贪婪：不要使用多于某个值的内存，不要打开多于某个值的文件。BIND 4.9 引入了这些限制，而且 BIND 8 和 9 还给出了一些新东西。

改变数据段大小的限制

有些操作系统对一个进程能使用的内存数有默认的限制。如果你的操作系统不允许名字服务器使用更多的内存，那么服务器可能会崩溃或退出。除非你的名字服务器处理的数据量非常非常大或者限制非常小，否则你是不会超过这个限制的。但是，如果你确实会超过这个限制的话，BIND 4.9、BIND 8和BIND 9.1.0及其后续版本有相应的配置选项可以改变系统对于数据段大小默认的限制。可以使用这些选项为*named*进程设置一个比系统默认限制更高一点的限制。

对于BIND 8和9来说，这个语句为：

```
options {
    datasize size
};
```

对于BIND 4来说，相应的指令是：

```
limit datasize size
```

*size*是一个整数值，默认的单位是字节。你还可以指定其他的单位，只要在这个整数后面追加一个相应的字符，例如：k（千字节）、m（兆字节）或者g（千兆字节）。譬如64m是指64兆字节。

注意：并不是所有的系统都支持增大某个进程的数据段大小。如果你的系统不支持，名字服务器就会发送一条LOG_WARNING级别的系统日志消息，告诉你这个特性无法实现。

改变栈大小的限制

除了允许修改名字服务器数据段的大小限制外，BIND 8、BIND 9.1.0及其后续版本的名字服务器还允许你调整系统规定的*named*进程的栈可使用的内存大小。语法如下：

```
options {
    stacksize size;
};
```

这里*size*的格式和上面*datasize*的一样。同*datasize*一样，这个特性只能在那些允许一个进程修改栈大小限制的系统上使用。

改变核心大小的限制

如果你不希望*named*在你的文件系统中留下一个巨大的核心文件，那么至少你可以使用 *coresize* 来限制它们的大小。反过来，如果 *named* 因为操作系统严格的限制而无法在内存中转储完整的核心文件，可以用 *coresize* 来增加这个限制。

coresize 的语法是：

```
options {
    coresize size;
};
```

与上面的 *datasize* 一样，这个特性只能在那些允许进程修改核心文件大小限制的操作系统上使用，而且也不能在 9.1.0 之前的 BIND 9 版本上使用。

改变打开文件总数的限制

如果你的名字服务器是大量区的权威，*named*进程就会在启动时打开许多文件 ——每个权威区一个，假定你对于那些自己是其辅名字服务器的区来说使用备份区数据文件的话。同样，如果运行你的名字服务器的主机有许多虚拟网络接口（注 10），*named*会要求对每个接口都使用一个文件描述符。大部分Unix操作系统对任何进程可以同时打开的文件数目都有限制。如果你的名字服务器试图打开多于系统限制所允许的文件个数，你就会在系统日志输出中看到这样的消息：

```
named[pid]: socket(SOCK_RAW): Too many open files
```

如果你的操作系统也允许修改单个进程能够打开的文件个数的限制，你可以使用 BIND 的 *files* 子语句来增加这个限制：

```
options {
    files number;
};
```

默认的值是 *unlimited*（这是一个有效值），虽然这意味着名字服务器对于同时打开的文件总数没有任何限制，但是操作系统却可能会有。虽然我们知道你会讨厌我们这样说，但是我们还是不得不说，那就是，9.1.0 以前的 BIND 9 并不支持该功能。

注 10： 对于“ 打开过多文件 ”这个问题，第十四章有一个比增加打开文件总数限制更好的解决方案。

限制客户端的数量

在BIND 9中你可以限制名字服务器能够同时服务的客户端的数量。利用*recursive-clients*子语句，你可以限制递归客户（解析器以及将你的名字服务器作为转发器的名字服务器）的数目：

```
options {
    recursive-clients 10;
};
```

默认的限制是1000。如果你发现你的名字服务器拒绝递归查询，并且在日志中记录像下面这样的出错消息：

```
Sep 22 02:26:11 terminator named[13979]: client 192.249.249.151#1677: no more
recursive clients: quota reached
```

你可能会打算增加上限。相反，如果你发现你的名字服务器正挣扎着处理其接收到的已泛滥的递归查询，你应当降低上限。

使用*tcp-clients*子语句，你也能够限制你的名字服务器能处理的并发TCP连接数（区传送和基于TCP的查询）。TCP连接消耗了比UDP更多的资源，这是因为主机需要追踪TCP连接的状态。默认限制是100。

限制SOA查询的数量

BIND 8.2.2及其后续版本的名字服务器允许你限制你的名字服务器所允许的对外（outstanding）SOA查询数量。如果你的名字服务器是上千个区的辅名字服务器，它可能在任意时刻都要面对许多请求那些区的SOA记录的查询。追踪每个查询需要一定数量的内存，因此默认情况下，BIND 8名字服务器限制对外SOA查询数量为4。如果你发现你的名字服务器不能完成其作为辅名字服务器的职责，你也许需要用*serial-queries*子语句来增加SOA查询数量的限制：

```
options {
    serial-queries 1000;
};
```

BIND 9不使用*serial-queries*子语句。BIND 9限制串行查询的速率（最高为每秒20个），而非对外查询的数量。

维护间隔

BIND名字服务器会进行周期性的维护工作 :例如刷新其作为辅名字服务器的区。使用 BIND 8 和 9，你就可以控制这些事情发生的频率，或者它们是否需要发生。

清理缓存的间隔

那些比 BIND 4.9 还要古老的名字服务器只能被动地从缓存中清除掉那些过时的数据。这样的名字服务器在向查询者发送答案前会先检查该记录的TTL是否已经期满了。如果是的话，名字服务器就会启动解析进程以获得最新的数据。这就意味着一个 BIND 4 的服务器可能会在名字解析中获得一堆记录，保存在缓存中，然后就只是让它们呆在那里，占用宝贵的内存，即使这些记录都已经过时了。

BIND 8 和 9 现在会在每个清理缓存间隔到了的时候，主动开始检查缓存并剔除掉那些过时的数据。这意味着在同样情况下，BIND 8 和 9 名字服务器会比 BIND 4 名字服务器占用较小的内存用于缓存。从另一方面来说，清理缓存的过程会占用CPU时间，所以对于非常慢的或者非常繁忙的服务器来说，你可能并不希望每个小时都运行一次。

默认的清理间隔是 60 分钟。你可以在 *options* 语句中加上 *cleaning-interval* 子语句来调整这个间隔。例如：

```
options {
    cleaning-interval 120;
};
```

就会把清理间隔设置为 120 分钟。要完全关掉缓存清理就把清理间隔设为 0。

检查接口的间隔

我们已经说过，BIND 会默认地监听主机上所有的网络接口。BIND 8 和 9 实际上很聪明，能注意到运行它的主机上的网络接口何时启用或停用。为了这个目的，它会周期性地检查该主机的网络接口。这会在每个检查接口间隔执行一次，默认的是60分钟。如果你知道运行你的名字服务器的主机上没有动态接口的话，就可以通过把检查接口的间隔设置为0 来禁止检查新接口，这样就可以避免每个小时不必要的开销了：

```
options {
    interface-interval 0;
};
```

从另一方面来说，如果你的主机比每个小时还要频繁地启动或停用网络接口的话，你就需要减少这个间隔了。

进行统计的间隔

好，调整进行统计的间隔，也就是 BIND 8 名字服务器把统计数据转储到统计文件中的频率，不会对性能有太多影响。但是，在这里把它和其他维护间隔一起讲比在本书任何其他地方讲都合适。

statistics-interval 子语句的语法和其他维护间隔的语法极其类似：

```
options {
    statistics-interval 60;
};
```

和其他维护时间间隔一样，默认的是 60 分钟，将时间间隔设为 0 就会使统计数据的周期性转储无效。因为 BIND 9 不向 *syslog* 写统计数据，所以它也就没有可配置的进行统计的间隔。

TTL

在内部，BIND 每隔一段时间就合理地减少缓存记录的 TTL 值。BIND 8 和 9 名字服务器使得这个时间段界限值可配置。

在 BIND 8.2 或其后版本的名字服务器中，利用 *max-ncache-ttl options* 子语句，能够限制针对缓存的否定信息的 TTL。对升级到 8.2 以及其新否定缓存机制（在第四章描述过的 RFC2308 以及所有的东西）的人来说，这样的设计是一个安全网。该新名字服务器根据区的 SOA 记录的最后字段来缓存否定信息，并且许多区管理员仍旧将那个字段作为该区的默认 TTL —— 对否定信息来说，这个值可能太长了。因此一个谨慎的名字服务器管理员能够用下面这样的子语句：

```
options {
    max-ncache-ttl 3600;  // 3600秒是1个小时
};
```

将较大的否定缓存 TTL 修整为 1 小时。默认的是 10800 秒（3 小时）。没有这种预防措施的话，查询一个新记录的人可能得到一个否定回答（也许是因为新记录还未到达区的辅名字服务器），并且名字服务器将在长时间里缓存该回答，使得该记录无法解析，这可不太妙啊。

使用 *max-cache-ttl* 子语句，BIND 9 名字服务器也允许你配置针对缓存记录的 TTL 上限。默认的是 1 周。BIND 8 名字服务器也将 TTL 改为了 1 周，但不允许你配置上限。

最后，还有一个叫做 *lame TTL* 的东西，其实它根本就不是 TTL。它是你的名字服务器记住某个给定远程名字服务器不是某个区的权威的时间，实际上该区已授权给那个远程名字服务器。这就避免了你的名字服务器浪费宝贵的时间和资源向一个名字服务器查询其实它一无所知的域名的信息。8.2 之后的 BIND 8 名字服务器和 9.1.0 之后的 BIND 9 名字服务器允许你用 *lame-ttl options* 子语句调节 lame TTL。默认的 lame TTL 是 600 秒（10 分钟），其最大为 30 分钟。尽管那将严重影响我们，但还是可以用一个为零的值来关闭对 lame 名字服务器的缓存。

兼容性

现在，我们将讨论一些关于名字服务器和解析器之间、名字服务器与名字服务器之间兼容性的配置子语句。

rfc2308-type1 子语句控制你的名字服务器发送否定回答的格式。默认情况下，BIND 8 和 9 名字服务器在否定回答中只包括来自区的 SOA 记录。该回答的另一个合法格式也包括区的 NS 记录，但是一些较老的名字服务器将这种响应错误地解释为指向另一个服务器。如果出于一些古怪的原因（一些我们也想像不出的原因）你也打算发送那些 NS 记录，就使用：

```
options {
    rfc2308-type1 yes;
};
```

rfc2308-type1 首先在 BIND 8.2 中被支持；而 BIND 9 并不支持它。

当你给较老的名字服务器发送缓存的否定响应时，可能也会引起问题。在有否定缓

存之前，所有的否定响应都是权威的。但是有的名字服务器的实现者对他们的名字服务器添加了一个检查：他们只接收权威的否定响应。但是有了否定缓存以后，否定响应可以是非授权的。

*auth-nxdomain options*子语句允许你的名字服务器谎称一个来自其缓存的否定应答实际上是权威的，只要这样那些老的名字服务器就会相信它。默认情况下，BIND 8 名字服务器是启用 *auth-nxdomain* 的（设为 yes），而 BIND 9 名字服务器则默认为关闭它。

最后，当一些喜欢冒险的人将 BIND 8.2.2 移植到 Windows NT 时，他们发现需要名字服务器在行结尾处像对待换行（Unix 的行结尾次序）那样对待回车与换行（Windows 的行结尾次序）。针对这样的情况，可以使用：

```
options {
    treat-cr-as-space yes;
};
```

BIND 9 忽略该选项，因为它总是以相同方式对待回车与换行和单独的换行。

IPv6 寻址规则入门

在介绍下面两个包括域名与 IPv6 地址相互映射的主题前，最好先描述一下 IPv6 地址的表示和结构。也许你知道，IPv6 地址是 128 位长的。IPv6 地址最佳的表示方式是冒号分隔的 8 组数字，每组数字用 4 个十六进制数字表示，例如：

```
0123:4567:89ab:cdef:0123:4567:89ab:cdef
```

第一组十六进制数字（本例中为0123）代表地址的最有效的（或最高次序的）16 位。

以一个或多个零开头的数字组不需要填满 4 个位置，所以你也可以如下写出上述地址：

```
123:4567:89ab:cdef:123:4567:89ab:cdef
```

每组至少必须包含一个数字，除非你使用::符号。::符号允许你压缩由零组成的序列组。当你指定仅仅一个 IPv6 前缀时，这是很方便的。例如：

```
dead:beef::
```

指定 IPv6 地址开始的 32 位为 *dead:beef*，而剩下的 96 位为零。

你也能在IPv6地址的开头使用::来指定一个后缀。例如，IPv6的回送地址通常写成：

```
::1
```

或在 127 个零后紧跟一个 1。你甚至可以在 IPv6 地址中间使用::作为连续零组的缩写：

```
dead:beef::1
```

在一个地址中，::简写只能使用一次，因为多次使用将造成歧义。

地址格式中的IPv6前缀与IPv4的CIDR表示法类似。尽可能多的有效前缀位用标准的IPv6表示法表示，随后是一个短斜杠和一个表示有效位数的十进制数。故此下面三个前缀表示是等效的（尽管明显地不同）：

```
dead:beef:0000:00f1:0000:0000:0000:0000/64
dead:beef::00f1:0:0:0:0/64
dead:beef:0:f1::/64
```

IP 版本 4 的地址是层次化的，反映了 IPv4 网络的本质：个体网络连接到 Internet 服务提供商，然后这些Internet服务提供商又连接到其他Internet服务提供商或直接连到Internet核心。每个服务提供商分配全部32位IP地址中的一些位：更接近Internet核心的服务提供商分配的位越接近地址的高位，最后网络管理员分配剩下的位。

IPv6 被设计用来适应更大的 Internet，因此 IPv6 地址存在更多的层次级别。其中，每一级别对应基于 IPv6 的互联网络中的一个网络级别。

在 IPv6 互联网络的核心，存在 TLA（Top-Level Aggregator，顶级聚集器）。TLA 是直接与互联网络骨干连接并且提供到NLA（Next-Level Aggregator，下级聚集器）连接的网络。NLA 网络将各个地区网络连接到 IPv6 互联网络。图 10-5 描述了整个排列。

与 IPv4 网络一样，每个组织分配一个 IPv6 地址中的某几位。为帮助你描绘出 IPv6 的地址层次，下面是在 RFC 2374 中描述的 IPv6 地址最普遍的结构图。

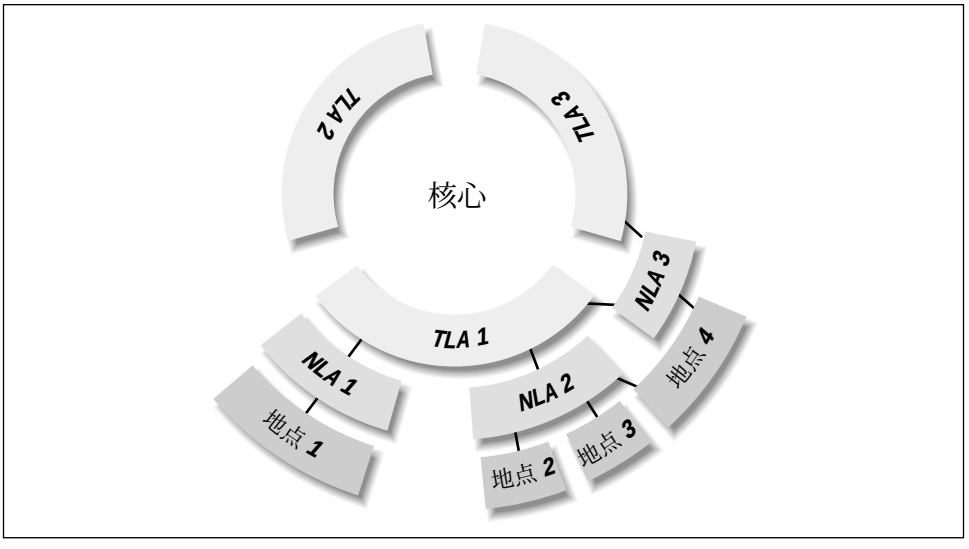


图 10-5　IPv6 互联网络的结构

```
| 3|  13 | 8 |   24   |   16   |          64位               |
+--+-----+---+--------+--------+-----------------------------+
|FP| TLA |RES|  NLA   |  SLA   |          接口ID             |
|  | ID  |   |  ID    |  ID    |                             |
+--+-----+---+--------+--------+-----------------------------+
```

FP 是格式前缀（Format Prefix），即地址的前 8 位，确定地址其余部分的格式。特定格式的格式前缀是001, 被称为IPv6可聚集全局单播地址格式(IPv6 Aggregatable Global Unicast Address Format)。其后的13位指定了TLA，随后是3个保留位（设为零），然后是用来指定NLA的24位。余下的位由地区网络使用：SLA（Site-Level Aggregator，地区级别聚集器）ID，它基本上类似于IPv4地址中的子网位；以及接口（Interface）ID，它惟一指定了在地区网络上的一个特定接口。

在这种地址格式里，每个 ID 由层次中的上一级实体分配给下一级。例如，一个单一、顶级注册机构为TLA分配了TLA ID。依次，TLA为其NLA客户分配NLA ID，同样 NLA 又为它们的客户分配 SLA ID。NLA ID 仅需要在一个 TLA ID 中是惟一的，而 SLA ID 也仅需要在一个特定的 NLA ID 中是惟一的。

地址和端口

BIND 9 名字服务器能够使用 IPv4 和 IPv6 作为传输；也就是说，它们能够在 IPv4 和IPv6 上发送、接收查询和响应。BIND 8 名字服务器只支持IPv4 作为传输。然而，两种服务器都支持相似的子语句来配置它们所监听及发送查询的网络接口和端口。

配置 IPv4 作为传输

使用 *listen-on* 子语句，能够指定你的 BIND 8 或 BIND 9 名字服务器用来监听查询的网络接口。在最简单的形式中，*listen-on* 采用地址匹配表作为参数：

```
options {
    listen-on { 192.249.249/24; };
};
```

名字服务器可在本地主机的任意网络接口上监听，条件是这些网络接口的地址必须与地址匹配表相匹配。另外，还可使用 *port* 修饰字指定特定端口（如，不同于53 的端口）：

```
options {
    listen-on port 5353 { 192.249.249/24; };
};
```

在 BIND 9 中，甚至可以为每个网络接口指定非标准端口：

```
options {
    listen-on { 192.249.249.1 port 5353; 192.253.253.1 port 1053; };
};
```

注意，没有一种方法可配置大多数解析器以查询一个名字服务器的非标准（即，非53）端口，因此这种名字服务器也许没有你所认为的那样有用。尽管如此，它仍可以为区传送服务，因为你能够在 *masters* 子语句中指定非标准端口：

```
zone "movie.edu" {
    type slave;
    masters port 5353 { 192.249.249.1; };
    file "bak.movie.edu";
};
```

或者，如果你的 BIND 9 名字服务器有多个主名字服务器，能够使用下面的语句使每一个主名字服务器在不同的端口上监听：

```
zone "movie.edu" {
    type slave;
    masters { 192.249.249.1 port 5353; 192.253.253.1 port 1053; };
    file "bak.movie.edu";
};
```

BIND 9 甚至允许你发送 NOTIFY 消息到非标准端口。为了使你的主名字服务器在古怪的端口上通知它的所有辅名字服务器，可以使用下列语句：

```
also-notify port 5353 { 192.249.249.9; 192.253.253.9; }; // zardoz 的两个地址
```

通过不同的端口通知每个辅名字服务器，可使用下列语句：

```
also-notify { 192.249.249.9 port 5353; 192.249.249.1 port 1053; };
```

如果你的辅名字服务器需要使用一个特定的本地网络接口发送查询（也许一台主名字服务器只能从众多地址中的一个来确认它），可使用 *query-source* 子语句。

```
options {
    query-source address 192.249.249.1;
};
```

注意，参数不是地址匹配列表，它是一个单个的IP地址。也可以为查询使用指定一个特定的源端口：

```
options {
    query-source address 192.249.249.1 port 53;
};
```

BIND 的默认操作是使用任何一个可到达目的地的网络接口，以及一个随机、非特权端口，例如：

```
options {
    query-source address * port *;
};
```

注意，*query-source* 只对基于UDP的查询有用；基于TCP的查询总是按照路由表选择源地址，并且使用随机的源端口。

还有一个与之相似的 *transfer-source* 子语句，它控制着区传送使用的源地址。在 BIND 9 中，该子语句也作用于辅名字服务器的 SOA 查询和转发动态更新：

```
options {
```

```
    transfer-source 192.249.249.1;
};
```

与*query-source*一样，其参数仅仅是一个不含关键字*address*的单一IP地址。在BIND 8 中，没有 *port* 修饰字。在 BIND 9 中，你可以指定一个源端口：

```
options {
     transfer-source 192.249.249.1 port 1053;
};
```

然而，源端口只对基于 UDP 的数据流有用（例如，SOA 查询和转发动态更新。）

transfer-source 也能够用做 *zone* 子语句，它只作用于该区的传送（在 BIND 9 中还包括 SOA 查询和动态更新）：

```
zone "movie.edu" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.movie.edu";
    transfer-source 192.249.249.1;    // 为了 movie.edu 的传送，总是使用同一
                                      // 网络上的 IP 地址
};
```

最后，比如BIND 9.1.0，甚至存在允许你控制使用哪个地址发送NOTIFY消息的子语句：*notify-source*。这给多宿主名字服务器带来了极大的方便，因为辅名字服务器只接受针对某个区的NOTIFY消息，限制是该消息的源IP地址出自该区 *masters* 子语句。*notify-source* 的语法与其他 *—source* 子语句的语法相似，例如：

```
options {
    notify-source 192.249.249.1;
};
```

与 *transfer-source* 一样，*notify-source* 能够指定一个源端口，并可用做只作用于该区的 *zone* 语句。

```
zone {
    type slave;
    masters { 192.249.249.3; };
    file "bak.movie.edu";
    notify-source 192.249.249.1 port 5353;
};
```

配置 IPv6 作为传输

默认情况下，BIND 9 名字服务器不会监听基于 IPv6 的查询。为配置名字服务器使其监听在本地主机的 IPv6 网络接口上，可使用 *listen-on-v6* 子语句：

```
options {
    listen-on-v6 { any; };
};
```

与 IPv4 不同，*listen-on-v6* 子语句只接受 *any* 和 *none* 为参数。然而，你能够配置 BIND 9 名字服务器用 *port* 修饰字在非标准端口上监听，或者甚至是多个端口。

```
options {
    listen-on-v6 port 1053 { any; };
};
```

当然，默认端口是 53。

你也能够用 *transfer-source-v6* 子语句来确定你的名字服务器使用哪个 IPv6 地址作为对外查询的源端口，例如：

```
options {
    transfer-source-v6 222:10:2521:1:210:4bff:fe10:d24;
};
```

或者：

```
options {
    transfer-source-v6 port 53 222:10:2521:1:210:4bff:fe10:d24;
};
```

默认是使用可到达目的地的网络接口的源地址，以及一个随机、非特权的端口。同 *transfer-source* 一样，你能够使用 *transfer-source-v6* 作为一个区子语句。而且源端口只作用于 SOA 查询和转发动态更新。

最后，BIND 9.1.0 及其后续版本允许用 *notify-source* 子语句来确定在 NOTIFY 消息中使用哪个 IPv6 地址。毫不意外，IPv6 子语句被称为 *notify-source-v6*：

```
options {
    notify-source-v6 222:10:2521:1:210:4bff:fe10:d24;
};
```

类似于 *transfer-source-v6* 子语句，能够在 *zone* 语句中指定源端口并使用该子语句。

IPv6 的前向和反向映射

显然，现有的 A 记录并不与 IPv6 的 128 位地址相符；BIND 希望 A 记录的数据是点分字节格式的 32 位地址。

针对该问题，IETF 提出了一个简单的解决方案，该方案在 RFC 1886 中说明。它用一个新的地址记录 AAAA 来存储 128 位的 IPv6 地址，并且新建一个 IPv6 反向映射域 *ip6.int*。这个解决方案非常简单易懂，以至于可在 BIND 4.9 中实现。不幸的是，不是每个人都喜欢简单的解决方案，因此他们提出一个更加复杂的方案。这个方案我们将待会儿介绍，包括新的 A6 和 DNAME 记录，并且在实现前要求对 BIND 名字服务器的现有代码进行彻底检查。

现在并不赞成使用老的 AAAA 记录和 *ip6.int*，但是仍有足够多的 IPv6 软件在使用它们，而没有采用新的方案，因此理解这两种方法都是很重要的。

AAAA 和 ip6.int

在 RFC 1886 中描述的简单方法就是使用长度为 A 记录长度四倍的地址记录，那就是 AAAA(读作“ quad ”A)记录。AAAA 记录使用前面所述的 IPv6 记录的文本格式作为其记录专用数据（record-specific data）。因此你可以看到如下所示的 AAAA 记录：

```
ipv6-host    IN    AAAA    4321:0:1:2:3:4:567:89ab
```

RFC 1886 同时建立了新的 IPv6 地址反向映射名字空间 *ip6.int*。在 *ip6.int* 下的每一级子域代表 128 位地址中的 4 位，并用十六进制数字编码，正如在 AAAA 记录的记录专用数据中一样。最低位出现在域名的最左边。与 AAAA 记录中的地址格式不同，它不允许省略开头的零，因此在 *ip6.int* 下，与完整的 IPv6 地址相对应的域名总包含 32 个十六进制数字和 32 级子域。在前面的例子里，与地址相应的域名如下：

```
b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.0.0.0.0.1.2.3.4.ip6.int.
```

这些域名含附加的 PTR 记录，如同在 *in-addr.arpa* 下的域名一样：

```
b.a.9.8.7.6.5.0.4.0.0.0.3.0.0.0.2.0.0.0.1.0.0.0.0.0.0.0.1.2.3.4.ip6.int.  IN  PTR
mash.ip6.movie.edu.
```

A6、DNAME、位串标号和 ip6.arpa

上面的方法十分简单。IPv6的前向和反向地址映射由于使用了两个新的地址记录A6和DNAME而变得复杂得多，现在这也成为一个正式的方法。RFC 2874和RFC 2672分别描述了 A6 和 DNAME 记录。BIND 9.0.0 是最早支持这些记录的 BIND 版本。

替换 AAAA 记录和 *ip6.int* 反向映射方案的主要原因是因为它们使网络重编号（renumber）变得困难。例如，如果一个组织打算改变NLA（下级聚集器），它将必须改变区数据文件中的所有AAAA记录，因为NLA标识符占据着IPv6地址中的24位（注11）。或者，想像一下TLA正在改变的一个NLA：这将给客户的区数据造成混乱。

A6 记录和前向映射

为使重编号更简单，A6记录能够仅仅指定IPv6地址的一部分，比如分配给主机网络接口的最后64位（接口ID），而后用一个符号域名指向地址的其他部分。这允许区管理员仅指定在其控制下的那部分地址。为了构建一个完整地址，解析器或名字服务器必须遵循从主机域名到 TLA ID 的 A6 记录链。而且，如果一个地区网络与多个 NLA 连接，或者是一个 NLA 与多个 TLA 相连，那么该链可能分支。

例如，A6记录：

```
$ORIGIN movie.edu.
drunkenmaster  IN   A6   64  ::0210:4bff:fe10:0d24  subnet1.v6.movie.edu.
```

指定 *drunkenmaster.movie.edu* 的IPV6地址的最后64位（64是未在该A6记录中指定的前缀所含的位数），并且可以通过查询 *subnet1.v6.movie.edu* 的一个 A6 记录来发现余下的 64 位。

依次，*subnet1.v6.movie.edu* 指定了64位前缀中的后16位（SLA ID），就像查询下一个 A6 记录的域名一样，这64位前缀我们并未在 *drunkenmaster.movie.edu* 的A6地址中指定：

```
$ORIGIN v6.movie.edu.
subnet1  IN  A6  48  0:0:0:1::  movie-u.nla-a.net.
```

注 11： 并且，新 NLA 可以使用一个不同的 TLA，这意味着多于 16 位要修改。

```
subnet1  IN  A6  48  0:0:0:1::  movie.nlab.net.
```

在*subnet1.v6.movie.edu*的记录专用数据中，前缀开始的48位被设为零，因为在这里它们没有意义。

实际上，这些记录告诉我们去查询下两个A6记录，一个在*movie-u.nla.a.net*，而另一个在*movie.nlab.net*。那是因为电影大学与两个NLA相连，分别是NLA A 和NLA B。在NLA A的区里，我们将发现：

```
$ORIGIN nla-a.net.
movie-u  IN  A6   40  0:0:21::  nla-a.tla-1.net.
```

这意味着在NLA ID字段中的8位地区ID模式，由NLA A为电影大学的网络而设置。你还可以看出NLA ID字段是层次化的，其中包括由对应TLA设置的我们的NLA ID和我们网络的ID。因为NLA分配了我们的地区ID，而NLA ID的余下部分却由对应的TLA分配，所以我们希望在我们的NLA的区数据里只能看到我们的地区ID。NLA ID的剩余部分将在其TLA区的某个A6记录中出现。

在NLA B的区中，我们将发现下列显示了我们网络的地区ID的记录：

```
$ORIGIN nlab.net.
movie  IN  A6  40  0:0:42::  nlab.tla-2.net.
```

在TLA的区中，我们将发现：

```
$ORIGIN tla-1.net.
nla-a  IN  A6  16  0:10:2500::  tla-1.top-level-v6.net.
```

和：

```
$ORIGIN tla-2.net.
nlab   IN  A6  16  0:19:6600::  tla-2.top-level-v6.net.
```

最后，在顶级IPv6地址注册区，我们发现这个记录，该记录向我们显示分配给TLA 1和TLA 2的TLA ID：

```
$ORIGIN top-level-v6.net.
tla-1  IN  A6  0  222::
tla-2  IN  A6  0  242::
```

沿着A6记录的这条链，名字服务器能够汇聚*drunkenmaster.movie.edu*的两个IPv6地址的所有128位，如下所示：

```
222:10:2521:1:210:4bff:fe10:d24
242:19:6642:1:210:4bff:fe10:d24
```

第一个地址使用一个通过 TLA 1 和 NLA A 到达电影大学网络的路由，而第二个地址使用通过 TLA 2 和 NLA B 到达电影大学网络的路由。（为冗余考虑，我们连接到两个 NLA。）记住，如果 TLA 1 为 NLA A 而改变它的 NLA 分配，TLA 1 只需要在区数据里改变针对 *nla-a.tla-1.net* 的 A6 记录，这个改变将通过 NLA 进入到所有的 A6 链。这使得 IPv6 网络上的地址分配管理非常方便，而且也使得容易改变 NLA。

警告：如果名字服务器出现在一个 NS 记录里，并且它拥有一个或多个 A6 记录，这些 A6 记录应当指定 IPv6 地址的 128 位。当解析器或名字服务器需要与远程名字服务器对话来解析该远程名字服务器的 IPv6 地址的一部分时，出现死锁问题，而这就可帮助避免该问题。

DNAME 记录和反向映射

现在你已经看到 A6 记录的前向映射是如何工作的了，那么让我们来看一看 IPv6 地址的反向映射是如何工作的？不幸的是，由于 A6 记录，这并没有 *ip6.int* 那样简单。

反向映射 IPv6 地址包括 RFC 2672 中描述的 DNAME 记录和 RFC 2673 中介绍的位串标号（*bitstring labe()*）。DNAME 记录与通配符 CNAME 记录有一点类似，它们用来替换域名的一个后缀。例如，如果我们起初在电影大学使用域名 *movieu.edu*，但后来改成了 *movie.edu*，我们能够使用下面的区来替换旧的 *movieu.edu* 区：

```
$TTL 1d
@    IN  SOA    terminator.movie.edu.  root.movie.edu. (
    2000102300
    3h
    30m
    30d
    1h  )

    IN   NS     terminator.movie.edu.
    IN   NS     wormhole.movie.edu.

    IN   MX     10 postmanrings2x.movie.edu.

    IN   DNAME  movie.edu.
```

movieu.edu 区中的 DNAME 记录应用于以 *movieu.edu* 结尾的任意域名，而不包括 *movieu.edu* 本身。与 CNAME 记录不同，DNAME 记录能够与相同域名的其他记录

类型共存，条件是只要这些记录不是CNAME或其他DNAME记录。尽管，DNAME记录的所有者可能不含任何子域。

当*movieu.edu*名字服务器接收到对以*movieu.edu*结尾的任意域名的查询时（比如，*cuckoosnest.movieu.edu*），DNAME记录告诉*movieu.edu*名字服务器通过用*movie.edu*替换*movieu.ed*的方法，来合成从*cuckoosnest.movieu.edu*到*cuckoosnest.movie.edu*的别名：

```
cuckoosnest.movieu.edu.  IN  CNAME  cuckoosnest.movie.edu.
```

这有点类似*sed*的“s”（替换）命令。*movieu.edu*名字服务器用这个CNAME记录来应答。如果它正在响应的是一个新名字服务器的话，*movieu.edu*名字服务器也在应答中发送DNAME记录，而接收名字服务器能够从缓存的DNAME来合成它自己的CNAME记录。

位串标号是IPv6反向映射中的另一半。简单地说，位串标号就是一种表示一个域名中二进制标号（比如，1位）长序列的压缩方法。例如，你想允许对IP地址的任意两位之间的位授权。那将迫使你表示地址的每一位为域名中的一个标号。为表示一个IPv6地址，那将要求域名的标号数超过128。

位串标号把连续标号中的位串连成一个较短的十六进制、八进制、二进制或点分字节字符串。为了区分它与普通标签，这个字符串被封装在记号“\[”和“]”之间以区别于传统的标号，并以一个确定字符串基数的字母开头：*b*代表二进制，*o*代表八进制，而*x*代表十六进制。

下面是与*drunkenmaster.movie.edu*的两个IPv6地址相对应的位串：

```
\[x022200102521000102104bfffe100d24]
\[x024200196642000102104bfffe100d24]
```

注意，与IPv6地址的文本表示一样，字符串的开头是最有效位，但是在*in-addr.arpa*域中，标号以反序排列。尽管如此，这两个位串标号只是传统域名的不同编码，该传统域名以下列串开头：

```
0.0.1.0.0.1.0.0.1.0.1.1.0.0.0.0.0.0.0.0.1.0.0.0.0.1.1.1.1.1.1.1...
```

也要注意，地址中的所有32个十六进制数字 —— 你不能丢弃开头的零，这是因为没有冒号来按四个数字一组分隔。

位串标号也能够表示IPv6地址的部分，在这种情况下，你需要指定字符串中的有效位数，用斜杠与字符串分开。因此TLA 1的TLA ID是*\[x0222/16]*。

DNAME和位串标号一起被用来匹配编码一个IPv6地址的长域名的一部分，并且被用来重复修改区中被查询的域名，该区在管理拥有IPv6地址主机的组织的控制之下。

想像一下我们反向映射了*\[x024200196642000102104bfffe100d24].ipv6.arpa*，这是与*drunkenmaster.movie.edu*的网络接口（通过TLA 2和NLA B到达）相对应的域名。根名字服务器也许将我们的名字服务器指向为包含以下记录的*ip6.arpa*名字服务器：

```
$ORIGIN ip6.arpa.
\[x0222/16]    IN     DNAME      ip6.tla-1.net.
\[x0242/16]    IN     DNAME      ip6.tla-2.net.
```

这些记录中的第一个与我们正在查询的域名开头匹配，因此*ip6.arpa*名字服务器用别名来回答我们的名字服务器，其内容如下：

```
\[x024200196642000102104bfffe100d24].ip6.arpa.  IN  CNAME
\[x00196642000102104bfffe100d24].ip6.tla-2.net.
```

注意，省去了地址中的头4个十六进制数字（最有意义的16位），并且因为我们知道这个地址属于TLA 2，所以别名的目标结尾现在是*ipv6.tla-2.net*。在*ipv6.tla-2.net*中，我们发现：

```
$ORIGIN ip6.tla-2.net.
\[x00196600/24]   IN   DNAME     ip6.nlab.net.
```

在我们的新查询中，转换域名：

```
\[x00196642000102104bfffe100d24].ip6.tla-2.net
```

为：

```
\[x42000102104bfffe100d24].ip6.nlab.net
```

紧接着，针对新域名，我们的名字服务器查询 *ipv6-nlab.net* 名字服务器。在 *ipv6-nlab.net* 区中的该记录如下：

```
$ORIGIN ip6.nlab.net.
\[x0042/8]    IN   DNAME    ip6.movie.edu.
```

将我们正在查询的域名转换成：

```
\[x000102104bfffe100d24].ip6.movie.edu
```

最后，*ip6.movie.edu* 区包含了 PTR 记录，该记录提供了我们寻找的主机域名：

```
$ORIGIN ip6.movie.edu.
\[x000102104bfffe100d24/80]  IN   PTR   drunkenmaster.ip6.movie.edu.
```

宽容一点来说，作为一个区管理员，你也许只负责维护类似于 *ip6.movie.edu* 中的 PTR 记录，即使是为顶级或下级聚集器创建从客户地址中“提取”合适的 NLA ID 或地区 ID 的 DNAME 记录也并不困难。即使每个主机有多个地址，为反向映射信息而使用单个的区数据文件也会十分方便，同时也使你不用修改区数据文件的全部即可转换 NLA。

本章内容：

- TSIG
- 保护名字服务器
- DNS 与 Internet 防火墙
- DNS 安全扩展

第十一章

安全

“我希望你已经固定了你的头发，”
他继续说着，当他们出发时。
“仅用通常的方法就可以办到，”爱丽丝微笑地说。
“但那几乎不够”，他急切地说。
“你可以看见这儿的风非常厉害。”
“你已经发明了不让你头发散乱的方法了吗？”
爱丽丝问。
“还没有，”武士回答道，
“但我已得到使其不倒的办法了。”

为什么应当关注你的DNS安全？为什么要特意去保护一种服务的安全，而这种服务主要是提供名字与地址间映射的？让我们给你讲一个故事。

1997 年 7 月，若干天中的两个时段，在浏览器中输入 *www.internic.net* 的 Internet 用户认为他们将进入InterNIC的站点，而实际上他们却到达了一个属于AlterNIC的站点。（AlterNIC 运行维护了根名字服务器的备用集，这些根名字服务器授权额外的顶级域，其名字类似于*med*和*porn*。）这是如何发生的呢？ Eugene Kashpureff 曾运行了一个程序来毒害世界上主要名字服务器的缓存，使这些名字服务器认为*www.internic.net*的地址实际上就是 AlterNIC 的Web服务器地址，而后他被接受为 AlterNIC 会员。

Kashpureff 没有企图掩盖他的行为，用户所到达的 Web 站点明白无误地就是 AlterNIC 的站点，而非 InterNIC 的站点。但是想像一下，如果有人毒害你的名字服务器的缓存，将 *www.amazon.com* 或 *www.wellsfargo.com* 指向其本人的 Web 服务器，而不违反当地法律的强制权限。进一步地，想像一下你的用户输入他们的信用卡账号和终止期限。现在你应该考虑名字服务器的安全了吧。

保护你的用户免遭这些类型的攻击就需要 DNS 安全。DNS 安全分为几个方面。你可以保护 DNS 的事务（transaction）——查询、响应和名字服务器发送与接收的其他消息。例如，你可以保护你的名字服务器，使其拒绝查询、区传送请求，以及来自非授权地址的动态更新。你甚至可以通过数字签名来保护区数据。

因为 DNS 安全是 DNS 里最复杂的内容之一，我们将先易后难地介绍给大家。

TSIG

BIND 8.2 使用了一种保护 DNS 消息的新机制，称为“事务签名”（transaction sigature），或简写成 TSIG。TSIG 使用共享密钥和单向散列函数来鉴别 DNS 消息，特别是响应和更新。

现在，在 RFC 2845 中说明的 TSIG 的配置相当简单，对解析器和名字服务器的使用影响较轻，保护 DNS 消息（包括区传送）和动态更新显得非常灵活。（可以与我们本章最后将讨论的 DNS 安全扩展相比较。）

如果配置了 TSIG，名字服务器或更新者将在一个 DNS 消息的附加数据部分添加一个 TSIG 记录。TSIG 记录对 DNS 消息签名，证明消息发送者和接收者共享一个加密密钥，并保证消息在离开发送者后不被修改（注 1）。

注 1： 严格来说，TSIG“签名”并不是真正意义上的密码签名，因为它不提供认可（nonrepudiation）。由于共享密钥的任何一个拥有者都可以生成一个签名消息，所以签名消息的接受者不能要求仅仅只有发送者能够发送这个签名消息（接受者也可以自己生成签名消息）。

单向散列函数

通过使用称为单向散列函数的特殊类型的数学公式，TSIG提供认证和数据的完整性检查。单向散列函数，也被称为加密校验和（checksum）或消息摘要（digest），它对任意大小的输入都计算出一个固定长度的散列值。单向散列函数的魔力体现在散列值的每一位都依赖于输入的每一位。若改变输入的某一位，散列值将急剧地变化，以至于对散列函数求反并获得产生指定散列值的输入的操作在计算上都是不可行的。

TSIG使用的单向散列函数称为MD5。特别是，它使用了MD5的变种HMAC-MD5。HMAC-MD5的特点是使用密钥，其128位的散列值不仅依赖于输入，还依赖于密钥。

TSIG 记录

因为你不需要知道TSIG记录的语法，所以我们不会详细地讲述它：TSIG是一个“元记录”（meta-record），它从未在区数据中出现过，而且解析器或名字服务器也从未缓存过它。签名者将TSIG记录添加到DNS消息中，而接收者在做正常操作之前（例如，缓存消息中的数据），需要从 DNS 消息中删除 TSIG 记录并验证该记录。

然而，你应当知道，TSIG 记录包括一个散列值，该散列值是根据整个 DNS 消息和一些附加字段计算出来的。（我们所说的计算是指原始的二进制DNS 消息以及附加字段，通过HMAC-MD5算法计算而产生散列值的这个过程。）通过在签名者和检查者之间共享密钥，散列值被密钥化了。散列值的验证证明了DNS消息是由共享密钥的一个持有者所签名的，并且在签名后消息未被修改。

TSIG 记录中的附加字段包括 DNS 消息签名的时间，这有助于对付重复攻击。这种攻击中，黑客捕获一个已签名的授权事务（比如说，要求删除一个重要资源记录的动态更新），并且后来重复发送该事务。已签名 DNS 消息的接收者检查签名时间，以确定该消息处于可允许的范围内（TSIG 记录的另一个字段）。

配置 TSIG

在为了认证而使用 TSIG 前，我们需要在事务处理的双方配置一个或多个 TSIG 密钥。例如，如果我们打算使用TSIG 保护*movie.edu*的主、辅名字服务器之间的区传送，就需要用同一个密钥来配置这两种名字服务器：

```
key terminator-wormhole.movie.edu. {
    algorithm hmac-md5;
    secret "skrKc4Twy/cIgIykQu7JZA==";
};
```

在这个例子中，*key* 语句的参数 *terminator-wormhole.movie.edu* 实际上是密钥的名称，尽管它看上去像一个域名。（密钥名称是以与域名相同的格式在DNS 消息中编码的。）TSIG 的 RFC 建议你在两台主机使用密钥后，对密钥命名。该 RFC 也建议你对不同的主机对使用不同的密钥。

密钥的名字（不仅仅是密钥指向的二进制数据）在事务处理双方的一致性是非常重要的。如果不一致，接收者将试图验证 TSIG 记录，并且发现自己不知道 TSIG 记录所声明的用于计算散列值的密钥。那会导致如下的错误：

```
Nov 21 19:43:00 wormhole named-xfer[30326]: SOA TSIG verification from server
[192.249.249.1], zone movie.edu: message had BADKEY set (17)
```

到目前为止，算法总是使用 HMAC-MD5。secret 是以 64 为基数的二进制密钥的编码。你可以使用 *dnssec-keygen* 程序或 *dnskeygen* 程序创建以 64 为基数的编码密钥，BIND 9 包含 *dnssec-keygen* 程序，而 BIND 8 包含 *dnskeygen* 程序。下面介绍应如何使用两者中更为简单的 *dnssec-keygen* 程序创建密钥：

```
#  dnssec-keygen -a HMAC-MD5 -b 128 -n HOST terminator-wormhole.movie.edu.
Kterminator-wormhole.movie.edu.+157+28446
```

–a 选项将密钥使用的算法名字作为参数。（因为 *dnssec-keygen* 能够产生其他类型的密钥，这一点在 DNSSEC 部分我们将看到，所以这是必需的。）*–b* 选项使用密钥长度作为参数；RFC 推荐使用 128 位的密钥。*–n* 使用 HOST（产生的密钥类型）作为参数。（DNSSEC 使用 ZONE 密钥。）最后的参数是密钥的名字。

dnssec-keygen 和 *dnskeygen* 都在各自的工作目录下创建文件，这些文件包含所生成的密钥。*dnssec-keygen* 在标准输出上打印文件的基本名。在本例中，*dnssec-keygen* 创建了文件 *Kterminator-wormhole.movie.edu.+157+28446.key* 和 *Kterminator-wormhole.movie. edu.+157+28446.private*。可以从这两个文件中得到密钥。有趣的数字（157 和 28446）是密钥的 DNSSEC 算法代号（157 是 HMAC-MD5）和密钥指纹（28446），密钥指纹是为识别密钥而对其进行计算的散列值。TSIG 中密钥指纹并不是特别有用，但是针对每个区，DNSSEC 支持多密钥，因此通过指纹来识别你所指定的密钥是非常重要的。

Kterminator-wormhole.movie.edu.+157+28446.key 包含：

```
terminator-wormhole.movie.edu. IN KEY 512 3 157 skrKc4Twy/cIgIykQu7JZA==
```

Kterminator-wormhole.movie.edu.+157+28446.private 包含：

```
Private-key-format: v1.2
Algorithm: 157 (HMAC_MD5)
Key: skrKc4Twy/cIgIykQu7JZA==
```

你也可以选择你自己的密钥，并且使用 *mmencode* 对它进行以64为基数的编码：

```
% mmencode
foobarbaz
Zm9vYmFyYmF6
```

正如子语句所提示的那样，因为实际的二进制密钥是一个secret，所以在将其传输给我们的名字服务器时必须格外小心（例如，通过 *ssh*），并且要确信不是每个人都可以读到密钥。为了做到这一点，我们可以使 *named.conf* 文件变成非广泛可读，或者使用 *include* 语句从其他非广泛可读文件中读 *key* 语句：

```
include "/etc/dns.keys.conf";
```

最后一个问题是我们在使用TSIG时经常看到的：时间同步。TSIG记录中的时间戳在防止重复攻击时是很有用的，但是初始的时间戳也会使我们自己失败，因为我们名字服务器的时钟没有同步。产生的错误消息如下：

```
Nov 21 19:56:36 wormhole named-xfer[30420]:SOA TSIG verificatiors from server
[192.249.249.1], zone movie.edu: BADTIME (-18)
```

通过使用NTP（网络时间协议），很快可解决该问题（注2）。

使用TSIG

现在，我们已不怕麻烦地配置了名字服务器，使其使用TSIG密钥，或许应当在配置TSIG密钥时考虑使用密钥的一些任务。在BIND 8.2及其后续版本的名字服务器中，我们可以利用TSIG保护查询、响应、区传送和动态更新。

注2： 关于NTP的信息请参见网址 *http://www.eecis.udel.edu/~ntp*。

配置的关键是*server*语句的*keys*子语句，*keys*子语句告诉名字服务器对发送给特定远程名字服务器的查询和区传送请求进行签名。例如，*server*语句告诉本地名字服务器*wormhole.movie.edu*使用密钥*terminator-wormhole.movie.edu*对所有发送给192.249.249.1（*terminator.movie.edu*）的请求进行签名：

```
server 192.249.249.1 {
    keys { terminator-wormhole.movie.edu.; };
};
```

现在，在*terminator.movie.edu*上，我们可以限制区传送，使之成为用*terminator-wormhole.movie.edu*密钥签名的区传送：

```
zone "movie.edu" {
    type master;
    file "db.movie.edu";
    allow-transfer { key terminator-wormhole.movie.edu.; };
};
```

*terminator.movie.edu*也对区传送签名，它允许*wormhole.movie.edu*执行区传送，条件是必须通过验证。

通过*allow-update*和*update-policy*子语句，也可以利用TSIG限制动态更新，这一点我们将在最后一章介绍。

BIND 8.2附带了*nsupdate*程序，其后的版本支持基于TSIG签名的动态更新。如果你有使用*dnssec-keygen*创建的密钥文件，你可以指定任一文件作为*nsupdate*的*-k*选项参数。下面是应如何使用BIND 9版本的*nsupdate*：

```
%  nsupdate -k Kterminator-wormhole.movie.edu.+157+28446.key
```

或者：

```
%  nsupdate -k Kterminator-wormhole.movie.edu.+157+28446.private
```

BIND 8.2或后续版本的*nsupdate*与BIND 9的*nsupdate*相比，语法上存在轻微的差别：*-k*选项采用使用冒号分隔的目录和密钥名作为参数：

```
%  nsupdate -k /var/named:terminator-wormhole.movie.edu.
```

如果你手边没有*nsupdate*（也许你正在另一台主机运行*nsupdate*），使用BIND 9的*nsupdate*，你仍旧可以在命令行指定密钥名和secret：

```
% nsupdate -y terminator-wormhole.movie.edu.:skrKc4Twy/cIgIykQu7JZA==
```

密钥名是 *-y*选项的第一个参数，其后跟随一个冒号和以64为基数的编码secret。你不需要引证（quote）secret，因为以 64 为基数的编码值不能包含 shell 的元字符（metacharacter），但是如果你喜欢，你也能够引证 secret。

Michael Fuhr的Net:DNS Perl模块也让你发送TSIG签名的动态更新和区传送请求。要了解 Net:DNS 的更多信息，请查看十五章。

现在我们已有一套便利的机制来保护DNS事务安全，下面我们讨论如何保护整个名字服务器。

保护名字服务器

BIND 4.9 引入了好几个重要的安全特性来帮助你保护名字服务器。BIND 8 和 9 延续了这个传统，并且引入了更多的安全特性。如果你的名字服务器运行于 Internet 上，那么这些特性就尤为重要，当然，它们对于内部名字服务器来说也是很有用的。

我们将首先讨论你可以在安全性要求较高的名字服务器上采取的措施。然后，我们将描述一种模式，它把名字服务器划分为两部分，一部分只服务解析器，而另一部分则回答其他名字服务器的查询。

BIND 版本

能够增强名字服务器安全性的最重要的一个措施就是，使用最新版本的 BIND。BIND 8.2.3 之前的所有版本都很容易遭受一些广为人知的攻击。请在 *http://www.isc.org/products/BIND/bind-security.html* 站点检查不同 BIND 版本的 ISC 漏洞列表更新。

但是不要在这里就停下来：任何时刻都可能出现新的攻击，所以你必须尽你所能地了解所有最新发现的 BIND 的弱点和 BIND 最“安全”的版本。要做到这一点，一个好方法就是定期阅读新闻组 *comp.protocols.dns.bind*。

BIND 版本与安全相关的另一方面就是：如果一个黑客能够轻易发现你所运行的

BIND版本，他或许可以根据你的BIND版本定制攻击。而且，你也许还不知道它，自从BIND 4.9以后，BIND名字服务器针对某个查询的应答中含有它们的版本信息。如果你在与域名 *version.bind* 相关的CHAOSNET类中查找TXT记录，BIND将返回下列信息：

```
%   dig txt chaos version.bind.

; <<>> DiG 9.1.0 <<>> txt chaos version.bind.
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34772
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;version.bind.                  CH      TXT

;; ANSWER SECTION:
version.bind.           0       CH      TXT     "9.1.0"
```

为解决这一问题，BIND 8.2之后的版本允许你定制针对 *version.bind* 查询的名字服务器响应。

```
options {
    version "None of your business";
};
```

当然，接收到类似“None of your business”的响应将向警觉的黑客泄露你的情况，即，你可能运行了BIND 8.2以上的版本，这仍旧给黑客留下了许多攻击的可能。

限制查询

直到BIND 4.9，管理员一直无法控制哪些人可以查询他们名字服务器上的数据。这样做是有一定道理的，创建DNS最初的目的就是让整个Internet都能很容易地找到所需信息。

不过Internet不再是一个友好的地方了。特别是那些运行Internet防火墙的人可能有合理的需求要隐藏其域名空间中特定部分不被外界所知，但是对一些特定用户开放。

BIND 8和9的 *allow-query* 子语句允许根据IP地址来创建允许查询服务器的访问列表。这个访问列表可以应用于特定的区，也可以应用于该服务器所收到的任何查询。特别是，该访问列表指定了允许哪些IP地址向名字服务器发送查询。

限制所有查询

allow-query 子语句的全局形式如下：

```
options {
    allow-query { address_match_list; };
};
```

所以，如果要限制你的名字服务器只回答从 *movie.edu* 的两个网络发送的查询，我们使用如下语句：

```
options {
    allow-query { 192.249.249/24; 192.253.253/24; 192.253.254/24; };
};
```

限制关于某个特定区的查询

BIND 8 也允许你对于某个特定的区使用访问列表。在这种情况下，只需在你要加以保护的那个区对应的 *zone* 语句中使用 *allow-query* 子语句即可：

```
acl "HP-NET" { 15/8; };

zone "hp.com" {
     type slave;
     file "bak.hp.com";
     masters { 15.255.152.2; };
     allow-query { "HP-NET"; };
};
```

任何类型的权威名字服务器，无论是主名字服务器还是辅名字服务器，都可以对某个区使用访问列表。针对区的访问列表比全局配置中规定的访问列表优先级要高。针对区的访问列表可能比全局访问列表还要更加宽容一些。如果没有定义任何针对区的访问列表，任何全局访问列表都会有效。

在 BIND 4.9 中，这个功能是由 *secure_zone* 记录提供的。它不仅限制对于单个资源记录的查询，它也限制区传送。（在 BIND 8 和 9 中限制区传送是分开的。）但是 BIND 4.9 名字服务器没有机制来限制谁可以向服务器发送有关不是服务器权威数据的查询；这个 *secure_zone* 机制只适用于权威数据。

为了使用 *secure_zone* 机制，需要在主名字服务器上你的区数据中包括一个或多个特

殊的TXT（text，文本）记录。这些记录很方便地就可以自动传送到辅名字服务器那里。当然，只有BIND 4.9的辅名字服务器才可以理解它们。

说这些TXT记录很特别，是因为它们和伪域名*secure_zone*相关联，并且资源记录专用（record-specific）数据的格式很特别，要么是：

```
address:mask
```

要么是：

```
address:H
```

第一种格式中的*address*是点分字节格式的IP网络，你允许该IP网络访问区中的数据。*mask*则是该地址的网络掩码。如果你希望所有网络15/8上的主机都可以访问你的区数据，就使用15.0.0.0:255.0.0.0。如果你只希望从15.254.0.0到15.255.255.255之间的地址可以访问你的区数据，那就使用15.254.0.0:255.254.0.0。

第二种格式指定了允许访问区数据的特定主机的IP地址。这里的H等价于网络掩码255.255.255.255；换句话来说，32位地址中的每一位都是有效的。于是15.255.152.4:H就允许IP地址为15.255.152.4的主机查找区中的数据。

如果想限制只有电影大学网络中的主机才可以查询*movie.edu*域中的信息，就可以在*movie.edu*的primary的*db.movie*文件中添加下面几行：

```
secure_zone     IN     TXT     "192.249.249.0:255.255.255.0"
secure_zone     IN     TXT     "192.253.253.0:255.255.255.0"
secure_zone     IN     TXT     "192.253.254.0:255.255.255.0"
secure_zone     IN     TXT     "127.0.0.1:H"
```

注意，在访问列表中也添加了127.0.0.1。这样，解析器就可以查询其本地的名字服务器了。

如果你忘记了*:H*，那么你就会看到下面的系统日志消息：

```
Aug 17 20:58:22 terminator named[2509]: build_secure_netlist
       (movie.edu): addr (127.0.0.1) is not in mask (0xff000000)
```

同样，要注意*secure_zone*记录只适用于它们自己所在的区，也就是*movie.edu*。如果你想阻止那些关于该服务器上其他区中数据的未授权的查询，你就必须在这些区所在的主名字服务器上也添加*secure_zone*记录。

防止未授权的区传送

确保只有你自己真正的辅名字服务器才可以从你的名字服务器得到区数据，这比控制谁可以查询你的名字服务器更加重要。在那些可以查询你的名字服务器的远程主机上的用户只能查询它们早已经知道的域名的记录（如，IP地址记录），一次一个。而能从你的名字服务器进行区传送的用户则可以列出你的区中所有的主机。允许随便什么人都能打电话到你公司的总机，查询 John Q. Cubicle 的电话号码，这与把你公司的电话号码簿送给他是不一样的。

BIND 8 和 9 的 *allow-transfer* 子语句和 4.9 的 *xfrnets* 指令允许管理员对于区传送使用访问列表。*allow-transfer* 作为 *zone* 的子语句可以限制有关某个特定区的传送，而作为 *options* 的子语句则可以限制所有的区传送。它使用一个地址匹配列表来作为它的参数。

我们的 *movie.edu* 区的辅名字服务器的 IP 地址为 192.249.249.1、192.253.253.1（*wormhole.movie.edu*）、192.249.249.9 和 192.253.253.9（*zardoz.movie.edu*）。下面的 *zone* 语句：

```
zone "movie.edu" {
     type master;
     file "db.movie.edu";
     allow-transfer { 192.249.249.1; 192.253.253.1; 192.249.249.9; 192.253.253.9; };
};
```

只允许那些辅名字服务器从主名字服务器传输 *movie.edu* 区的数据。注意，因为 BIND 8 和 9 的默认配置是允许所有 IP 地址进行区传送，而且因为黑客可以很容易地从辅名字服务器那里获得区的数据，你也许应该在辅名字服务器上也加上以下的 *zone* 语句：

```
zone "movie.edu" {
     type slave;
     masters { 192.249.249.3; };
     file "bak.movie.edu";
     allow-transfer { none; };
};
```

BIND 8 和 9 也允许你建立区传送的全局访问列表。这将适用于任何没有用 *zone* 语句明确定义访问列表的区。例如，我们可能想限制只有内部的 IP 地址才能进行区传送：

```
options {
      allow-transfer { 192.249.249/24; 192.253.253/24; 192.253.254/24; };
};
```

BIND 4.9 的 *xfrnets* 指令对所有区传送使用访问控制列表。*xfrnets* 使用允许从名字服务器进行区传送的网络或IP地址作为参数。网络是以点分字节形式表示的，例如：

```
xfrnets 15.0.0.0 128.32.0.0
```

就会只允许位于 A 类网络 15/8 或者 B 类网络 128.32/16 上的主机从这台名字服务器进行区传送。与 *secure_zone* 记录不同，这个限制应用于任何以该服务器为权威的区。

如果只想指定网络的一部分，或者是单个 IP 地址，你可以添加一个网络掩码。包括网络掩码的语法是 *network&netmask*。注意，在网络号和“ & ”符号之间没有空格，“ & ”和网络掩码之间也没有空格。

要把在前面例子里允许进行区传送的地址减少到 IP 地址 15.255.152.4 和子网 128.32.1.0，就要使用下面的 *xfrnets* 指令：

```
xfrnets 15.255.152.4&255.255.255.255 128.32.1.0&255.255.255.0
```

最后，正如在本章前面提到的那样，那些新的 BIND 8.2 以及后续版本的名字服务器允许限制到辅名字服务器的区传送，这些辅名字服务器在它们的请求中包含一个正确的事务签名。在主名字服务器上，需要用 *key* 语句定义密钥，而后在地址匹配列表中指定密钥：

```
key terminator-wormhole. {
    algorithm hmac-md5;
    secret "UNd5xYLjz0FPkoqWRymtgI+paxW927LU/gTrDyulJRI=";
};

zone "movie.edu" {
    type master;
    file "db.movie.edu";
    allow-transfer { key terminator-wormhole.; };
};
```

在辅名字服务器端需要配置辅名字服务器，使其使用相同的密钥对区传送请求签名：

```
key terminator-wormhole. {
```

```
        algorithm hmac-md5;
        secret "UNd5xYLjz0FPkoqWRymtgI+paxW927LU/gTrDyulJRI=";
    };

    server 192.249.249.3 {
        keys { terminator-wormhole.; };  // 用这个密钥给192.249.249.3
                                         // 的请求签名
    };

    zone "movie.edu" {
        type slave;
        masters { 192.249.249.3; };
        file "bak.movie.edu";
    };
```

对于一台可以通过 Internet 访问到的主名字服务器来说，你可能想限制只允许你的辅名字服务器才能进行区传送。你可能不需要担心防火墙内部的名字服务器，除非你担心自己的雇员会列出你所有的数据。

以最小权限运行 BIND

以超级用户的身份运行像 BIND 这样的网络服务器是非常危险的，然而 BIND 通常都是由超级用户来运行的。如果一个黑客发现名字服务器上有缺陷使得他可以读或写文件，他就可以像超级用户那样访问文件系统。如果他发现有漏洞使得他可以执行命令，那么他就会以超级用户的身份执行这些命令。

BIND 8.1.2以及后续版本包括允许你改变运行名字服务器所使用的用户和组的功能。这就允许你最小权限（要完成DNS工作所需的最少权限）地运行名字服务器。这样的话，如果有人通过名字服务器闯进你的系统，那么至少他不可能获得超级用户的权限。

BIND 8.1.2 以及后续版本也包括一个允许你对名字服务器执行 *chroot（）* 命令的选项：改变在它看来的文件系统的样子，这样它的根目录实际上是你主机文件系统上某个特定的子目录。这就很有效地把你的名字服务器限制在它自己的目录中，即使有些人闯入了该系统。

实现这些特性的命令行参数如下：

¬u 指定了在启动以后名字服务器要转到的用户名或用户ID，例如*named ¬u bin*。

-g 指定了在启动后名字服务器要转到的组或组 ID，例如 *named -g other*。如果单独使用 *-u* 而不使用 *-g*，那么名字服务器就会使用该用户所属的主组。BIND 9 名字服务器总是转到用户所属的主组，所以它们不支持 *-g* 选项。

-t 指定了名字服务器使用 *chroot()* 要转到的目录。

如果选择使用 *-u* 和 *-g* 选项，你就必须决定要使用什么用户和组。最好是创建一个新的用户和组供名字服务器运行使用，例如 *named*。因为名字服务器在放弃超级用户权限前会读取 *named.conf* 文件的内容，你不必去改变文件的权限。但是，你可能必须改变你的区的数据文件的权限和所有权，这样名字服务器运行时所作为的用户就可以读取这些文件。如果使用动态更新，就必须使得进行动态更新的区的数据文件对于名字服务器来说是可写的。

如果名字服务器被配置为要把日志记录到文件（而非记录到 *syslog*）中，在启动名字服务器前，要确保这些文件是存在的，并且对名字服务器来说是可写的。

其中的 *-t* 选项需要略多一些的特殊配置。特别是需要确保 *named* 使用的所有文件都在你所限制名字服务器使用的目录中。下面是建立你的 *chrooted* 环境所应遵循的过程，我们设想该环境建立在 */var/named* 之下（注 3）：

1. 如果 */var/named* 目录不存在，创建该目录。创建 *dev*、*etc*、*lib*、*usr* 和 *var* 子目录。在 *usr* 子目录里，创建 *sbin* 子目录。在 *var* 子目录里，创建 *named* 和 *run* 子目录：

```
#   mkdir /var/named
#   cd /var/named
#    mkdir -p dev etc lib usr/sbin var/named var/run
```

2. 复制 *named.conf* 到 */var/named/etc/named.conf*：

```
#   cp /etc/named.conf etc
```

3. 如果你正在运行 BIND 8，复制 *named-xfer* 二进制文件到 *usr/sbin/* 或 *etc* 子目录下（这依赖于你是否在 */usr/sbin* 或 */etc* 目录里找到该二进制文件）。

```
#   cp /usr/sbin/named-xfer usr/sbin
```

注 3： 这个过程是基于 Red Hat Linux 6.2 的，所以如果使用的是其他操作系统，那么你的行程会有所不同。

另外，你也可以将该二进制文件复制到*/var/named*目录里你喜欢的地方，并且使用*named-xfer*子语句来告诉*named*到哪里寻找该文件。记住从路径名里去掉*/var/named*，这是因为*named*读取*named.conf*时，*/var/named*就被*named*视为文件系统的根。（如果你正在运行BIND 9，因为BIND 9不使用*named-xfer*，跳过这一步。）

4. 在*chrooted*环境里创建*dev/null*：

```
#   mknod dev/null c 1 3
```

5. 如果你正在运行BIND 8，向*lib*子目录里复制标准C共享库和装入程序：

```
#    cp /lib/libc.so.6 /lib/ld-2.1.3.so lib
#    ln -s lib/ld-2.1.3.so lib/ld-linux.so.2
```

在你的操作系统上，路径名可能不同。BIND 9名字服务器是自包含的。

6. 编辑你的启动文件，用附加选项和选项参数：*—a /var/named/dev/log*来启动*syslogd*。在许多Unix现代版本中，*syslogd*是从*/etc/rc.d/init.d/syslog*启动的。当*syslogd*下一次重新启动时，它将创建*/var/named/dev/log*，并且*named*将日志记录到*/var/named/dev/log*。

   如果你的*syslogd*不支持*—a*选项，使用第七章描述的*logging*语句，将日志记录到*chrooted*目录下的文件里。

7. 如果你正在运行BIND 8并且使用*—u*或*—g*选项，在*etc*子目录里创建*passwd*和*group*文件，将*—u*和*—g*选项的参数映射到它们的数值（或只使用数值作为参数）：

```
#    echo "named:x:42:42:named:/:" > etc/passwd
#    echo "named::42" > etc/group
```

然后给系统的*/etc/passwd*和*/etc/group*文件添加条目。如果你正在运行BIND 9，你只能向系统的*/etc/passwd*和*/etc/group*文件添加条目，这是因为BIND 9名字服务器在调用*chroot()*之前读取它们需要的信息。

8. 最后，编辑你的启动文件，使用*—t*选项和选项参数：*—t /var/named*来启动*named*。与*syslogd*相似，许多Unix现代版本是从*/etc/rc.d/init.d/syslog*启动*named*的。

如果你喜欢使用*ndc*控制你的名字服务器，只要指定Unix域套接字作为*ndc*的*—c*选项参数，就可以继续使用它：

```
#    ndc -c /var/named/var/run/ndc reload
```

rndc 在使用 BIND 9 之前将继续工作，因为它只通过端口 953 与服务器对话。

拆分名字服务器

名字服务器实际上有两个主要功能：回答来自远程名字服务器的反复查询，并且回答来自本地解析器的递归查询。如果分离这些功能，将名字服务器的一部分专门用于回答反复查询，而另一部分则回答递归查询，这样就能更有效地保护那些名字服务器的安全。

“授权”名字服务器的配置

一些名字服务器回答来自 Internet 上其他名字服务器的非递归查询，因为这些名字服务器出现在将你的区授权给它们的 NS 记录中。我们把这些名字服务器称为“授权”名字服务器。

可以使用一些特殊的方法来保护你的授权名字服务器。但是，首先应该确保这些名字服务器不会收到任何递归查询（就是说，你没有配置任何解析器去使用这些服务器，而且没有任何名字服务器将它们作为转发器）。我们会很小心谨慎（让服务器即使对于递归查询也只给以非递归响应）排除掉你的解析器使用它们的可能。如果你确实有解析器使用你的被授权名字服务器，考虑建立只为你的解析器服务的另一类名字服务器，或者使用“名字服务器的二合一”的配置，本章后面会描述这二者。

一旦知道了你的名字服务器只回答来自其他名字服务器的查询，你就可以关闭递归功能了。这就减少了一种主要的攻击：最常见的欺骗攻击中就包括向被攻击的目标名字服务器发送一个递归查询，要它查找一个受黑客控制的服务器所服务的区中的域名，使得目标服务器查询由黑客所控制的名字服务器。你可以在一台 BIND 8 或 9 服务器上用下面的语句关闭递归功能：

```
options {
    recursion no;
};
```

在 BIND 4.9 的服务器上则要用：

```
options no-recursion
```

你也应该限制只对已知的slave服务器进行区传送，就像在本章前面“防止未授权的区传送”中讲到的那样。最后，你可能也想关闭fetch-glue。某些名字服务器会自动地试图解析NS记录中的任何名字服务器的名字；要阻止服务器这样做，使得你的名字服务器不要自己发送任何查询，在BIND 8服务器上要使用（BIND 9名字服务器默认地关闭fetch-glue）：

```
options {
      fetch-glue no;
};
```

在BIND 4.9上则要使用：

```
options no-fetch-glue
```

“解析”名字服务器配置

我们会把一台服务一个或多个解析器的名字服务器，或者是一台配置为另外一台名字服务器的转发器的服务器，称为一台“解析”名字服务器。同“授权”名字服务器不同，一台“解析”名字服务器不能拒绝递归查询，因此我们必须把它配置得有点不同，以便更好地保护它。因为我们知道名字服务器只应该从我们自己的解析器那里接受查询，这样就可以配置它拒绝来自其他解析器的查询。

只有BIND 8和9会允许我们限制哪个IP地址可以向我们的名字服务器发送任意的查询。（BIND 4.9服务器会允许我们通过*secure_zone* TXT记录，限制哪个IP地址可以向服务器发送有关其权威区的查询，但是实际上我们更担心的是有关其他区的递归查询。）这个*allow-query*子语句会限制查询只能来自我们的内部网络：

```
options {
    allow-query { 192.249.249/24; 192.253.253/24; 192.253.254/24; };
};
```

使用这个配置，惟一可以向我们的服务器发送递归查询的解析器，以及告诉你的名字服务器去查询其他名字服务器的解析器都是你内部的解析器，这些都被认为是相当友好的。

可以利用一个选项*use-id-pool*使我们的解析名字服务器更加安全：

```
options {
    use-id-pool yes;
```

```
};
```

BIND 8.2引入了*use-id-pool*。它告诉我们的名字服务器在查询中要小心使用随机的消息ID。通常，消息ID并非足够随机得可以防止那些尽力猜测我们名字服务器ID的暴力攻击，攻击的方法就是要欺骗应答。

ID pool代码成为BIND 9的标准部分，因此不需要在BIND 9名字服务器上指定它。

名字服务器的二合一

如果你只有一台名字服务器来通告你的区以及服务你的解析器该怎么办？或者说你无法支付购买另外一台服务器来运行名字服务器的开销该怎么办？这里仍然有一些选项可供你选择，其中一个是单独一台服务器的解决方案，它利用了BIND 8的灵活性。这个配置允许任何人查询你的服务器以获取权威区中的信息，但是只有你内部的解析器可以查询你的服务器以获取其他区的数据。虽然这个配置没有阻止远程的解析器向你的名字服务器发送递归查询，但是这些查询必须是针对你的权威区的，所以它们这些查询不会引导你的名字服务器去发送其他名字服务器的查询。

下面是完成这个功能的*named.conf*文件：

```
acl "internal" {
    192.249.249/24; 192.253.253/24; 192.253.254/24;
};

acl "slaves" {
    192.249.249.1; 192.253.253.1; 192.249.249.9; 192.253.253.9;
};

options {
    directory "/var/named";
    allow-query { "internal"; };
    use-id-pool yes;
};

zone "movie.edu" {
    type master;
    file "db.movie.edu";
    allow-query { any; };
    allow-transfer { "slaves"; };
};

zone "249.249.192.in-addr.arpa" {
    type master;
```

```
        file "db.192.249.249";
        allow-query { any; };
        allow-transfer { "slaves"; };
    };
```

这里，更宽容的区专用（zone-specific）的访问控制列表只适用于对该服务器的权威区的查询，而限制更严格的全局访问列表适用于所有其他的查询。

如果我们正运行 BIND 8.1.2 或更新的版本，使用 *allow-recursion* 子语句我们能够稍微简化该配置。

```
acl "internal" {
    192.249.249/24; 192.253.253/24; 192.253.254/24;
};

acl "slaves" {
    192.249.249.1; 192.253.253.1; 192.249.249.9; 192.253.253.9;
};

options {
    directory "/var/named";
    allow-recursion { "internal"; };
    use-id-pool yes;
};

zone "movie.edu" {
    type master;
    file "db.movie.edu";
    allow-transfer { "slaves"; };
};

zone "249.249.192.in-addr.arpa" {
    type master;
    file "db.192.249.249";
    allow-transfer { "slaves"; };
};
```

我们不再需要 *allow-query* 子语句:尽管名字服务器可能接收来自我们内部网之外的查询，它将那些查询当做非递归来处理，而不管查询是不是非递归查询。结果，来自外部的查询将不会导致我们的名字服务器发送任何到其他名字服务器的查询。该配置也不会遇到前面设置所容易遇到的问题：如果你的名字服务器是父区的权威，它可能接收来自远程名字服务器解析区子域中域名的查询。*allow-query* 解决方法将拒绝那些合法的查询，但是 *allow-recursion* 解决方法却不会。

另外一个选择就是在一台主机上运行两个 *named* 进程。一个被配置为“ 授权 ”服务

器，一个作为“解析”名字服务器。因为我们无法告诉远程服务器或者配置解析器使用除端口 53（默认的 DNS 服务端口）以外的端口来访问名字服务器，我们就必须在不同的 IP 地址上运行这两个名字服务器。

当然，如果你的主机已经有多个网络接口的话，就毫无问题。即使它只有一个，操作系统也可能会支持 IP 地址别名。这些就允许你将一个网络接口对应多个 IP 地址。一个 *named* 进程可以监听一个。最后，如果操作系统不支持 IP 别名，你仍然可以把一个 *named* 进程绑定在网络接口的 IP 地址上，而把一个绑定在回送地址上。只有本地主机可以向监听回送地址的 *named* 进程发送请求，不过，如果本地主机的解析器是你惟一需要服务的解析器的话，这就很好了。

首先，下面是授权名字服务器的 *named.conf* 文件，监听网络接口的 IP 地址：

```
acl "slaves" {
    192.249.249.1; 192.253.253.1; 192.249.249.9; 192;253.253.9; };
};

options {
    directory "/var/named-delegated";
    recursion no;
    fetch-glue no;
    listen-on { 192.249.249.3; };
    pid-file "/var/run/named.delegated.pid";
};

zone "movie.edu" {
    type master;
    file "db.movie.edu";
    allow-transfer { "slaves"; };
};

zone "249.249.192.in-addr.arpa" {
    type master;
    file "db.192.249.249";
    allow-transfer { "slaves"; };
};

zone "." {
    type hint;
    file "db.cache";
};
```

接下来，是解析名字服务器的 *named.conf* 文件，监听回送地址：

```
options {
```

```
        directory "/var/named-resolving";
        listen-on { 127.0.0.1; };
        pid-file "/var/run/named.resolving.pid";
        use-id-pool yes;
    };

    zone "." {
        type hint;
        file "db.cache";
    };
```

注意，对于解析名字服务器，我们不需要访问控制列表，因为它只在回送地址监听，不能接收来自其他主机的查询。（如果我们的解析名字服务器在一个IP别名或另一个网络接口上监听，我们可以使用 *allow-query* 来防止其他人使用我们的名字服务器。）我们在授权服务器上关闭了递归功能，但是在解析名字服务器上的递归功能必须打开。我们也给每个名字服务器自己的PID文件和自己的目录，所以它们不会试图访问相同的默认 PID 文件、调试文件和统计数据文件。

为了使解析名字服务器监听在回送地址上，本地主机的 *resolv.conf* 文件就必须将：

```
    nameserver 127.0.0.1
```

作为第一个 *nameserver* 指令。

如果你正在运行BIND 9，利用视图（view），你甚至能够将两个名字服务器的配置合并成一个：

```
    options {
        directory "/var/named";
    };

    acl "internal" {
        192.249.249/24; 192.253.253/24; 192.253.254/24;
    };

    view "internal" {
        match-clients { "internal"; };
        recursion yes;

        zone "movie.edu" {
            type master;
            file "db.movie.edu";
        };

        zone "249.249.192.in-addr.arpa" {
```

```
            type master;
            file "db.192.249.249";
        };

        zone "." {
            type hint;
            file "db.cache";
        };
    };

    view "external" {
        match-clients { any; };
        recursion no;

        zone "movie.edu" {
            type master;
            file "db.movie.edu";
        };

        zone "249.249.192.in-addr.arpa" {
            type master;
            file "db.192.249.249";
        };

        zone "." {
            type hint;
            file "db.cache";
        };
    };
```

这是一个相当简单的配置：两个视图，一个内部一个外部。只对我们内部网络有用的内部视图存在递归。只对其他网络有用的外部视图不存在递归。区 *movie.edu* 和 *249.249.192.in-addr.arpa* 在两个视图里的定义相同。使用视图，你能够多做一些事（例如，针对内部和外部网络定义区的不同版本），我们将在下一节讲述这部分内容。

DNS 和 Internet 防火墙

域名服务器并非设计用来与 Internet 防火墙协同工作，但是你可以配置 DNS 并使之与 Internet 防火墙协同工作，这恰恰说明了 DNS 及其 BIND 实现的灵活性。

尽管配置 BIND 使之在防火墙环境中工作并非很困难，但是却要对 DNS 和 BIND 的一些较晦涩的特性有透彻的理解。描述 BIND 的配置也将占用本章的较大篇幅，因此这里只做简单介绍。

我们将从描述两大主要的Internet防火墙系列开始，即，包过滤防火墙和应用网关防火墙。将要介绍的每个系列防火墙的功能取决于你需要如何配置BIND使之通过防火墙工作。下一步，我们将详细说明使用防火墙的两个最主要的DNS体系结构，即，转发器和内部根，并且描述各自体系结构的优缺点。然后，我们将介绍一个使用新特性 —— 转发区（forward zone）的解决方案，该新特性融合了转发器和内部根体系结构的优势。最后，我们将讨论分离式名字空间（split namespace）和位于你的防火墙系统核心的堡垒主机的配置。

防火墙软件的类型

在开始配置BIND与防火墙一起工作之前，搞清楚你的防火墙都能干些什么是很重要的。防火墙的功能可能会影响你对DNS体系结构的选择，并决定你如何实现它。如果你不知道本部分所涉及问题的答案，请找知道答案的人并向他请教。最好和防火墙管理员一起设计DNS的结构，从而确保它能与防火墙共存。

请注意，这距离完全解释清楚防火墙还相当遥远，这里的几段文字只能描述防火墙常见的两种类型，以及它们功能的差别是如何影响名字服务器的。对防火墙的全面论述请参见Brent Chapman和Elizabeth D.Zwicky著的《Building Internet Filewalls》。

包过滤

我们首先讲述包过滤防火墙。它主要操作在TCP/IP协议栈的传输层和网络层（即OSI参考模型中的第三层和第四层）。防火墙决定是否依据包级标准（packet-level criteria）路由包，包级标准包括传输协议（也就是说不论是TCP还是UDP）、源和目的IP地址，以及源和目的端口（见图11-1）。

对我们来说，包过滤防火墙最重要的一点是通过对它的典型配置可以有选择地控制Internet上主机与内部网中主机之间的DNS通信。也就是说，你可以让内部网中的任意一组主机与Internet的名字服务器通信。有些包过滤防火墙甚至允许你的名字服务器查询Internet上的名字服务器，而反之则不行。所有基于路由器的Internet防火墙都是包过滤防火墙。Checkpoint的FireWall-1、Cisco的PIX和Sun的SunScreen等都是著名的商业包过滤防火墙。

图 11-1　包过滤器工作在协议栈的网络层和传输层

应用层网关

应用层网关操作在应用协议层，在OSI参考模型中该层比包过滤器所操作的最高层还要高出几层（见图 11-2）。在某种意义上，它采用和特定应用服务器一样的方法理解应用协议。例如，FTP 应用层网关决定是允许还是拒绝一个特定的 FTP 操作，比如 RETR（得到一个文件）或者 STOR（发送一个文件）。

图 11-2　应用层网关工作在协议栈的应用层

BIND 8 或 9 与包过滤防火墙

BIND 4名字服务器总是从53端口（该端口为DNS服务器的默认服务端口）发送查询到53端口。相反，解析器通常从大数值（大于1023）的端口发送查询到53端口。尽管名字服务器毫无疑问地要向远程主机的DNS端口发送它们的查询，但是它们并非一定要从DNS端口来发送这些查询。并且，也许你不知道，BIND 8和9名字服务器默认时并不从53端口发送查询。相反，与解析器类似，它们从大数值端口发送查询。

当包过滤防火墙配置成允许名字服务器到名字服务器的通信，而不允许解析器到名字服务器的通信时，这将产生问题，原因是包过滤防火墙一般认为名字服务器到名字服务器的通信都是以53端口作为源端口和目的端口的。

针对该问题存在两种解决方法：

- 重新配置防火墙，允许你的名字服务器从不同于53端口的其他端口来发送和接收查询（假设Internet上的主机发送到内部名字服务器大数值端口上的包并不危及防火墙的安全）。
- 配置BIND，通过*query-source*子语句恢复其原来的行为。

*query-source*使用两个参数，一个是地址，一个是可选的端口号。例如，语句

```
options { query-source address * port 53; };
```

告诉BIND使用53端口作为从所有本地网络接口发送查询的源端口。可以使用无通配符的地址声明来限制BIND发送查询的源地址。例如，在*wormhole.movie.edu*上，语句：

```
options { query-source address 192.249.249.1 port *; };
```

告诉BIND以192.249.249.1为源地址发送所有查询（例如，不是从192.253.253.1）并且使用动态的大数值端口。

尽管可以告诉早期的BIND 9名字服务器从一个特定地址的53端口发送所有查询，但是使用含通配符的*query-source*语句已在9.1.0版本之前的BIND 9中取消了。

坏消息就是大多数应用层网关都仅能处理基于TCP的应用协议，这对我们的目标是很重要的。当然，绝大多数DNS都是基于UDP的，而我们知道没有为DNS设计的应用层网关防火墙。这意味着，如果运行了应用层网关防火墙，内部主机就很可能不能直接与Internet上的名字服务器进行通信。

来自TIS（Trusted Information Systems，现在属于Network公司）的著名防火墙工具包是一套适用于Telnet、FTP、HTTP这类通用Internet协议的应用层网关。Network公司的Gauntlet产品就是基于应用层网关的，Axent的Eagle防火墙也是如此。

注意，这两类防火墙已经通用化了。防火墙的最新技术变化很快，当读到这里的时候，你也许已经拥有了一个包括为DNS而设计的应用层网关的防火墙。防火墙属于哪一类主要取决于它能做些什么，更重要的是你的独特防火墙是否允许任意内部主机与Internet之间进行DNS通信。

一个不好的实例

最简单的配置就是允许DNS自由地通过防火墙通信（假定你可以这样配置防火墙）。这样，任何内部名字服务器都可以查询任意一个Internet上的名字服务器，而且任何Internet上的名字服务器也都可以查询任意一个内部的名字服务器。不需要任何特殊的配置。

不幸的是，有大量的理由可以证明这是个很糟糕的主意。

版本控制

BIND的开发者们不断地发现和修补BIND代码中的安全漏洞。因此，运行最新发布的BIND版本是很重要的，对直接暴露于Internet上的名字服务器尤其如此。如果一台或一些内部名字服务器可直接与Internet上的名字服务器通信，版本升级是很容易的事。如果任意一台内部名字服务器都可以直接与Internet上的名字服务器通信，那么这些服务器的升级将困难得多。

可能遭受攻击

即使你没有在一台特定的主机上运行名字服务器，黑客也能利用你允许DNS数据穿过防火墙来攻击该主机。例如，在内部工作的同谋者可以启动一个

Telnet守护进程来监听主机的DNS端口，从而允许黑客直接远程登录到该主机上。

在后面，我们尝试举一个好的例子。

Internet 转发器

由于允许双向的DNS数据不加限制地穿过防火墙会带来危险，大多数组织都限制与Internet 上的名字服务器交流的内部主机。在应用层网关防火墙，或其他任何没有DNS 数据通过能力的防火墙中，能和 Internet 上的名字服务器通信的惟一主机是堡垒主机（见图 11-3）

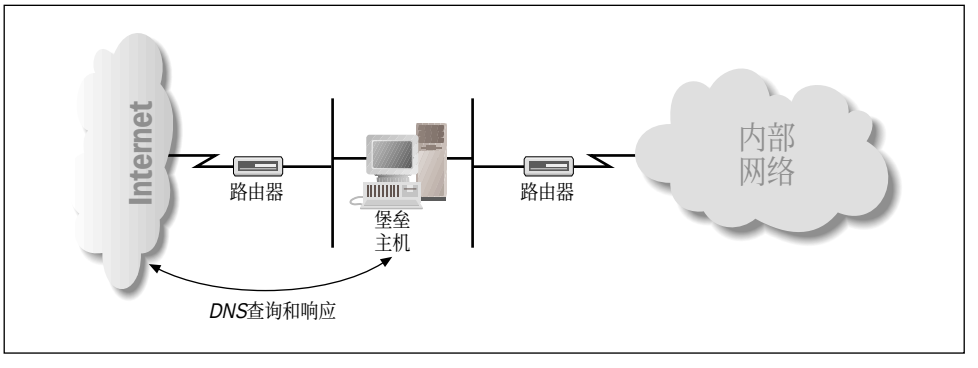


图 11-3　展示堡垒主机的小型网络图

在包过滤防火墙中，防火墙管理员可以配置防火墙使任何一组内部名字服务器与Internet名字服务器通信。通常，这是在网络管理员直接控制下运行名字服务器的一小组主机（见图 11-4）。

内部名字服务器不需要任何特殊的配置就可以直接查询 Internet 上的名字服务器。它们的根线索文件包含 Internet 的根名字服务器，从而允许它们解析 Internet 域名。不能查询 Internet 上名字服务器的内部名字服务器必须知道如何转发（forward）自己不能解析的查询到能解析该查询的名字服务器上。这需要用到第十章介绍的 *forwarders* 指令或子语句。

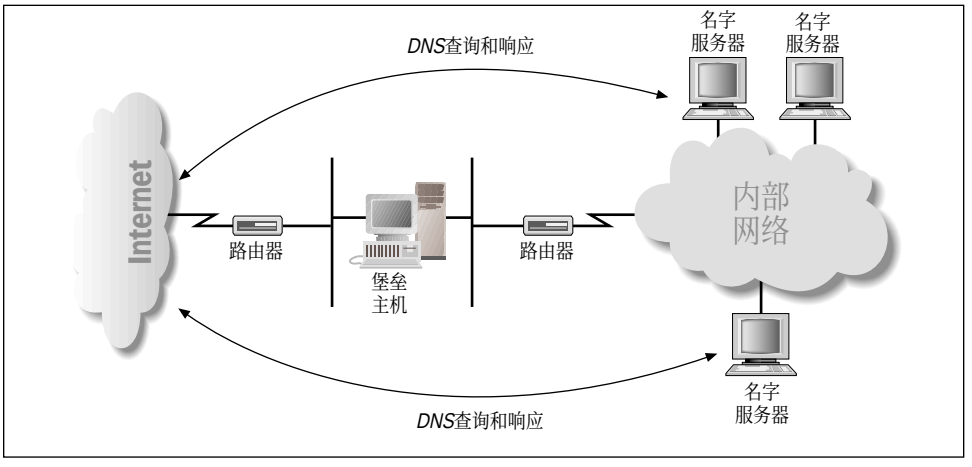


图 11-4　展示选定的内部名字服务器的小型网络图

图11-5演示了通用的转发设置, 内部名字服务器转发查询到运行在堡垒主机上的名字服务器。

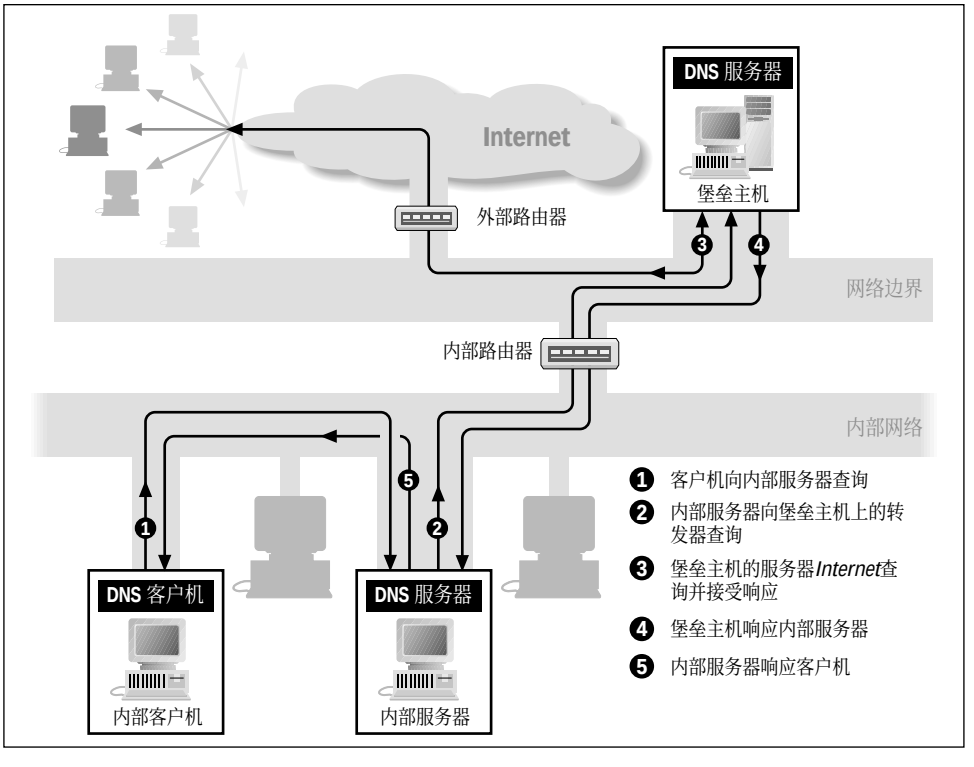


图 11-5　利用转发器

在电影大学，几年前，针对庞大且环境恶劣的Internet，我们使用防火墙来保护自己。我们用的是包过滤防火墙，而且我们与防火墙管理员协商，允许在Internet名字服务器和两台内部名字服务器*terminator.movie.edu*和*wormhole.movie.edu*之间进行DNS通信。下面是我们对该大学其他内部名字服务器的配置。对于BIND 8和9名字服务器，我们使用下面的语句：

```
options {
    forwarders { 192.249.249.1; 192.249.249.3; };
    forward only;
};
```

对于BIND 4名字服务器，我们使用：

```
forwarders 192.249.249.3 192.249.249.1
options forward-only
```

我们改变转发器出现的顺序以便分散它们的负荷，但是对于BIND 8.2.3名字服务器，并不需要那样，因为BIND 8.2.3是根据往返时间选择转发器进行查询的。

当内部名字服务器收到一个本机无法解析的域名查询，比如一个Internet域名的时候，就将该查询转发到可以利用Internet名字服务器解析该域名的转发器中的一个。真是简单！

转发带来的麻烦

不幸的是，这有点太简单了。一旦授权子域或是构建一个庞大的网络，转发就开始引起麻烦了。为了解释清楚，请看*zardoz.movie.edu*配置文件的如下部分：

```
options {
    directory "/var/named";
    forwarders { 192.249.249.1; 192.253.253.3; };
};

zone "movie.edu" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.movie.edu";
};
```

*zardoz.movie.edu*是*movie.edu*的辅名字服务器，并且使用我们的两个转发器。当*zardoz.movie.edu*收到对*fx.movie.edu*域内的一个名字查询时会发生什么事呢？作为

movie.edu 的授权名字服务器，*zardoz.movie.edu* 有 NS 记录，该 NS 记录授权 *fx.movie.edu*作为它的权威名字服务器。而同时它也被配置为将本地不能解析的查询转发给 *terminator.movie.edu* 和 *wormhole.movie.edu*。那么它将执行哪一项呢？

意想不到的事发生了，*zardoz.movie.edu* 将忽略授权信息而把查询转发给 *terminator.movie.edu*。之后，*terminator.movie.edu* 接收递归查询，并且代表 *zardoz.movie.edu* 向 *fx.movie.edu* 的名字服务器查询。但是这样的查询效率不高，因为 *zardoz.movie.edu* 可以容易地将查询直接发给 *fx.movie.edu* 名字服务器。

现在设想网络的规模非常之大：一个公司网络，它跨越好几个大陆，拥有几万台主机和成百上千台名字服务器。所有未直接与 Internet 连通的内部名字服务器（绝大多数内部名字服务器都是这种情况）使用一小组转发器。在这种局面下会发生什么问题呢？

单一故障点

如果转发器故障，你的名字服务器就不能解析那些没有在缓存中、又不在权威数据中的 Internet 域名和内部网域名。

负荷集中

转发器将承担巨大的查询负荷。原因有两个，一是有大量的内部名字服务器使用它们；二是查询是递归的，且要花费大量的精力去回答。

低效率的解析

设想有两个内部名字服务器，分别是 *west.acmebw.com* 和 *east.acmebw.com* 的权威，两者在美国科罗拉多州 Boulder 的同一个网段，并被配置为使用在马里兰州 Bethesda的本公司的同一台转发器。*west.acmebw.com* 的名字服务器为了解析 *east.acmebw.com* 域里的一个域名，它要向位于 Bethesda 的转发器发送一个查询。然后该转发器发回一个查询到 Boulder，再到 *east.acmebw.com* 的名字服务器 —— 始发查询者的邻居。作为答复，*east.acmebw.com* 的名字服务器发回一个响应到 Bethesda，然后再由转发器发回 Boulder。

根据对根名字服务器的传统配置，*west.acmebw.com* 名字服务器很快就会知道有一个 *east.acmebw.com* 名字服务器就在隔壁并且会利用它（因为它的来回时间短）。利用转发器的“短路”功能，正常效率的解析得以进行。

于是我们得出结论，对于小型网络和简单域名空间，转发效果是好的，而对于大型网络和复杂域名空间，转发可能是不合适的。在电影大学中，随着网络的增大，我们发现转发是困难的，这迫使我们采用其他方法。

使用转发区

我们可以通过使用 BIND 8.2 中引入的转发区（forward zone）解决这个问题。我们将 *zardoz.movie.edu* 的配置改为下面所示的配置：

```
options {
    directory "/var/named";
    forwarders { 192.249.249.1; 192.253.253.3; };
};

zone "movie.edu" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.movie.edu";
    forwarders {};
};
```

现在，如果 *zardoz.movie.edu* 接收一个针对以 *movie.edu* 结尾，但不在 *movie.edu* 区的域名查询（例如，在 *fx.movie.edu*），它将忽略转发器而发送重复查询。

使用这个配置，*zardoz.movie.edu*仍旧向我们的转发器发送我们反向映射区中的域名查询。为减少转发器的负载，我们增加一些 *zone* 语句到 *named.conf*:

```
zone "249.249.192.in-addr.arpa" {
    type stub;
    masters { 192.249.249.3; };
    file "stub.192.249.249";
    forwarders {};
};
zone "253.253.192.in-addr.arpa" {
     type stub;
     masters { 192.249.249.3; };
     file "stub.192.253.253";
     forwarders {};
};
zone "254.253.192.in-addr.arpa" {
     type stub;
     masters { 192.253.254.2; };
     file "stub.192.253.254";
     forwarders {};
};
```

```
zone "20.254.192.in-addr.arpa" {
     type stub;
     masters { 192.253.254.2; };
     file "stub.192.254.20";
     forwarders {};
};
```

这些新的*zone*语句负责解释：首先，它们配置电影大学的反向映射区为存根。这使得我们的名字服务器通过周期性地查询那些区的主名字服务器来跟踪那些区的NS记录。然后，*fowarders*子语句关闭针对反向映射域中域名的转发。比如，对于*2.254.253.192.in-addr.arpa*的PTR记录，*zardoz.movie.edu*将直接查询*254.253.192.in-addr.arpa*名字服务器，而非查询转发器。

在我们所有的内部名字服务器上都需要有类似上述的*zone*语句，这也暗示我们所有的名字服务器都需要运行BIND 8.2之后的某些版本（注4）。

这给我们一个相当健壮的解析体系结构，该体系结构使我们在Internet上的暴露最小化：它使用有效、健壮的重复名字解析来解析内部域名，并且只有当需要解析Internet域名时才使用转发器。如果我们的转发器失效或丢失了到Internet的连接，我们也仅仅只是丧失了解析Internet域名的能力。

内部根

如果想避免转发引起的扩展性（scalability）问题，可以设立自己的根名字服务器。内部根仅仅服务于组织内部的名字服务器，它们只知道与组织相关的名字空间。

这有什么好处呢？通过采用基于根名字服务器的结构，可以获得Internet域名空间的扩展性（这对大多数公司将是足够好的）、增加的冗余、分布的负荷和高效的解析。你可以有与Internet的根一样多的内部根（十三个左右），反之，有这么多的转发器可能会降低安全性和增加配置负担。大多数情况下，内部根不会轻易地使用。名字服务器只有在顶级区的NS记录超时之后才会查询内部根。若使用转发器，名字服务器可能每解析一次名字就要查询一次转发器。

我们的教训就是：如果你有或者打算有巨大的域名空间和众多的内部名字服务器，则内部根名字服务器将比任何其他方案都要好。

注4： 正如我们在上一章提到的那样，BIND 9直到BIND 9.1.0才支持转发区。

在哪里放置内部根名字服务器

因为名字服务器利用最短的往返时间“锁定”最邻近的根名字服务器，所以要将根名字服务器散布在网络中。如果你组织的网络跨越美国、欧洲和太平洋沿岸地区，则要考虑在每一个大陆至少放置一台根名字服务器。如果在欧洲有三个主要的站点，则要为每个站点分配一个内部根。

转发映射授权

这里将介绍如何配置一台内部根名字服务器。内部根直接授权你管理的任何区。例如，在 *movie.edu* 网络上，根区的数据文件将包括如下内容：

```
movie.edu.  86400  IN  NS  terminator.movie.edu.
            86400  IN  NS  wormhole.movie.edu.
            86400  IN  NS  zardoz.movie.edu.
terminator.movie.edu.  86400  IN  A  192.249.249.3
wormhole.movie.edu.    86400  IN  A  192.249.249.1
                       86400  IN  A  192.253.253.1
zardoz.movie.edu.      86400  IN  A  192.249.249.9
                       86400  IN  A  192.253.253.9
```

在 Internet 上，这些信息会在 *edu* 名字服务器的区数据文件中出现。当然，在 *movie.edu* 网络中，没有任何 *edu* 名字服务器，因此直接从根授权到 *movie.edu*。

注意，这不包括对 *fx.movie.edu* 或任何其他 *movie.edu* 的子域的授权。*movie.edu* 名字服务器知道哪些名字服务器是哪个 *movie.edu* 子域的权威，所有关于这些子域信息的查询都将通过 *movie.edu* 名字服务器，因此，没有必要在这里对它们进行授权。

in-addr.arpa 授权

我们还要从内部根授权给 *movie.edu* 相应网络的 *in-addr.arpa* 区。

```
249.249.192.in-addr.arpa.  86400  IN  NS  terminator.movie.edu.
                           86400  IN  NS  wormhole.movie.edu.
                           86400  IN  NS  zardoz.movie.edu.
253.253.192.in-addr.arpa.  86400  IN  NS  terminator.movie.edu.
                           86400  IN  NS  wormhole.movie.edu.
                           86400  IN  NS  zardoz.movie.edu.
254.253.192.in-addr.arpa.  86400  IN  NS  bladerunner.fx.movie.edu.
                           86400  IN  NS  outland.fx.movie.edu.
                           86400  IN  NS  alien.fx.movie.edu.
20.254.192.in-addr.arpa.   86400  IN  NS  bladerunner.fx.movie.edu.
```

```
                        86400  IN  NS  outland.fx.movie.edu.
                        86400  IN  NS  alien.fx.movie.edu.
```

注意，我们包括了对*254.253.192.in-addr.arpa*区和*20.254.192.in-addr.arpa*区的授权，尽管它们对应于*fx.movie.edu*区。我们不需要授权*fx.movie.edu*，因为我们已经授权了它的父区*movie.edu*。*movie.edu*的名字服务器授权了*fx.movie.edu*，所以根据传递性，根也授权了*fx.movie.edu*。因为其他两个*in-addr.arpa*区都不是*254.253.192.in-addr.arpa*或*20.254.192.in-addr.arpa*的父区，所以我们要从根授权给它们。如前所述，我们不需要为*bladerunner.fx.movie.edu*、*outland.fx.movie.edu*和*alien.fx.movie.edu*这三个特效（特殊效应）名字服务器添加地址记录，因为远程名字服务器可以通过*movie.edu*的授权找到它们的地址。

db.root文件

剩下的工作就是为根区添加一个SOA记录，并为这个内部根名字服务器和任何其他名字服务器添加NS记录：

```
$TTL 1d
.  IN  SOA  rainman.movie.edu.  hostmaster.movie.edu.  (
            1    ; 序列号
            3h   ; 刷新
            1h   ; 重试
            1w   ; 期满
            1h ) ; 否定缓存TTL

   IN  NS  rainman.movie.edu.
   IN  NS  awakenings.movie.edu.

rainman.movie.edu.    IN  A  192.249.249.254
awakenings.movie.edu. IN  A  192.253.253.254
```

*rainman.movie.edu*和*awakenings.movie.edu*是运行内部根名字服务器的主机。我们不应该在堡垒主机上运行内部根名字服务器，因为如果Internet上的名字服务器偶然向它查询的数据不是它的权威数据，内部根将响应它的根列表 —— 都是内部的。

整个*db.root*文件（按照惯例，我们称根区的数据文件为*db.root*）如下所示：

```
$TTL 1d
.  IN  SOA  rainman.movie.edu.  hostmaster.movie.edu.  (
            1    ; 序列号
            3h   ; 刷新
            1h   ; 重试
```

```
                1w   ; 期满
                1h ) ; 否定缓存 TTL

        IN  NS  rainman.movie.edu.
        IN  NS  awakenings.movie.edu.

    rainman.movie.edu.    IN  A  192.249.249.254
    awakenings.movie.edu. IN  A  192.253.253.254

    movie.edu.  IN  NS  terminator.movie.edu.
                IN  NS  wormhole.movie.edu.
                IN  NS  zardoz.movie.edu.

    terminator.movie.edu.  IN  A  192.249.249.3
    wormhole.movie.edu.    IN  A  192.249.249.1
                           IN  A  192.253.253.1
    zardoz.movie.edu.      IN  A  192.249.249.9
                           IN  A  192.253.253.9

    249.249.192.in-addr.arpa.   IN  NS  terminator.movie.edu.
                                IN  NS  wormhole.movie.edu.
                                IN  NS  zardoz.movie.edu.
    253.253.192.in-addr.arpa.   IN  NS  terminator.movie.edu.
                                IN  NS  wormhole.movie.edu.
                                IN  NS  zardoz.movie.edu.
    254.253.192.in-addr.arpa.   IN  NS  bladerunner.fx.movie.edu.
                                IN  NS  outland.fx.movie.edu.
                                IN  NS  alien.fx.movie.edu.
    20.254.192.in-addr.arpa.    IN  NS  bladerunner.fx.movie.edu.
                                IN  NS  outland.fx.movie.edu.
                                IN  NS  alien.fx.movie.edu.
```

rainman.movie.edu 和 *awakenings.movie.edu* 两个内部根名字服务器上的 *named.conf* 文件包括下面几行：

```
zone "." {
    type master;
    file "db.root";
};
```

对于 BIND 4 服务器的 *named.boot* 文件则包括下面一行：

```
    primary    .    db.root
```

这替代了*hint*类型或*cache*指令的*zone*语句，根名字服务器不需要根线索（root hint）文件告诉它其他的根在哪里；它可以在*db.root*中找到。我们真的是说每个根名字服务器都是根区的主名字服务器吗？事实上，这要取决于你运行的BIND版本。4.9以后的BIND版本要你为根区声明一个辅名字服务器，但是4.8.3及比它早的BIND版本要求所有根名字服务器都要以主名字服务器加载根区。

如果你手头没有很多闲置的主机来作为内部根，请不要失望！任何内部名字服务器（就是指不是运行在堡垒主机或者防火墙之外的名字服务器）都可以既作为内部根，同时又是任何其他你需要加载的区的权威名字服务器。请记住，一台名字服务器可以是很多很多域的权威，包括根。

配置其他内部名字服务器

一旦建立了内部名字服务器，就要配置所有内部网主机上的名字服务器去使用它们。任何运行在没有直接连接到Internet的主机上的名字服务器（比如，在防火墙之后）都要在根线索文件中列出内部根：

```
; 内部根线索文件，用于电影大学没有直接连接到 Internet 上的主机
;
; 不要将本文件在连接到 Internet 的主机上使用！
;

.  99999999  IN  NS  rainman.movie.edu.
   99999999  IN  NS  awakenings.movie.edu.

rainman.movie.edu.     99999999  IN  A  192.249.249.254
awakenings.movie.edu.  99999999  IN  A  192.253.253.254
```

运行在使用这个根线索文件的主机上的名字服务器将能够解析 *movie.edu* 域和电影大学的 *in-addr.arpa* 域中的域名，但是不能解析这两个域之外的域名。

内部名字服务器如何使用内部根

为了将整个方案的工作联系在一起，让我们仔细体会那个在一台使用这些内部根名字服务器的只缓存（caching-only）名字服务器上的域名解析的例子。首先，内部名字服务器收到一个对 *movie.edu* 内的一个域名的查询，比如说要查询的是 *gump.fx.movie.edu* 的地址。如果内部名字服务器没有缓存任何“更好的”信息，它就先查询内部根名字服务器。如果在此之前它与内部根通信过，那么它会用与每个内部根相关的往返时间，告诉它哪个内部根响应最快，它就向该内部根发送一个查找 *gump.fx.movie.edu* 地址的非递归查询。内部根的回答将指向 *terminator.movie.edu* 上、*wormhole.movie.edu* 上和 *zardoz.movie.edu* 上的 *movie.edu* 名字服务器。随后，只缓存内部名字服务器向这些 *movie.edu* 名字服务器中的一个发送另一个非递归的查询，以查找 *gump.fx.movie.edu* 的地址。该 *movie.edu* 名字服务器的响应将其指向

*fx.movie.edu*名字服务器。只缓存名字服务器再向这些*fx.movie.edu*名字服务器中的一个发送相同的查找*gump.fx.movie.edu*的地址的非递归查询，最后收到一个响应。

我们把它与转发结构的工作方法做个比较。设想只缓存名字服务器被配置成首先转发查询到*terminator.movie.edu*，然后到*wormhole.movie.edu*，而不是使用内部根名字服务器。这种情况下，只缓存名字服务器在缓存里没有找到*gump.fx.movie.edu*的地址，就会转发查询到*terminator.movie.edu*。然后，*terminator.movie.edu*代表只缓存名字服务器查询一个*fx.movie.edu*名字服务器，并返回回答。如果只缓存名字服务器还需要查找*fx.movie.edu*内的另一个域名，那么它还要询问转发器，即使转发器在有关查找*gump.fx.movie.edu*的查询的响应中已经包括了*fx.movie.edu*名字服务器的名字和地址。

从内部主机到 Internet 的邮件

等一下，这些并不是内部根为你做的全部。我们下面将讨论在整个网络范围内不改变*sendmail*的配置而将邮件发送到Internet上。

通配符记录是使邮件系统得以工作的关键，准确地说，是通配符MX记录。比方说，我们希望发送到Internet的邮件经由*postmanrings2x.movie.edu*（电影大学的堡垒主机，它直接连通到Internet上）转发。那么添加这些记录到*db.root*：

```
*        IN    MX    5 postmanrings2x.movie.edu.
*.edu.   IN    MX    10 postmanrings2x.movie.edu.
```

就可以完成这项工作。除了*记录还需要*.edu*记录，因为要遵循DNS通配符产生规则，关于这部分信息请参看第十六章。基本上，因为在区中有*movie.edu*的显式数据，所以第一个通配符记录不会匹配*movie.edu*以及任何其他*edu*的子域。于是我们需要另一个对应*edu*的更显式的通配符记录来匹配除*movie.edu*之外的*edu*的所有子域。

现在我们内部的*movie.edu*主机上的邮件收发器（mailer）可以将要发送给Internet域的邮件发往*postmanrings2x.movie.edu*并由其转发。例如，发往*nic.ddn.mil*的邮件将匹配第一个通配符MX记录：

```
%  nslookup -type=mx nic.ddn.mil.  一用MX记录来匹配*
Server:  rainman.movie.edu
```

```
Address:  192.249.249.19

nic.ddn.mil
     preference = 5, mail exchanger = postmanrings2x.movie.edu
postmanrings2x.movie.edu    internet address = 192.249.249.20
```

发往 *vangogh.cs.berkeley.edu* 的邮件将匹配第二个通配符 MX 记录：

```
%  nslookup -type=mx vangogh.cs.berkeley.edu.  - 用 MX 记录来匹配 *.edu
Server:  rainman.movie.edu
Address:  192.249.249.19

vangogh.cs.berkeley.edu
     preference = 10, mail exchanger = postmanrings2x.movie.edu
postmanrings2x.movie.edu    internet address = 192.249.249.20
```

一旦邮件到达了 *postmanrings2x.movie.edu*（我们的堡垒主机），*postmanrings2x.movie.edu*的邮件收发器将自己查找关于这些地址的MX记录。因为*postmanrings2x.movie.edu* 将用 Internet 的域名空间而不是内部域名空间来解析 Internet 域名，所以可以找到对应目的域的真正 MX 记录，并递送邮件。不改变 *sendmail* 的配置是必然的。

发送邮件到指定的 Internet 域

内部根方案还有另一个额外的好处：如果你有多于一个的堡垒主机，可以通过特定的堡垒主机转发发往某个 Internet 域的邮件。例如，我们可以选择先将所有发给 *uk* 域收件人的邮件全部发往我们在伦敦的堡垒主机，然后再发送到 Internet 上。如果内部网的连通性或可靠性比 Internet 的英国部分好的话，这将是很有用的。

电影大学有一个私有网络连接到我们在伦敦 Pinewood 工作室（Pinewood Studios）附近的姊妹大学。出于安全性原因，我们更喜欢通过私有连接发送邮件到英国的堡垒主机，然后到达 Pinewood 的主机。因此我们添加如下通配符记录到 *db.root* 文件中：

```
; holygrail.movie.ac.uk 是在 U.K. Internet 连接的另一端
*.uk.    IN    MX    10 holygrail.movie.ac.uk.
holygrail.movie.ac.uk.    IN   A    192.168.76.4
```

现在，发给 *uk* 子域的用户的邮件将被转发到我们姊妹大学的主机 *holygrail.movie.ac.uk*，该主机将把邮件转发到英国的其他地方。

内部根带来的问题

不幸的是，就像转发有它的问题一样，内部根也有它的局限性。主要一点就是内部主机看不到 Internet 域名空间。在有些网络上这不是问题，因为绝大多数内部主机没有任何 Internet 连通性，少部分内部主机配置它们的解析器使用堡垒主机上的名字服务器。这些少部分主机中的一些将可能需要运行代理服务器以允许其他内部主机访问 Internet 上的服务。

不过对于其他网络，Internet防火墙或其他软件可能要求所有的内部主机都要有解析 Internet 域名空间里的域名的能力。对这些网络而言，内部根结构将不能工作。

分离式名字空间

许多组织更愿意使 Internet 上公布的区数据与内部公布的区数据有所不同。在多数情况下，因为有 Internet 防火墙，很多内部区数据都和 Internet 无关。防火墙可以不允许直接访问多数内部主机，并且将内部没有注册的IP地址转换为该组织注册了的 IP 地址。因此，该组织可能需要剔除一些对外无关的消息，或者将内部地址转变为对应的外部的地址。

不幸的是，BIND 不支持区域数据的自动过滤和转换。因此，许多机构手工生成众所周知的“ 分离式名字空间 ”（split namespace）。在分离式名字空间中，真实的名字空间只有内部才可以使用。它的删减转化版本，被称为“ 影子名字空间 ”（shadow namespace），是 Internet 可以看见的。

影子名字空间包含一些主机名到地址和地址到主机名的映射，其中这些主机是Internet通过防火墙可以访问的。被公布的地址可以是真实内部地址转化后的对外地址。影子名字空间也可以包含一个或多个 MX 记录，引导邮件从 Internet 通过防火墙到达内部的邮件服务器。

因为电影大学的防火墙大大限制了从 Internet 到内部网络的访问，所以我们选择创建一个影子名字空间。对于区 *movie.edu*，我们需要对外公布的信息仅仅包括域名 *movie.edu*（一个 SOA 和几个 NS 记录）、堡垒主机（*postmanrings2x. movie.edu*）、新的兼做外部 Web 服务器（*www.movie.edu*）的外部名字服务器（*ns.movie.edu*）。堡垒主机上的外部接口的地址是 200.1.4.2，名字 /Web 服务器的地址是 200.1.4.3。影子 *movie.edu* 的区数据文件如下：

```
$TTL 1d
@    IN    SOA    ns.movie.edu.    hostmaster.movie.edu. (
                          1    ; 序列号
                          3h   ; 刷新
                          1h   ; 重试
                          1w   ; 期满
                          1h ) ; 否定缓存 TTL

    IN    NS    ns.movie.edu.
    IN    NS    ns1.isp.net. ; 我们 ISP 的名字服务器是 movie.edu 的辅名字服务器

    IN    A     200.1.4.3
    IN    MX    10 postmanrings2x.movie.edu.
    IN    MX    100 mail.isp.net.

www             IN    CNAME movie.edu.

postmanrings2x  IN    A     200.1.4.2
                IN    MX    10 postmanrings2x.movie.edu.
                IN    MX    100 mail.isp.net.

;postmanrings2x.movie.edu handles mail addressed to ns.movie.edu
ns              IN    A     200.1.4.3
                IN    MX    10 postmanrings2x.movie.edu.
                IN    MX    100 mail.isp.net.
*               IN    MX    10 postmanrings2x.movie.edu.
                IN    MX    100 mail.isp.net.
```

注意，这里没有提及*movie.edu*的任何子域，以及对这些子域的名字服务器的授权。这些信息是根本不需要的，因为从Internet上不能访问或获取这些子域的任何信息，而且从外部进入这些子域的主机的邮件可以与通配符匹配。

我们反向映射电影大学的两个可以从 Internet 上看到的 IP 地址，所需的*db.200.1.4*文件看起来如下所示：

```
$TTL 1d
@    IN    SOA    ns.movie.edu.    hostmaster.movie.edu. (
                          1    ; 序列号
                          3h   ; 刷新
                          1h   ; 重试
                          1w   ; 期满
                          1h ) ; 否定缓存 TTL

    IN    NS    ns.movie.edu.
    IN    NS    ns.isp.net.

2    IN    PTR    postmanrings2x.movie.edu.
3    IN    PTR    ns1.movie.edu.
```

我们注意一定要确保的一件事是不能配置堡垒主机上的解析器使用在*ns.movie.edu*主机上的名字服务器。因为该服务器看不到真正的内部*movie.edu*，使用它会导致*postmanrings2x.movie.edu*不能将内部域名映射为地址或内部地址映射为域名。

配置堡垒主机

在分离式名字空间的配置中，堡垒主机是一个特殊情况。它在内外两个环境中都有一个网络接口：一个接口将其连接到Internet，一个接口将其连接到内部网络。现在我们已经将名字空间分离成两个，堡垒主机怎样才能看到Internet名字空间和我们真正的内部名字空间呢？如果我们在堡垒主机的根线索（root hint）文件中用Internet根名字服务器对其进行配置，它将沿着授权关系，从Internet的*edu*名字服务器找到一个拥有影子区数据的外部*movie.edu*名字服务器。这样堡垒主机将看不到内部名字空间，而该名字空间对于堡垒主机看见连接日志、送入站邮件等等是非常重要的。另一方面，如果用内部根对堡垒主机进行配置，那么它将看不到Internet的名字空间，而很显然，堡垒主机看见Internet的名字空间是必需的。该怎么办？

如果我们有内部名字服务器既能解析内部域名又能解析Internet域名（使用本章前面介绍的转发区技术），那么我们可以简单地配置堡垒主机的解析器去查询这些名字服务器。如果依据所运行防火墙的类型而使用内部转发，我们也依然需要在堡垒主机上运行一个转发器名字服务器。如果防火墙不允许DNS数据通过，我们至少需要在堡垒主机上运行一台用Internet根来配置的只缓存（caching-only）名字服务器，以便内部名字服务器有地方转发本地不能解析的查询。

如果我们内部的名字服务器不支持转发区（forward zone），堡垒主机上的名字服务器为了解析地址必须配置成*movie.edu*和任何*in-addr.arpa*区的辅名字服务器。在这种情况下，如果收到了一个对*movie.edu*内域名的查询，它就用本地的权威数据解析该域名。（如果我们的内部名字服务器支持转发区并且正确配置，我们堡垒主机上的名字服务器将不会接收到对*movie.edu*内域名的查询。）如果被查询的域名属于某个*movie.edu*的授权子域，它将沿着区数据中的NS记录去查询内部名字服务器。因此，它不需要被配置成任何*movie.edu*域的子域（比如，*fx.movie.edu*）的辅名字服务器，仅仅需要对“顶部”区进行配置（见图11-6）。

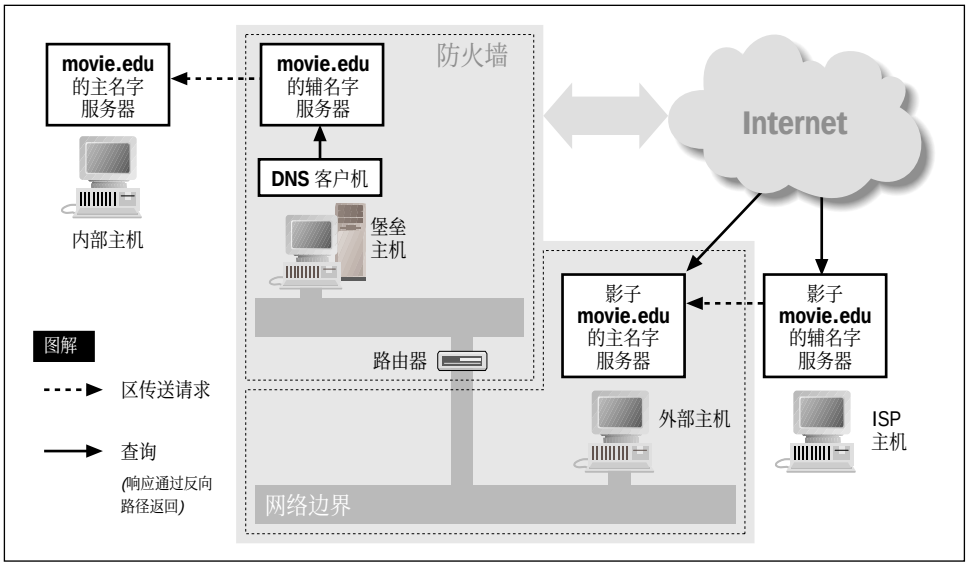


图 11-6　一个分离式 DNS 方案

堡垒主机上的 *named.conf* 文件如下所示：

```
options {
    directory "/var/named";
};

zone "movie.edu" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.movie.edu";
};

zone "249.249.192.in-addr.arpa" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.192.249.249";
};

zone "253.253.192.in-addr.arpa" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.192.253.253";
};

zone "254.253.192.in-addr.arpa" {
    type slave;
    masters { 192.253.254.2; };
```

```
    file "bak.192.253.254";
};

zone "20.254.192.in-addr.arpa" {
    type slave;
    masters { 192.253.254.2; };
    file "bak.192.254.20";
};

zone "." {
    type hint;
    file "db.cache";
};
```

一个等价的 *named.boot* 文件如下所示：

```
directory    /var/named
secondary    movie.edu    192.249.249.3   bak.movie.edu
secondary    249.249.192.in-addr.arpa    192.249.249.3    bak.192.249.249
secondary    253.253.192.in-addr.arpa    192.249.249.3    bak.192.253.253
secondary    254.253.192.in-addr.arpa    192.253.254.2    bak.192.253.254
secondary    20.254.192.in-addr.arpa     192.253.254.2    bak.192.254.20
cache    .    db.cache    ; 列出Internet的根
```

保护堡垒主机上的区数据

不幸的是，在堡垒主机上装载这些区数据的同时，也可能使它们暴露在Internet上，而这正是我们采用分离式名字空间所试图避免的。不过，只要运行了 BIND 4.9 或更好的版本，我们就能够用本章讨论的 *secure_zone* TXT 记录或者 *allow-query* 子句保护这些区数据。利用 *allow-query*，我们可以为区数据设置一个全局访问列表。下面是来自 *named.conf* 文件的新的 *options* 语句：

```
options {
    directory "/var/named";
    allow-query { 127/8; 192.249.249/24; 192.253.253/24;
        192.253.254/24; 192.254.20/24; };
};
```

使用 BIND 4.9 的 *secure_zone*，我们可以通过在每个 *db* 文件里包含这些 TXT 记录来关闭所有对区数据的外部访问：

```
secure_zone     IN    TXT    "192.249.249.0:255.255.255.0"
                IN    TXT    "192.253.253.0:255.255.255.0"
                IN    TXT    "192.253.254.0:255.255.255.0"
                IN    TXT    "192.254.20.0:255.255.255.0"
                IN    TXT    "127.0.0.1:H"
```

不要忘了在这个列表里包括回送地址，否则堡垒主机自己的解析器也不能从名字服务器上得到答案！

最后的配置

最后，我们需要对堡垒主机上的名字服务器施加前面讨论过的其他安全措施。特别地，我们应当：

- 限制区传送
- 使用 ID pool 特性（BIND 8.2 或更新的名字服务器支持，但不包括 BIND 9）
- （可选地）以最小权限运行 BIND *chrooted*

最后，我们的 *named.conf* 文件如下所示：

```
acl "internal" {
    127/8; 192.249.249/24; 192.253.253/24;
    192.253.254/24; 192.254.20/24;
};

options {
    directory "/var/named";
    allow-query { "internal"; };
    allow-transfer { none; };
    use-id-pool yes;
};

zone "movie.edu" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.movie.edu";
};

zone "249.249.192.in-addr.arpa" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.192.249.249";
};

zone "253.253.192.in-addr.arpa" {
    type slave;
    masters { 192.249.249.3; };
    file "bak.192.253.253";
};

zone "254.253.192.in-addr.arpa" {
```

```
    type slave;
    masters { 192.253.254.2; };
    file "bak.192.253.254";
};
zone "20.254.192.in-addr.arpa" {
    type slave;
    masters { 192.253.254.2; };
    file "bak.192.254.20";
};

zone "." {
    type hint;
    file "db.cache";
};
```

在堡垒主机上使用视图

如果在堡垒主机上运行 BIND 9，我们就可以使用视图在同一名字服务器上既对外安全地提供影子 *movie.edu*，又能解析 Internet 域名。那样，就可以避免在我们的 Web 服务器 *www.movie.edu* 所在的主机上运行外部名字服务器。如果不这样做，我们将使用两台名字服务器来通告外部 *movie.edu*。

其配置与第十章“视图”一节所展示的配置非常相似。

```
options {
    directory "/var/named";
};

acl "internal" {
    127/8; 192.249.249/24; 192.253.253/24; 192.253.254/24; 192.254.20/24;
};

view "internal" {
    match-clients { "internal"; };
    recursion yes;
    zone "movie.edu" {
        type slave;
        masters { 192.249.249.3; };
        file "bak.movie.edu";
    };

    zone "249.249.192.in-addr.arpa" {
        type slave;
        masters { 192.249.249.3; };
        file "bak.192.249.249";
    };

    zone "253.253.192.in-addr.arpa" {
```

```
        type slave;
        masters { 192.249.249.3; };
        file "bak.192.253.253";
    };

    zone "254.253.192.in-addr.arpa" {
        type slave;
        masters { 192.253.254.2; };
        file "bak.192.253.254";
    };

    zone "20.254.192.in-addr.arpa" {
        type slave;
        masters { 192.253.254.2; };
        file "bak.192.254.20";
    };

    zone "." {
        type hint;
        file "db.cache";
    };
};

view "external" {
    match-clients { any; };
    recursion no;

    acl "ns1.isp.net" { 199.11.28.12; };

    zone "movie.edu" {
        type master;
        file "db.movie.edu.external";
        allow-transfer { "ns1.isp.net"; };
    };

    zone "4.1.200.in-addr.arpa" {
        type master;
        file "db.200.1.4";
        allow-transfer { "ns1.isp.net"; };
    };

    zone "." {
        type hint;
        file "db.cache";
    };
};
```

注意, 内部和外部视图提供*movie.edu*的不同版本：一个从区数据文件*db.movie.edu*加载, 而另一个从*db.movie.edu.external*中加载。如果在外部视图里有更多的区, 我们也许会使用一个不同于内部区数据文件的子目录以管理外部区数据。

DNS 安全扩展

我们已经在本章的前面描述了TSIG, 它非常适合于保护两台名字服务器之间或更新者与名字服务器之间的通信安全。然而，如果你的名字服务器不能保护其自身安全的话，TSIG 就不能保护你，这就好像有人闯入了某台运行你的名字服务器的主机，他就可能获取该名字服务器的 TSIG 密钥。而且，因为 TSIG 使用共享 secret，所以在众多的名字服务器中配置 TSIG 并不实用。因为不能发布和管理众多的密钥，因此不能使用 TSIG 保护你的名字服务器与任意 Internet 名字服务器之间的通信安全。

处理类似的密钥管理问题的最普遍方法就是使用公共密钥加密系统（public key cryptography）。在 RFC 2535 中说明的 DNS 安全扩展（DNS Security Extensions）用公共密钥加密系统使区管理员对其区数据进行数字签名，从而证明数据的真实性。

注意：我们将讨论当前在RFC 2535描述的DNS安全扩展, 即DNSSEC。然而, IETF的DNSEXT工作组仍在完善 DNSSEC，也许在其成为标准之前还要修改 DNSSEC 的某些方面。

另外注意，尽管 BIND 8 早在 BIND 8.2（注 5）就提供对 DNSSEC 的基本支持，但直到 BIND 9 之前，DNSSEC 都不能在实际中使用。因而接下来，我们将使用 BIND 9 作为我们的例子，你不应当使用任何旧版本的 BIND。

公共密钥加密系统和数字签名

公共密钥加密系统用非对称加密算法解决了密钥发布的问题。在非对称加密算法中，一个密钥用来解密数据，而另一个用于加密。这两个密钥（密钥对）由一个数学公式同时生成。这是找到特殊非对称性（一个密钥用于解密另一个密钥所加密的信息）两个密钥的一个非常简单的方法。对这种算法而言，已知一个密钥而要确定另一个密钥是非常困难的 （对最流行的的非对称加密算法 RSA 来说，已知一个密钥而要确定另一个密钥牵涉到大自然数的因式分解）。

公共密钥加密系统中，个体（individual）先产生一个密钥对。然后，公开该密钥对

注 5： 实际上，BIND 不遵从委托链（chain of trust），它只能对 *trusted-keys* 语句所作用的区中的 SIG 记录进行验证。

中的一个密钥（例如，在目录系统中发布），与此同时，将另一个密钥保密。某个打算与那个个体安全通信的人能够用该个体的公共密钥加密消息，然后给个体发送加密后的消息。（他甚至可以将加密后的消息贴到新闻组或Web站点上。）如果接收者秘密地保存了私钥，那么只有他可以解密该消息。

相反地，个体可以用其私钥加密消息并且发送给某人。用该个体的公共密钥解密消息，接收者就能够证实该消息是否来自该个体。如果消息被解密成某些合理的信息（比如，不是乱七八糟的语言），而且发送者拥有私钥，那么该个体必定加密了消息。成功的解密也证实了消息在传输中未被篡改（例如，消息通过了一个邮件服务器），因为如果消息被修改了，它就不能被正确解密。因此，接收者验证了消息。

不幸的是，用非对称加密算法加密大容量数据非常缓慢 —— 相对于用对称加密算法而言更加慢。但是当使用公共密钥加密验证消息的真实性（不是私有性）时，我们不需要加密整个消息。我们首先用单向散列函数操作消息，而后仅加密代表原始消息的散列值。我们可以将加密后的散列值，如今称为数字签名（digital signature），附加到我们打算验证真实性的消息上。通过解密数字签名并用其自身的单向散列函数操作接收的消息，接收者仍旧能验证消息的真实性。如果散列值匹配，消息则是真实的。消息的签名和验证过程如图 11-7 所示。

KEY 记录

在 DNS 安全扩展（DNSSEC）中，每个保护区（secure zone）都有对应的密钥对。区的私钥存放在某个安全的地方 —— 通常在名字服务器文件系统的一个文件里。区的公钥被公布为一个新的记录类型，即 KEY 记录，它被附加到区的域名。

如果仔细研究KEY记录，会发现KEY记录实际是一种多用途的记录。能够使用KEY记录存储不同种类的加密密钥，不仅仅是DNSSEC使用的区的公钥。然而，在本书中，我们仅讨论区的公钥。

KEY 记录如下所示：

```
movie.edu. IN KEY 256 3 1 AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX7+uBxB11Bq
L7 LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B
```

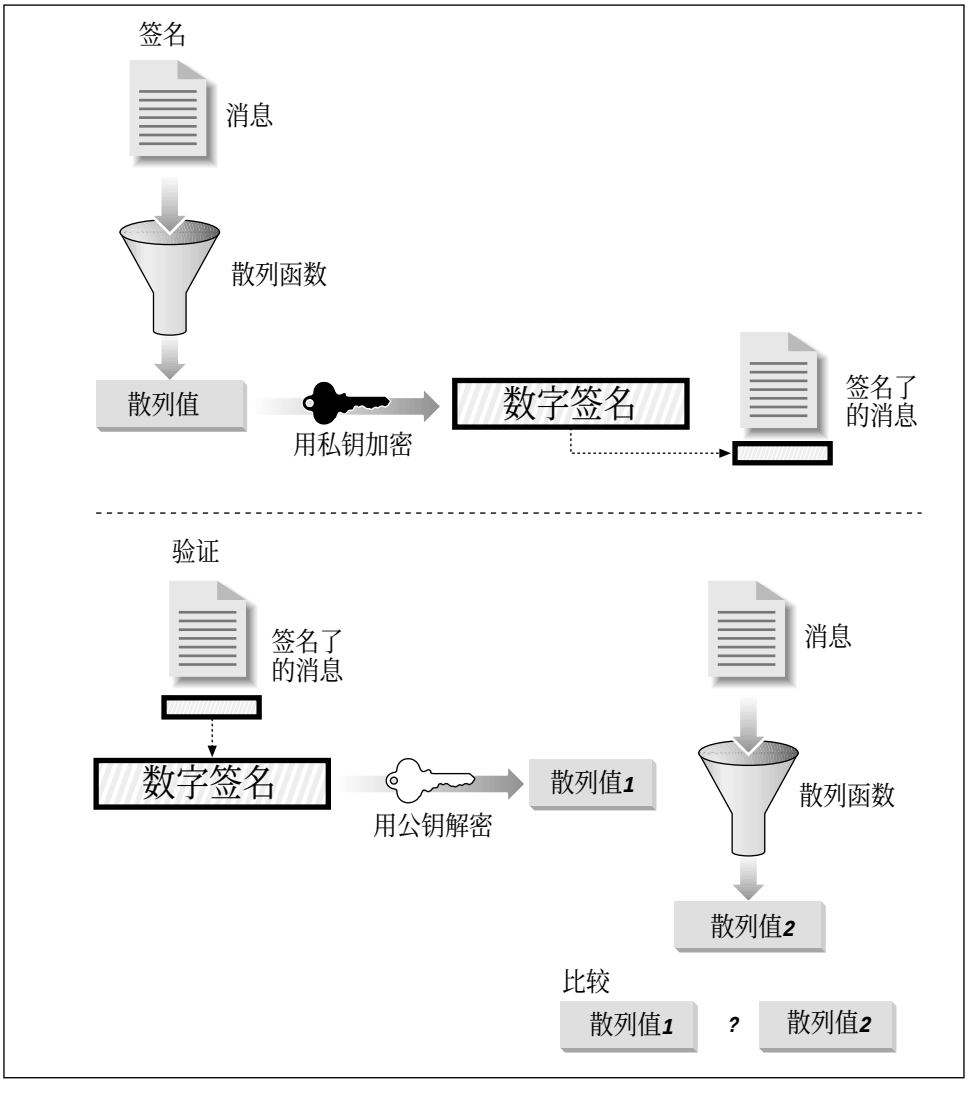


图 11-7　签名并验证一条消息

所有者就是拥有该公钥的区的域名。类型之后的第一个字段 256 是标志字段。标志字段长两字节，并编码成一组 1 位或 2 位的值：

```
  0   1   2   3   4   5   6   7   8   9   0   1   2   3   4   5
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  A/C  | Z | XT| Z | Z | NAMTYP| Z | Z | Z | Z |      SIG      |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

如果第一位的值是0，该密钥可用于认证。很明显，不能用于认证的密钥在DNSSEC中也不是很有用，因而该位的值总为0。

如果第二位的值为0，该密钥用于机密（confidentiality）。DNSSEC并不使你的区数据私有，但也不禁止你将你的区公钥用于机密。因而对区公钥来说，该位的值总为0。

前两位均为1的KEY记录被称为null密钥。在后面，我们将展示null密钥如何在父区中使用。

第三位被预留用于将来使用。现在，它的值必须是0。第四位是“标志扩展”位。它被设计用来提供未来扩展。如果该位被设置，那么KEY记录必须在算法字段（类型后面的第三个字段）之后、公钥（一般在第四个字段）之前包含另一个两字节的字段。该附加的两字节字段中的位含义仍未定义，到目前为止第四位总为0。类似第三位，第五和第六位被预留，且必须是0。

第七和第八位是密钥类型的编码：

00
: 用户密钥。邮件用户代理可使用用户的密钥对发给该用户的电子邮件加密。在DNSSEC中，该密钥类型未被使用。

01
: 区的公钥。所有的DNSSEC密钥均为该类型。

10
: 主机密钥。IPSEC的实现可使用主机密钥加密所有发送到该主机的IP包。DNSSEC不使用主机密钥。

11
: 预留，给将来使用。

第九到第十二位预留，必须为0。最后四位是签名字段，现已不用。

在前面显示的KEY记录中，标志字段（记录中在类型之后的第一个字段）表明该KEY记录是*movie.edu*的区密钥，可被用于认证和机密。

记录中的下一个字段，在前面的例子中的值为 3，称为协议字节（protocol octet）。因为可以为不同的用途使用KEY记录, 所以必须指定一个特定的密钥被用于何种用途。

下列值被定义：

0 保留。

1 用于 RFC 2246 中描述的传输层安全（TLS）。

2 用于电子邮件的连接，例如，S/MIME 密钥。

3 用于 DNSSEC，显然所有 DNSSEC 密钥的协议字节的值都为 3。

4 用于 IPSEC。

255
用于使用 KEY 记录的任意协议。

所有在 4 到 255 之间的值都可用于未来分配。

KEY 记录中的下一字段（第三个字段）是算法号，在前面的例子中的值为 1。DNSSEC可以使用许多公共密钥加密算法, 因此需要确定区使用的算法和该密钥使用的算法。下列值被定义：

0 保留。

1 RSA/MD5。RFC 2535推荐、但不要求使用RSA/MD5。然而，RSA非常流行，而且 RSA 算法所涉及的专利最近期满了，所以已经开始出现一些强制使用 RSA 的讨论。

2 Diffie-Hellman。RFC 2535 使 Diffie-Hellman 为可选使用。

3 DSA。RFC 2535 支持强制使用 DSA（而非一定要使用）。然而，前面提到过，这可能不久就要改变。

4 为基于椭圆形曲线（elliptic curve-based）的公共密钥算法而保留。

因为认为RSA密钥可能成为标准，所以我们在例子中使用RSA密钥。

KEY记录中最后的字段是以64为基数编码的公钥本身。DNSSEC支持长度更长的密钥，这一点我们将在不久后生成*movie.edu*公钥时可以看见。密钥越长，找到对应私有密钥越困难，但用私有密钥对区数据签名和用公共密钥检验区数据所花费的时间也越长。

尽管null密钥有协议字节和算法号，但其没有公共密钥。

SIG记录

如果KEY记录存储了区的公共密钥，那么必然存在一个存储相应私有密钥签名的新记录，对吗？当然如此，这个新记录就是SIG记录。SIG记录存储针对一个*RRset*的私有密钥的数字签名。一个RRset就是具有相同所有者、类和类型的一组资源记录；例如，所有*wormhole.movie.edu*的地址记录组成了一个RRset。相似地，所有*wormhole.movie.edu*的MX记录组成了另一个RRset。

为何对RRset签名而不是对单个记录呢？这样做是为了节省时间。不存在仅查询一个*wormhole.movie.edu*地址记录的方法；名字服务器总是按组返回地址。所以为何在可以对组签名的情况下还要自找麻烦地对单个记录签名呢？

下面是包含*wormhole.movie.edu*地址记录的SIG记录：

```
wormhole.movie.edu.              SIG     A 1 3 86400 20010102235426 (
                                 20001203235426 27791 movie.edu.
                                 1S/LuuxhSHs2LknPC7K 7v4+PNxESKZnjX6CtgGLZDWf
                                 Rmovkw9VpW7htTNJYhzlFck/BOk17tRj0fbQ6JWaA== )
```

所有者名是*wormhole.movie.edu*，与被签名记录的所有者相同。类型后面的第一个字段，例子中的值为A，称为被包含的类型（type covered）。该字段告诉我们*wormhole.movie.edu*中的何种记录被签名了；在本例中，是地址记录。对于*wormhole.movie.edu*可能所有的每种类型的记录，必然存在一个独立的SIG记录。

第二个字段，例子中的值为1，是算法号。该字段的值与KEY记录中算法号字段的值具有相同的定义，因此1就代表RSA/MD5。如果你产生一个RSA密钥并用它对你的区数据签名，自然你将得到RSA/MD5签名。如果你用多种类型的密钥签名你

的区，比如用 RSA 密钥和 DSA 密钥，最终针对每个 RRset，你将得到两个 SIG 记录，一个的算法号是 1（即 RSA/MD5），另一个的算法号是 3（即 DSA）（注 6）。

第三个字段称为标签（label）字段。它指示在签名记录的所有者名中的标签数目。显然，*wormhole.movie.edu* 有三个标签，因此标签字段的值为 3。究竟何时标签字段的值与SIG记录所有者名字的标签数目不同呢？何时SIG记录包含一个某种类型的通配符记录呢？不幸的是（或者也许是幸运的），在保护区中，BIND不支持通配符记录。

第四个字段是签名 RRset 中记录的原始 TTL（一个 RRset 中的所有记录被认为具有相同的 TTL 值）。在这里存放 TTL 的原因是缓存 SIG 记录所包含 RRset 的名字服务器将递减缓存记录的 TTL。如果没有原始的TTL，对原始地址记录而言，通过在其原始状态的单向散列函数来验证数字签名是不可能的。

后两个域分别是签名期满和初始字段。它们都存储成以秒为单位的无符号整数，该值是自从 Unix 时代（即 1970 年 1 月 1 日）以来的秒数，但在 SIG 记录的文本表示中，为方便起见，它们表示成 YYYYMMDDHHMMSS 的格式。（上面所显示的 SIG 记录的签名期满时间是 2001 年 1 月 2 日下午 11 点 54 分后。）签名初始时间通常是你运行程序来签名区的时间。运行签名程序时，你也可以选择签名期满时间。在签名期满时间后，SIG 记录不再有效，不能用于验证 RRset。真失败。这表示你必须周期性地对你的区数据重新签名，以保持签名有效。有趣。感到高兴的是，重新签名比第一次签名花费的时间要少。

SIG 记录中的下一个字段（第七个字段），例子中是 27791，是密钥标记（keytag）字段。密钥标记是从签名区的私有密钥相对应的公有密钥派生而来的指纹。如果区有多个公钥（当你更改密钥时你将得到多个公钥），DNSSEC 验证软件使用密钥标记来确定到底是哪个密钥用于验证签名。

包含 *movie.edu* 的第八个字段是签名人的名字（signer's name）字段。正如你预料的那样，该字段是某公钥的域名，验证器要用它来检查签名。与密钥标记字段一起，签名人的名字字段确定要使用的KEY记录。在大多数情况下，签名人的名字字段是

注 6： 可以用两种算法的密钥对你的区签名，这样，仅支持 DSA 的软件就可以验证你的数据，同时那些使用 RSA 的人也能够用 RSA 验证你的数据。

签名记录所在区的域名。然而，在一种情况下（我们后面将讨论）签名人的名字是父区的域名。

最后的字段是签名（signature）字段。这是针对签名记录以及除去该字段的 SIG 记录的区的私钥数字签名。与KEY 记录中的密钥相同，该签名是以64 为基数编码的。

NXT 记录

DNSSEC 引入了一种更新的记录类型：NXT 记录。我们将解释它的用途。

如果你在保护区中所查询的域名不存在时会发生什么？如果该区不安全，名字服务器将简单地用“无此域名”的响应代码来应答。但是如何签名一个响应代码呢？如果你签名整个响应消息，那么它将很难缓存。

NXT记录解决了对否定响应签名的问题。NXT记录跨越区中两个相邻域名之间的间隙，告诉你在给定域名之后的域名 —— 用记录名来表示。

但是相邻域名是否暗示着区中的域名应规范排序呢？确实如此。

要排列区中的域名，可从域名中最右的标签开始排序，而后按从右到左的顺序对标签排序。标签排序与字母大小写无关，且按字典顺序，即数字在字母之前且不存在的标签在数字之前（换句话说，*movie.edu* 应在 *0.movie.edu* 之前）。因而 *movie.edu* 中的域名将如下排序：

```
movie.edu
bigt.movie.edu
carrie.movie.edu
cujo.movie.edu
dh.movie.edu
diehard.movie.edu
fx.movie.edu
bladerunner.fx.movie.edu
outland.fx.movie.edu
horror.movie.edu
localhost.movie.edu
misery.movie.edu
robocop.movie.edu
shining.movie.edu
terminator.movie.edu
wh.movie.edu
wh249.movie.edu
```

```
wh253.movie.edu
wormhole.movie.edu
```

注意，类似 *movie.edu* 在 *big.movie.edu* 之前一样，*fx.movie.edu* 在 *bladerunner.fx.movie.edu* 之前。

一旦区按规范排序，NXT 记录就有意义了。下面是 *movie.edu* 的一个 NXT 记录：

```
movie.edu.                    NXT   bigt.movie.edu. ( NS SOA MX SIG NXT )
```

该记录说明区中 *movie.edu* 之后的下一个域名是 *bigt.movie.edu*，这一点我们能从域名排序列表中看出。它也说明 *movie.edu* 有 NS 记录、一个 SOA 记录、MX 记录、一个 SIG 记录和一个 NXT 记录。

区中最后一个 NXT 记录是特殊的。因为在最后一个记录之后不存在下一个域名，最后的 NXT 记录将绕回指向区中第一个记录：

```
wormhole.movie.edu.                NXT     movie.edu. ( A SIG NXT )
```

换句话说，为显示 *wormhole.movie.edu* 是区中的最后一个域名，我们说下一个域名是 *movie.edu*，即区中的第一个域名。

那么 NXT 记录是如何提供已认证的否定响应呢？如果在内部查询 *www.movie.edu*，你将获得 *wormhole.movie.edu* NXT 记录，告诉你不存在 *www.movie.edu*，因为在 *wormhole.movie.edu* 之后区中不存在域名。相似地，如果你力图查询 *movie.edu* 的 TXT 记录，你将得到我们显示给你的第一个 NXT 记录，该记录告诉你不存在 *movie.edu* 的 TXT 记录，而只有 NS、SOA、MX、SIG 和 NXT 记录。

包含 NXT 记录的 SIG 记录与 NXT 记录一起出现在响应中，用来认证你所寻找的不存在的域名或数据类型。

NXT 记录，特别地鉴别区中不存在的数据，这是非常重要的。一个简单而包罗万象的记录 “ 那不存在 ”可以在线下被捕获，并被重复使用来错误地宣称实际存在的域名或记录并不存在。

对于那些担心添加新记录到区中并需要手动更新记录的人来说 —— 哦哦，现在我添加了一个主机，我必须调整我的 NXT 记录 —— 要振作起来：BIND 提供了工具来为你自动添加 NXT 和 SIG 记录。

你们中的一些人也许担心NXT记录显示了你的区信息。例如，黑客能够查询附加到区域名的NXT记录来发现下一个域名，然后重复该过程来获取区中所有的域名。不幸的是，那是保护你的区安全带来的不可避免的副作用。记住：我的区数据是安全的，但是也是公开的。

委托链

我们应当讨论的 DNSSEC 理论还有一个方面：委托链（Chain of Trust）。（不，这并非感觉上难以处理的组生成练习。）目前为止，在我们保护区中的每个RRset有一与之相联系的SIG 记录。为了允许其他人验证 SIG 记录，我们的区在 KEY 记录中向外界公布它的公共密钥，但是想像一下，如果有人进入我们的主名字服务器中，有什么东西能够阻止他创建自己的密钥对呢？然后，他可以修改我们的区数据，用他新创建的私钥重新对我们的区签名，并且在 KEY 记录中发布他新创建的公钥。

为解决该问题，我们的公钥由更高的权威机构来“ 认定 ”。这个更高的权威机构证明我们KEY 记录中的 *movie.edu* 公钥真正属于拥有和维护该区的组织，而非一些任意的组织。在认定我们之前，该更高的权威机构要查询一些证据，来证明我们正是我们所说的那个人，以及我们是 *movie.edu* 的授权管理员。

这个更高的权威机构是我们的父区 *edu*。当我们创建密钥对，并对区签名时，我们也向*edu*的管理员发送我们的公钥，以及我们作为*movie.edu*的两个真实的管理员的身份和位置的证明（注 7）。他们用 *edu* 区的私钥对我们的KEY 记录签名，并将签名发回给我们，因此可以将该签名添加到我们的区中。下面是我们的 KEY 记录和伴随的 SIG 记录：

```
movie.edu               IN SIG  KEY 1 2 3600 20010104010141 (
                               20001205010141 65398 edu.
                               aE4sCZKgFtp5RuD1sib0+19dc3MF/y9S2Fr8+h66g+Y2
                               1bc31M4y0493cSoyRpapJrd7qfG+Cr7GK+uY+eLCRA== )
                        KEY     256 3 1 (
                               AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX
                               7+uBxB11BqL7LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B )
```

注 7： 实际上，任何顶级区都不会对它们子区的 KEY 记录签名。尽管如此，欧洲的注册机构可能要开始对其子区的 KEY 记录签名。

注意，SIG记录中签名者的名字字段是*edu*，而非*movie.edu*，显示我们的KEY记录是由我们父区的私钥来签名的，而非我们自己的私钥。

如果有人进入*edu*区的主名字服务器会发生什么呢？*edu*区的KEY记录由根区的私钥来签名，而根区呢？根区的公钥是广为人知的，配置在支持DNSSEC的每一个名字服务器上。

也就是说，一旦DNSSEC广泛实现，根区的公钥将在每一个名字服务器上配置。而现在，根区和*edu*区都未签名，也没有密钥对。尽管如此，在DNSSEC广泛实现之前，零星使用DNSSEC还是可能的。

安全根

假设我们打算开始在电影大学使用DNSSEC来提高我们区数据的安全性。我们已经对*movie.edu*区签名，但不能让*edu*对我们的KEY记录签名，因为*edu*还未使其区安全并且没有密钥对。那么Internet上的其他名字服务器如何验证我们的区数据呢？我们自己的名字服务器如何验证我们的区数据呢？

BIND 9名字服务器提供了一种指定公共密钥的机制，该公钥对应*named.conf*文件中的特定区：*trusted-keys*语句。下面就是针对*movie.edu*的*trusted-keys*语句：

```
trusted-keys {
    movie.edu. 256 3 1 "AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX7+uBxB11Bq
L7 LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B";
};
```

它基本上就是KEY记录，但不包含类（class）和类型（type）字段，以及所引用的密钥。区的域名可被引用，但不是必需的。如果*movie.edu*有多个公钥（比如，一个DSA密钥），我们也能够包括它：

```
trusted-keys {
    movie.edu. 256 3 1 "AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX7+uBxB11Bq
L7 LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B";
    movie.edu. 256 3 3 "AMnD8GXACuJ5GVnfCJWmRydg2A6JptSm6tjH7QoL81SfBY/kcz1Nbe
Hh z4l9AT1GG2kAZjGLjH07BZHY+joz6iYMPRCDaPOIt9LO+SRfBNZg62P4 aSPT5zVQPahDIMZmTIvv
O7FV6IaTV+cQiKQl6noro8uTk4asCADrAHw0 iVjzjaYpoFF5AsB0cJU18fzDiCNBUb0VqE1mKFuRA/K
1KyxM2vJ3U7IS to0IgACiCfHkYK5r3qFbMvF1GrjyVwfwCC4NcMsqEXIT8IEI/YYIgFt4 Ennh";
};
```

该*trusted-keys*语句使BIND 9名字服务器可以验证*movie.edu*区中的任何记录。名字服务器也能够验证子区（如，*fx.movie.edu*）中的任何记录，假设子区的KEY记录由*movie.edu*的私钥签名，并且可验证孙子区的记录，假定有一个有效的委托链可回溯到*movie.edu*区的公钥。换句话说，*movie.edu*成为安全根，我们的名字服务器可以验证在安全根之下的任意安全区数据。

null密钥

安全根允许你的名字服务器验证在名字空间中某节点之下的签名记录。*null*密钥与之相反：它们告诉你的名字服务器在名字空间中某节点之下的记录是非安全的，假设*fx.movie.edu*的管理员并未使他们的区安全。当我们对*movie.edu*签名时，BIND 9签名软件针对*fx.movie.edu*添加一个特殊的null密钥到*movie.edu*区：

```
fx.movie.edu.           KEY     49408 3 3 (
                                )
```

注意，在记录中不存在以64为基数编码的公钥。如果非常仔细地查看（或者取出你的科学计算器），你将发现标志字段说明该密钥不应当用于认证或机密（confidentiality）；也就是说，它是null密钥。支持DNSSEC的名字服务器将其解释为*fx.movie.edu*区不安全，并且名字服务器不应该期待来自该区的签名数据。

如果BIND 9的签名软件运行时发现一个文件包含*fx.movie.edu*的KEY记录，该软件将忽略null密钥，表明*fx.movie.edu*是安全的。

记录如何使用

我们将仔细检查支持DNSSEC的名字服务器是如何验证*movie.edu*中的记录的。特别是，我们将看看当查询*wormhole.movie.edu*的地址时会发生什么。首先，当然，名字服务器发送查询地址的请求：

```
%  dig +dnssec +norec wormhole.movie.edu.

; <<>> DiG 9.1.0 <<>> +dnssec +norec wormhole.movie.edu.
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15766
;; flags: qr aa ra; QUERY: 1, ANSWER: 3, AUTHORITY: 4, ADDITIONAL: 6
```

```
;; OPT PSEUDOSECTION:
; EDNS: version:    0, udp=   4096
;; QUESTION SECTION:
;wormhole.movie.edu.            IN      A

;; ANSWER SECTION:
wormhole.movie.edu.     86400   IN      A       192.249.249.1
wormhole.movie.edu.     86400   IN      A       192.253.253.1
wormhole.movie.edu.     86400   IN      SIG     A 1 3 86400
20010215174848 20010116174848 27791 movie.edu.
cYKQvgVksHjwGedNz72iyIpjXBhtSOeUEQA6V0b6l8asG3mpV6hzrzNf
YwTpLoh9FSjSf0kUzmXkW9aYJmd5Bw==

;; AUTHORITY SECTION:
movie.edu.              86400   IN      NS      outland.fx.movie.edu.
movie.edu.              86400   IN      NS      wormhole.movie.edu.
movie.edu.              86400   IN      NS      terminator.movie.edu.
movie.edu.              86400   IN      SIG     NS 1 2 86400
20010215174848 20010116174848 27791 movie.edu.
ZXRnlbJBWJa4XX3YTWgkYnoQjGLFDN+2JwoGpLpxTidwkJ0FT+N3gMSw anSxa22b+X/
7v4b99t2WMcxCtUIXvw==

;; ADDITIONAL SECTION:
outland.fx.movie.edu.   86400   IN      A       192.253.254.3
terminator.movie.edu.   86400   IN      A       192.249.249.3
terminator.movie.edu.   86400   IN      SIG     A 1 3 86400
20010215174848 20010116174848 27791 movie.edu.
GSnxseyN4w5sA2Fb9uK9zVNSRJRbbcvr0DaDRwLDO8X2m6ZBbkRssSHJ
tZYwoO4ZIFERLKakB//VTDMhYJmNvw==
movie.edu.              86400   IN      KEY     256 3 1
AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX7+uBxB11BqL7 LAB7/
C+eb0vCtI53FwMhkkNkTmA6bI8B
movie.edu.              86400   IN      SIG     KEY 1 2 86400
20010215174837 20010116174837 65398 edu. LW+nc2gmz618u/
LjDtlKSorv9OkJOwC8wj/sa/CpzCJJqceB/55JhsWI
t1ADlfQwb4h9hs6oMeN2sU9jHiYQmw==

;; Query time: 3 msec
;; SERVER: 206.168.194.122#53(206.168.194.122)
;; WHEN: Tue Jan 16 10:49:48 2001
;; MSG SIZE  rcvd: 671
```

注意，我们必须在命令行指定 *+dnssec*。仅当查询者指示它能够处理 DNSSEC 时，BIND 9.1.0 及后续版本的名字服务器才在响应中包括 DNSSEC 记录（SIG、NXT 和 KEY）。查询者如何指示它能够处理 DNSSEC 呢？这是通过在协议首部的“伪部分”（pseudosection）设置一个特殊标记来实现的，称它为伪部分是因为它并非协议首部中的实际部分。实际上，它是一个新的记录类型 OPT，携带在 DNS 消息中。OPT 记录通常指示查询者的能力。

还要注意，响应应包括四个SIG记录：一个覆盖在回答段中的记录，一个覆盖在权威段中的记录，一个覆盖在附加段中的*terminator.movie.edu*的地址记录，最后一个则覆盖在附加段中的*movie.edu*的KEY记录。附加段包括覆盖*outland.fx.movie.edu*的一个SIG记录，*wormhole.movie.edu*的地址记录，以及覆盖这些地址（如果地址记录能装配在一个UDP数据报中，但通常情况下，UDP数据报没有足够的空间）的一个SIG记录。

为验证SIG记录，名字服务器必须看看包含在附加段中的*movie.edu*的KEY记录。但是在使用该密钥之前，必须验证该密钥的SIG。验证至少需要一个附加的查询：向其中一个*edu*名字服务器查询*edu*区的公钥 —— 除非名字服务器已经从*trusted-keys*语句中知道了*movie.edu*公钥。

DNSSEC和性能

从*dig*的输出明显看出DNSSEC增加了DNS消息的平均大小，因此验证区数据的名字服务器需要更多的计算能力，并且对区数据签名显著地增加了区的大小 —— 当前的估计是签名使区的大小增大了七倍。这些影响有其相应的后果，其中的一些后果不太明显。

- 更大的DNS消息意味着更多的截断（truncated）消息，这意味着更多地依赖TCP的使用。显然，与UDP相比，使用TCP将占用更多的资源。
- 区数据的验证将花费时间并且减慢解析过程。
- 更大的区意味着更大且更难管理的*named*进程。

实际上，DNSSEC的复杂性意味着BIND 8的体系结构不能支持DNSSEC。DNSSEC也为开发BIND 9并设计它使之运行在多处理器主机上提供部分的推动力。如果你计划对你的区签名，要确信你的权威名字服务器有足够的内存加载新的、更大的区。如果你的名字服务器要解析保护区中的更多记录，要确信它们有足够的处理器能力来验证所有的数字签名 —— 并且记住BIND 9能够利用你添加到BIND 9所运行的主机上的任何处理器。

给区签名

好了，现在你已经知道实际给区签名的理论背景。我们将向你展示我们是如何给 *movie.edu* 签名的。记住，我们使用 BIND 9 工具 —— 相对于 BIND 8 工具，它们更易于使用，并且 BIND 9 要比 BIND 8 更加完善地支持 DNSSEC。

创建你的密钥对

首先，我们为 *movie.edu* 创建密钥对：

```
#   cd /var/named
#   dnssec-keygen -a RSA -b 512 -n ZONE movie.edu.
Kmovie.edu.+001+27791
```

我们在名字服务器的工作目录下运行 *dnssec-keygen*。为方便起见：区数据文件也在该目录下，所以我们不需要使用绝对路径作为参数。如果我们打算使用 DNSSEC 的动态更新，我们需要名字服务器的工作目录下的密钥。

回忆本章 TSIG 一节描述的 *dnssec-keygen* 选项（很久以前了）：

−a 所使用的加密算法，例子中为 RSA。我们也可以使用 DSA，但 RSA 更有效。

−b 所创建的密钥长度，按位计算。RSA 密钥在 512~2000 位之间。DSA 密钥在 512~1024 位之间，只要长度可被 64 整除。

−n 密钥类型。DNSSEC 密钥总是区密钥。

惟一的非可选参数是区的域名 *movie.edu*。*dnssec-keygen* 程序输出密钥被写入文件的基本名。基本名之后的数字（001 和 27791），正如我们在 TSIG 部分解释的，是类似于用于 KEY 记录的密钥的 DNSSEC 算法号（001 是 RSA/MD5）和密钥的指纹（在多个密钥与同一区相关联时，密钥的指纹被用来区别不同的密钥）。

公钥被写入文件 *basename.key*（*Kmovie.edu.+001+27791.key*）。私钥被写入文件 *basename.private*（*Kmovie.edu.+001+27791.private*）。记住，要保护你的私钥；任何知道私钥的人都可以仿照签名的区数据。*dnssec-keygen* 做了它能够帮助你的：*dnssec-keygen* 使得只有运行该程序的用户可以读写 *.private* 文件。

发送用来签名的密钥

下面，我们发送KEY记录给父区的管理员，使之对父区签名。BIND 9包含一个很好的小程序来为传输打包密钥，该程序就是*dnssec-makekeyset*：

```
#   dnssec-makekeyset -t 172800 Kmovie.edu.+001+27791.key
```

*dnssec-makekeyset*创建了一个名为*keyset-movie.edu*的文件（注8），它包含以下内容：

```
$ORIGIN .
$TTL 172800      ; 2天
movie.edu             IN SIG  KEY 1 2 86400 20010104034839 (
                              20001205034839 27791 movie.edu.
                              M7RDKMyc9w1djDc0mQAXQc1PJdmLRBg3nfaGEUZe9Fbi
                              mjiNVaQK33IWhzI95oD8AS0WqRDy5TusTXt4nx1/dQ== )
                      KEY     256 3 1 (
                              AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX
                              7+uBxB11BqL7LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B )
```

*–t*选项为记录提交了一个TTL值。这意味着向你的父区管理员建议了你所希望的该记录的TTL值（按秒计算）。当然，父区管理员可以忽略它。SIG记录实际上包含一个与你的区的KEY记录有关的签名，该签名用你的区的私钥产生。那证明你确实有与KEY记录中公钥相对应的私钥 —— 你并非随意提交一个KEY记录。

签名的期限和初始字段默认分别是“现在”和“从现在开始30天”。这些就向签名者建议你所期望的签名的生命周期。可以使用–s(start)和–*e*(end)选项调整签名的期限和初始时间。这两个选项或者接受格式为YYYYMMDDHHMMSS的绝对时间为参数，或者是以相对时间为参数。对于–*s*，相对时间是基于当前时间计算的。对于–*e*，相对时间是基于开始时间计算的。

你也可以用签名的期限和初始字段来绑定几个密钥集，并一次性地提交给你的父区管理员去签名。例如，你可以一次性提交一月、二月和三月的有效密钥集给你的父区管理员，而后每月使用一个密钥集。

注8： BIND 9.0.1和以前版本附带的*dnssec-makekeyset*版本所生成的文件名为*movie.edu.keyset*。但是，使区的域名首先出现将导致以下问题的产生：根区的keyset文件名为*.keyset*，在Unix文件系统中这意味着一个隐含文件。

随后，我们把文件寄给我们的父区管理员去签名。因为消息包含了我们的身份证明（注 9），父区管理员使用 *dnssec-signkey* 程序对消息签名：

```
#   dnssec-signkey keyset-movie.edu Kedu.+001+65398.private
```

并且把结果文件 *movie.edu.signedkey* 发送给我们：

```
$ORIGIN .
$TTL 172800       ; 1小时
movie.edu               IN SIG  KEY 1 2 3600 20010104010141 (
                                20001205010141 65398 edu.
                                aE4sCZKgFtp5RuD1sib0+19dc3MF/y9S2Fr8+h66g+Y2
                                1bc31M4y0493cSoyRpapJrd7qfG+Cr7GK+uY+eLCRA== )
                        KEY     256 3 1 (
                                AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX
                                7+uBxB11BqL7LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B )
```

如果不关心签名 KEY 记录，我们将忽略这一步。但是这时，只有针对 *movie.edu* 的 *trusted-keys* 条目的名字服务器能够验证我们的数据。

对你的区签名

在对我们的区签名之前，必须添加 KEY 记录到我们基本的区数据文件中：

```
# cat "$INCLUDE Kmovie.edu.+001+27791.key" >> db.movie.edu
```

那么给签名程序一个信息，即签名程序需要知道使用哪个密钥对区签名。签名程序自动发现并包括 *movie.edu.signedkey* 的内容。

而后我们用 *dnssec-signzone* 对区签名：

```
#   dnssec-signzone -o movie.edu. db.movie.edu
```

我们使用 *–o* 选项指定区数据文件的起始，因为 *dnssec-signzone* 没有读 *named.conf* 文件来确定文件所描述的区。惟一的非选项参数是区数据文件的名字。

这就产生了一个新的区数据文件 *db.movie.edu.signed*，该文件如下所示：

```
$ORIGIN .
```

注 9： 因为顶级区还没有开始对区签名，所以仍然存在一些如何要求我们认证我们自己的问题。使用加密的签名邮件是一种可行的方法。

```
$TTL 86400      ; 1天
movie.edu               IN SOA  terminator.movie.edu. al.robocop.movie.edu. (
                                 2000092603 ; 序列号
                                 10800      ; 刷新(3小时)
                                 3600       ; 重试(1小时)
                                 604800     ; 期满(1周)
                                 3600       ; 最小值(1小时)
                                 )
                        SIG     SOA 1 2 86400 20010104041530 (
                                20001205041530 27791 movie.edu.
                                aO1eZKhGSm99GgC9PfLXfHjl3tAWN/Vn33msppmyhN7a
                                RlfvJMTpSoJ9XwQCdjqhz0lcnCnQiL+jZkqU3uUecg== )
                        NS      terminator.movie.edu.
                        NS      wormhole.movie.edu.
                        NS      outland.fx.movie.edu.
                        SIG     NS 1 2 86400 20010104041530 (
                                20001205041530 27791 movie.edu.
                                pkmZJHqFlNmZdNjyupBMMzDDGweGsf9TS1EGci9cwKe5
                                c0o9h/yncInn2e8QSakjxpwKB8aw9D9uiStxJ/sLvQ== )
                        SIG     MX 1 2 86400 20010104041530 (
                                20001205041530 27791 movie.edu.
                                ZcKKeT0XaNlw83eSzRxt74DaLXvQtPYCdgKGOfSiJmYQ
                                WxI5zZUEWA6ku3w48mo9jbVF+/7nF3QcpFTIiwV1ug== )
$TTL 3600       ; 1小时
                        SIG     NXT 1 2 3600 20010104041530 (
                                20001205041530 27791 movie.edu.
                                upMjK21eD7OQkrHpxSWqkOPcRXbfL8WAgQVK1aGHcTPE
                                X3JtaLtCLuKLd3YFs7T8BuZoN7aJYRVREWSPVedYPw== )
                        NXT     bigt.movie.edu. ( NS SOA MX SIG KEY NXT )
$TTL 172800     ; 2天
                        SIG     KEY 1 2 172800 20001205040220 (
                                20001205040219 65398 edu.
                                HIReZ98rieIuRI04XsoL+xLRLe8tCQbNKD8USlV35vb4
                                VsLUGCAEgBq7lLsHty7YCskbxhQu8ncysBKnR/muiA== )
                        KEY      256 3 1 (
                                AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX
                                7+uBxB11BqL7LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B )
$TTL 86400      ; 1天
                        MX      10 postmanrings2x.movie.edu.
```

不管你相信与否，那些就是附加到域名 *movie.edu* 的记录。整体上区数据文件几乎在长度上增加了四倍，在大小上增加了五倍。

最后，我们改变 *named.conf* 文件中的 *zone* 语句，使 *named* 加载新的区数据文件：

```
zone "movie.edu" {
    type master;
    file "db.movie.edu.signed";
};
```

这时，我们重新加载区并检查 *syslog*。

dnssec-signzone 使用一些我们不使用的选项：

-s，*-e*
: 这些选项指定 SIG 记录中使用的签名的期限和初始时间；它们与 *dnssec-makekeyset* 有相同的语法。

-i 指定一个选项参数，即重新对记录签名的循环周期（我们用分钟来表示）。在 BIND 9.1.0 之前，这叫做 *-c* 选项。

-f 指定一个选项参数，即写入签名区的文件名。默认情况下是区数据文件名与 *.signed* 相连而成。

作为第二个非选项参数，你也可以指定用于对区签名的私钥。默认时，*dnssec-signzone* 用目录下每个区的私钥来对区签名。如果你指定一个或一个以上的包含区的私钥的文件名，程序将仅使用那些密钥对区签名。

记住，改变区数据时，每次都需要重新对区签名，尽管每次你的确不需要产生新的密钥对或重新对你的 KEY 记录签名。通过在签名的区数据上运行 *dnssec-signzone*，可以重新对区签名：

```
#   dnssec-signzone -o movie.edu -f db.movie.edu.signed.new db.movie.edu.signed
#   mv db.movie.edu.signed db.movie.edu.signed.bak
#   mv db.movie.edu.signed.new db.movie.edu.signed
#   rndc reload movie.edu
```

这个程序足够智能，它能重新计算 NXT 记录，对新记录签名，并且对签名期限时间即将到来的记录重新签名。默认情况下，*dnssec-signzone* 对签名期限时间超过 7.5 天以内（默认的签名初始和期限时间差的四分之一）的记录重新签名。如果你指定了不同的签名初始和期限时间，*dnssec-signzone* 将相应地调整重新签名的循环（cycle）时间。或者通过 *-i* 选项（以前是 *-c* 选项），你可以简单地指定循环时间。

DNSSEC 和动态更新

dnssec-signzone 不是对区数据签名的惟一方法。BIND 9 名字服务器还可以动态地对更新记录签名（注 10）。

注 10： 这是另一个 BIND 8 没有的 DNSSEC 功能。

只要保护区的私钥在名字服务器的工作目录下（当前的文件名为*.private*），BIND 9名字服务器就能对通过动态更新而添加的任何记录签名。如果任何记录从区中删除或向区中添加，名字服务器也调整（并重新签名）相邻的NXT记录。

我们将展示给你这些操作。首先，我们查询一个*movie.edu*中不存在的域名：

```
%   dig +dnssec perfectstorm.movie.edu.

; <<>> DiG 9.1.0 <<>> +dnssec perfectstorm.movie.edu.
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 4705
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 3

;; OPT PSEUDOSECTION:
; EDNS: version:    0, udp=   4096
;; QUESTION SECTION:
;perfectstorm.movie.edu.                IN      A

;; AUTHORITY SECTION:
movie.edu.              3600    IN      SOA     terminator.movie.edu.
al.robocop.movie.edu. 2001011600 10800 3600 604800 3600
movie.edu.              3600    IN      SIG     SOA 1 2 86400 20010215174848
20010116174848 27791 movie.edu.
Ea0+xyEsj0Hy4JP115r0D0UFVpWfxqf0NQA8hpKwlLCsxJ3rA+sJBg2Q ZiCTEwfAcwGRfbNsRYu/CcuV/
VJTDA==
misery.movie.edu.       86400   IN      NXT     robocop.movie.edu. A SIG NXT
misery.movie.edu.       86400   IN      SIG     NXT 1 3 86400 20010215174848
20010116174848 27791 movie.edu. ZVfV9KbPb8hKZdZirlpv+WnUxv72di8lUgZiot/
JaWDsZPfNoYqSnKPW ND4H92guwj7oR6CgrhsgLJ9dMDYSpg==
```

（我们删减了一点输出。）注意*misery. movie.edu*的NXT记录，表明域名不存在。现在我们将使用*nsupdate*来为*perfectstorm.movie.edu*添加一个地址记录。

```
%  nsupdate
>   update add perfectstorm.movie.edu. 3600 IN A 192.249.249.91
>
```

现在，我们再次查询*perfectstorm.movie.edu*：

```
%   dig +dnssec perfectstorm.movie.edu.

; <<>> DiG 9.1.0 <<>> +dnssec perfectstorm.movie.edu.
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11973
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 4, ADDITIONAL: 9
```

```
;; OPT PSEUDOSECTION:
; EDNS: version:    0, udp=   4096
;; QUESTION SECTION:
;perfectstorm.movie.edu.                IN      A

;; ANSWER SECTION:
perfectstorm.movie.edu. 3600    IN      A       192.249.249.91
perfectstorm.movie.edu. 3600    IN      SIG     A 1 3 3600
20010215195456 20010116185456 27791 movie.edu. C/
JXdCLUdugxN91v0DZuUDTusi2XNNttb4bdB2nBujLxjwwPAf/D5MJz //
cDtuZ3X+uYzhkN8MDROqOwUQuQSA==

;; AUTHORITY SECTION:
movie.edu.              86400   IN      NS      terminator.movie.edu.
movie.edu.              86400   IN      NS      outland.fx.movie.edu.
movie.edu.              86400   IN      NS      wormhole.movie.edu.
movie.edu.              86400   IN      SIG     NS 1 2 86400
20010215195301 20010116195301 27791 movie.edu.
1ZR592izM1AjMusJ26e4lvQ0V9lFiFvQh6hCluBxSv7FwNqF7TcJFImc
W52XhXbHUEtiFOzDqYMHOzPV7j23nA==
```

（我们又删减了一点输出。）现在不但产生了一个地址记录，而且还用 *movie.edu* 的私钥产生了一个 SIG 记录。默认情况下，签名期限设为从更新开始的 30 天，但你能够使用 *sig-validity-internal* 子语句改变它，该子语句采用若干天作为参数（注 11）：

```
options {
    sig-validity-interval 7;  // 在更新的记录上我们要 SIG 来持续 1 周
};
```

签名初始时间总被设为更新前的 1 小时，以便允许验证者的时钟与我们的时钟有略微的不同。

如果我们查询 *perfectstorm2. movie.edu*（尽管该电影的结局如何我不知道）。我们的发现如下所示：

```
%   dig +dnssec perfectstorm2.movie.edu.

; <<>> DiG 9.1.0 <<>> +dnssec perfectstorm2.movie.edu.
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 11232
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 3
```

注 11： 在 BIND 9.1.0 之前，*sig-validity-interval* 将其参数解释为秒，而不是天。

```
;; OPT PSEUDOSECTION:
; EDNS: version:    0, udp=   4096
;; QUESTION SECTION:
;perfectstorm2.movie.edu.       IN      A

;; AUTHORITY SECTION:
movie.edu.              3600    IN      SOA     terminator.movie.edu.
al.robocop.movie.edu. 2001011601 10800 3600 604800 3600
movie.edu.              3600    IN      SIG     SOA 1 2 86400 20010215195456
20010116185456 27791 movie.edu. c1RwtgBX2SO8Q7Hz7vJD0aJfNfA6lsrqH4txHJI/slRpx/
UFYbnz3Gje N0JspZEihdLw0ZYMEiN6hnwRAzB4ag==
perfectstorm.movie.edu. 3600    IN      NXT     robocop.movie.edu. A SIG NXT
perfectstorm.movie.edu. 3600    IN      SIG     NXT 1 3 3600 20010215195456
20010116185456 27791 movie.edu.
qsB9l5AmSrB+qKmv+cKa+htCw84zwaakTmPC2yl+shzSEparrKwIMSR6 x5N69w8cze/
AW+gyFIwQZZkfZInJZA==
```

注意NXT记录：当我们添加*perfectstorm.movie.edu*地址记录时，NXT记录被自动添加，这是因为*perfectstorm.movie.edu*是区中的一个新域名。真好。

如同这一功能给人留下的深刻印象一样，你还应当小心什么时候允许动态更新保护区。你应当确信你使用了严格的认证（如，TSIG）来认证更新，否则你将留给黑客一个简单的后门，黑客可以通过该后门，修改你的安全区。而且你应当保证你有足够的计算能力来执行任务：通常，动态更新不会花费太多处理时间。但是对保护区的动态更新需要重新计算NXT，以及更为重要的非对称加密（计算新的SIG记录），因此你应该希望你的名字服务器花费更长的时间和更多的资源来处理动态更新。

更改密钥

尽管我们说：你对区签名时不需要每次都产生一个新的密钥，但还是存在你需要创建新密钥的可能，因为你已耗尽你的私钥，或者更糟的是你的私钥被破解。

在使用一段时间后，继续用你的私钥对记录签名是危险的。用你的私钥加密的有用数据集越大，通过密码分析学，黑客就越容易确定你的私钥。尽管不存在简单的规则来告诉你何时你的私钥到期，但还是有一些基本的方针：

- 你的区越大，则你的私钥加密的数据越多。如果你的区很大，要更频繁地更改密钥。
- 你的密钥越长，破解它越困难。长密钥不需要像短密钥那样经常更改。

- 你更新区数据和对其签名越频繁，则你的私钥加密的有用数据越多。如果你频繁更新你的区（并且特别是如果你动态更新你的保护区）则需频繁更改密钥。
- 你的区数据越有价值，那么黑客将花费越多的时间和金钱来试图破解你的私钥。如果你的区数据的完整性至关重要，那么你需要频繁更改密钥。

因为我们大约一天才更新一次*movie.edu*，而且区数据不是特别大，所以我们每六个月更改一次我们的密钥对。毕竟，我们仅仅是一所大学。如果我们更关心我们的区数据，我们就要使用更长的密钥或频繁更改密钥。

不幸的是，使用新密钥并非简单地产生一个新的密钥并替换旧密钥。如果你那样做的话，你将使得已缓存区数据的名字服务器无法获取区的KEY记录并验证那些数据。因此，使用新密钥将包含多个步骤：

1. 产生新的密钥对。
2. 生成一个包括你的新KEY记录和你的旧KEY记录的新密钥集，并将其发送给父区管理员。
3. 生成一个仅包括你的新KEY记录的密钥集，并也将其发送给父区管理员。
4. 如果使用*trusted-keys*，在语句中为你的新KEY记录添加一个条目。并告诉其他使用*trusted-keys*的名字服务器。
5. 合并（incorporate）来自父区管理员的签名的密钥集，该密钥集包括新、旧KEY记录。
6. 用新的私钥对你的区数据签名，但是不对区中的旧KEY记录签名。
7. 在所有用旧私钥签名的记录过期后，从区中移走旧KEY记录。
8. 合并来自父区管理员的签名的密钥集，该密钥集仅包括新KEY记录。
9. 用新的私钥对你的区数据签名。

让我们仔细研究一下这个过程。首先，我们产生一对新密钥：

```
#  dnssec-keygen -a RSA -b 512 -n ZONE movie.edu.
Kmovie.edu.+001+47703
```

紧接着，我们生成一个包括新KEY记录和旧KEY记录的新密钥集，并将其发送给父区管理员。

```
#   dnssec-makekeyset -t 172800 Kmovie.edu.+001+27791.key Kmovie.edu.+001+47703.key
#    mail -s "Sign my keys, please" hostmaster@nsiregistry.net < keyset-movie.edu
#   mv keyset-movie.edu keyset-movie.edu.2key
```

而后我们生成一个仅包括新KEY记录的密钥集，并将其发送给父区管理员。

```
%   dnssec-makekeyset -t 172800 Kmovie.edu.+001+47703.key
%    mail -s "Sign my keys, please" hostmaster@nsiregistry.net < keyset-movie.edu
```

（使每个人对我们的密钥签名比起那些消息来将花费更多。）

第一个包括新、旧KEY记录和SIG记录的密钥集包含两个记录：

```
$ORIGIN .
$TTL 172800     ; 2天
movie.edu               IN SIG  KEY 1 2 172800 20010104060917 (
                               20001205060917 27791 movie.edu.
                               RyNYoZ/k0tHqnFhUiVs2yjJWPFNeP8BKZ/Jaw+7xO9Jl
                               ZwJN2ZYQjVNVGLk30rJlxQRjCCdaaYQSq8u81up3xw== )
                        IN SIG  KEY 1 2 172800 20010104060917 (
                               20001205060917 47703 movie.edu.
                               1JGNBQydq6U+qKfq1wxfu1nsu283Zf7mNDDmuBtuuB7o
                               lwaeBL96tzBKpMUAcDYXsM8zxiStF+wTY+I5wfgevA== )
                        KEY     256 3 1 (
                               AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX
                               7+uBxB11BqL7LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B )
                        KEY     256 3 1 (
                               AQPjAfGtDkx6PSgYFs6G2vY1bJEldAMudngn6ZpGJnyg
                               k+LeU5PYiYd48wFLimihjyzlTWMb+C+egtQGpDVQulez )
```

注意，SIG记录中的一个记录是用标记为27791的密钥（旧私钥）产生的，而另一个记录是用标记为47703的密钥（新私钥）产生的。这证明我们有两个相应的私钥。

一旦我们从父区的管理者处得到了应答，我们将其保存到*/var/named/movie.edu.signedkey*，该文件名被我们$INCLUDE到*db.movie.edu.signed*中。*movie.edu.signedkey*如下所示：

```
$ORIGIN .
$TTL 172800     ; 2天
movie.edu               IN SIG  KEY 1 2 172800 20010104060917 (
                               20001205060917 65398 edu.
                               qzvmuTVv9yGZf963ZuN2jxk8brEX/VP3sI5pOM/g2mU/
```

```
                                EPa57fyhHDNo7ny8Q2Su5vXnAIoxaaKAR8VmognQ7A== )
                        KEY      256 3 1 (
                                AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX
                                7+uBxB11BqL7LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B )
                        KEY      256 3 1 (
                                AQPjAfGtDkx6PSgYFs6G2vY1bJEldAMudngn6ZpGJnyg
                                k+LeU5PYiYd48wFLimihjyzlTWMb+C+egtQGpDVQulez )
```

*edu*的SIG记录包含两个KEY记录，因此我们能够使用任意一个或两个密钥来对我们的区数据签名。

然后，我们仅用新私钥对我们的区数据签名：

```
#   dnssec-signzone -o movie.edu. db.movie.edu Kmovie.edu.+001+47703.private
```

*dnssec-signzone*并不对已用另一密钥签名的区重新签名，因此我们从*movie.edu*的未签名版本开始。下面是签名的区数据文件（*db.movie.edu.signed*）的摘录：

```
$ORIGIN .
$TTL 86400      ; 1天
movie.edu               IN SOA  terminator.movie.edu. al.robocop.movie.edu. (
                               2000092603 ; 序列号
                               10800      ; 刷新(3小时)
                               3600       ; 重试(1小时)
                               604800     ; 期满(1周)
                               3600       ; 最小值(1小时)
                                 )
                        SIG     SOA 1 2 86400 20010104062430 (
                                20001205062430 47703 movie.edu.
                                LIsndGD5q2VPWb+Ha0ffFP54UE6RYPweqtTp1xhgw4B9
                                Pyb/7z54J8q8LC0NmzQ6SthnfecBQhDBpc72HfNeJQ== )
                        NS      terminator.movie.edu.
                        NS      wormhole.movie.edu.
                        NS      outland.fx.movie.edu.
                        SIG     NS 1 2 86400 20010104062430 (
                                20001205062430 47703 movie.edu.
                                Ktq2mYMzTrBfGjdSb2F7ghyh2nXaLc0iTPV4k8I64jl0
                                nJt/hsBZPpeyM2u+Zymvp3mJMWg66E4tirj0AvlGXw== )
                        SIG     MX 1 2 86400 20010104062430 (
                                20001205062430 47703 movie.edu.
                                20001205062430 47703 movie.edu.
                                l/XnJ+JwhmAZLp6YF27YQQl0yT7iZ0qGDXPw860P6U1H
                                NmgDkUKoHfD6CdYwpKz15NyxRKilVmx2ne3oB0TUEQ== )
$TTL 3600       ; 1小时
                        SIG     NXT 1 2 3600 20010104062430 (
                                20001205062430 47703 movie.edu.
                                2sxN3rQXn/JklugmyGV+on1Io6tV1wEYP6m4oD1xHCP1
                                +NHPR+uT2IknW8SvGc3Kaj16kb2Ej+i3RvleWSI4Tg== )
                        NXT     bigt.movie.edu. ( NS SOA MX SIG KEY NXT )
```

```
$TTL 172800     ; 2天
                        SIG     KEY 1 2 172800 20010104060917 (
                                20001205060917 65398 edu.
                                qzvmuTVv9yGZf963ZuN2jxk8brEX/VP3sI5pOM/g2mU/
                                EPa57fyhHDNo7ny8Q2Su5vXnAIoxaaKAR8VmognQ7A== )
                        KEY      256 3 1 (
                                AQPdWbrGbVv1eDhNgRhpJMPonJfA3reyEo82ekwRnjbX
                                7+uBxB11BqL7LAB7/C+eb0vCtI53FwMhkkNkTmA6bI8B )
                        KEY      256 3 1 (
                                AQPjAfGtDkx6PSgYFs6G2vY1bJEldAMudngn6ZpGJnyg
                                k+LeU5PYiYd48wFLimihjyzlTWMb+C+egtQGpDVQulez )
$TTL 86400      ; 1天
                        MX      10 postmanrings2x.movie.edu.
```

尽管该区包括两个KEY记录和*edu*的SIG记录，且*edu*的SIG记录包含两种记录，但是区中的其他记录仅用标记为47703的新私钥签名。

当删除旧的KEY记录时，我们需要第二个签名的密钥集；这时，SIG记录*edu*发送给我们包含两个KEY记录的响应是毫无用处的。如果我们使用仅包含新KEY记录的密钥集就对了。

我们猜测在阅读这些以后，你也许决定使用可利用的最长密钥来避免对密钥的更改。

总结

我们认识到DNSSEC有一点令人畏缩。（在首次见到DNSSEC时，我们几乎昏厥。）但DNSSEC是设计用来做一些非常重要的事情的：使DNS具有欺骗性越来越难。并且当人们在Internet上进行越来越多的商业活动时，知道你确实到达了你所想去的地方将变得极其重要。

也就是说，我们认识到我们在本章描述的DNSSEC和其他安全措施并非全部对你有用。（当然它们并非是你必须全部做到的。）你应当在你对安全的要求和实现代价之间寻求一种平衡，即施加到你的基础设施和生产力上的负担。

本章内容：

- nslookup 是一个好工具吗？
- 交互式与非交互式
- 选项设置
- 避免搜索列表
- 常见的任务
- 不太常见的任务
- nslookup 的故障诊断与排除
- 网络中的无名英雄
- 使用 dig

第十二章

nslookup 和 dig

“不要那样自言自语地唠叨不休。”
矮胖子一见到她就说道，
“告诉我你的名字，是干什么的。”
“我叫爱丽丝，可是 ..... ”
“那真是一个愚蠢十足的名字，”矮胖子
不耐烦地打断道，“那是什么意思？”
“名字必须有什么意义吗？”爱丽丝怀疑地问。
“当然，”矮胖子一笑，说道 .....

为了熟练地对名字服务器的问题排错，你需要一个你能完全控制的排错工具来发送 DNS 查询。本章我们将讨论 *nslookup* 工具，它总是与 BIND 以及许多厂商的操作系统一起发布的。然而，这并不意味着它就是最好的DNS排错工具。实际上，*nslookup* 有许多的缺陷。虽然这些缺陷在 BIND 9 中已经修正了，但是由于它们已经为人们所熟知，我们还是要讨论一下。在本章我们还将讨论 *dig*（挖掘），它具有和 *nslookup* 相似的功能，却没有 *nslookup* 的那些缺陷。

注意，本章不打算涉及 *nslookup* 和 *dig* 的所有方面；*nslookup* 和 *dig* 的某些方面（大多是晦涩且很少使用的）我们将不讨论。对那些未涉及的部分，你可以随时查阅相关的手册。

nslookup 是一个好工具吗？

尽管有时你会用*nslookup*来查询其他名字服务器，就像名字服务器所做的那样，但在大多数时候你会按解析器的方式使用*nslookup*发送查询。你使用何种方式取决于你试图排除的故障。你或许想知道，“*nslookup*能非常精确地模拟解析器或名字服务器吗？*nslookup*是否调用BIND解析器的库例程？”答案是否定的，*nslookup*使用自己的例程来查询名字服务器，但这些例程是基于解析器的例程的。因此，*nslookup*的行为与解析器的行为十分相似，但有稍许差别。我们将指出两者的部分不同之处。至于模拟名字服务器的行为，*nslookup*允许我们使用与名字服务器相同的查询信息来查询另一服务器，但重传（retransmission）机制是很不同的。尽管与一台名字服务器一样，*nslookup*可以下载区数据的拷贝。所以，*nslookup*并非完全模拟解析器或名字服务器，但是它确实模拟得非常好，以至于成为一个好的故障诊断与排除工具。让我们来深入讨论前面所提到的不同之处。

多服务器

*nslookup*一次仅与一个名字服务器会话。这是*nslookup*与解析器（resolver）在行为表现方式上的最大区别。解析器使用在文件*resolv.conf*中的每一个*nameserver*配置指令。如果在文件*resolv.conf*中有两个*nameserver*配置指令，解析器将首先尝试与第一个名字服务器会话，然后是第二个服务器，再后来是第一个、第二个，这样下去直至它收到一个响应或它放弃查询。解析器对每一个查询均采用这种方式来进行。另一种方式是，*nslookup*将首先不断地尝试与在文件*resolv.conf*中配置指令的第一个名字服务器会话，直到放弃，然后才会尝试与第二个名字服务器会话。一旦它收到一个响应，它将锁定到那台服务器上，并不再尝试其他的服务器。但是，你希望排除工具仅与一个服务器会话，这样在分析故障时，就可以减少许多不确定的因素。如果*nslookup*使用多个服务器，你就难以控制排错会话。所以，仅与一个名字服务器会话正是排除工具所应具有的特性。

超时

当解析器只查询一个名字服务器时，*nslookup*的超时时间与解析器的超时时间是一致的。然而，名字服务器的超时时间是动态计算的，它的值取决于远程服务器响应上次查询的速度。*nslookup*的超时时间与名字服务器的超时时间永远不会一致，但

这并不会成为问题。当你使用*nslookup*查询远程名字服务器时，你可能只关心响应的内容是什么，而不关心响应时间的长短。

搜索列表

正如解析器程序那样，*nslookup*实现了搜索列表。BIND 4.9以前版本上的*nslookup*往往使用“全”搜索列表：包括本地域名和至少含有两个标号的所有祖先域名。BIND 4.9及其后续版本的名字服务器上的*nslookup*使用删减的搜索列表：仅包括本地域名。为了不使你迷惑，我们后面将会讲到如何确定你的*nslookup*的类型。

由于名字服务器未实现搜索列表功能，因此为了模拟名字服务器，必须关闭*nslookup*的搜索列表功能——更多的讨论将在稍后继续。

区传送

*nslookup*像名字服务器一样进行区传送。所不同的是，在下载区数据之前，*nslookup*不检查SOA序列号。如果需要，你只能用手工的方式来完成。

使用NIS和/etc/hosts

这最后一点不是比较*nslookup*和解析器或名字服务器的，而是在一般意义上讨论名字查找的方式。*nslookup*作为Internet软件协会发布的软件，它只使用DNS，而不使用NIS或*/etc/hosts*。大多数应用系统根据系统的不同配置使用DNS、NIS或*/etc/hosts*。除非你的主机确实被配置为使用名字服务器，否则不要指望*nslookup*来帮助你发现名字查找中的问题（注1）。

交互式与非交互式

让我们的*nslookup*教程从如何启动和退出*nslookup*开始吧。*nslookup*可以以交互式或非交互式的方式运行。如果你只想查一个域名的某个记录，请使用非交互式的方

注1： 或者你的供应商的*nslookup*（如，HP-UX）已经具备查询NIS服务器和检查*/etc/host*文件的能力。

式。如果你计划做更多的事，如更改服务器或更改选项设置，那么就使用一个交互式会话。

开始一次交互式会话，只需键入 *nslookup*:

```
% nslookup
Default Server:  terminator.movie.edu
Address:  0.0.0.0

> ^D
```

如果需要帮助，键入？或 *help*（注 2）。当你想退出时，键入 ^*D*（control-D）或 exit。如果要以中断 *nslookup* 的方式退出，十之八九可以按 ^*C* 键（或其他任何中断字符）。*nslookup* 会捕获中断，停止它正在做的工作并回到 > 提示符。

对非交互式查询，把你要查的名字包括在命令行内：

```
% nslookup carrie
Server:  terminator.movie.edu
Address:  0.0.0.0

Name:    carrie.movie.edu
Address: 192.253.253.4
```

选项设置

nslookup 有它自己的一组拨号盘和按钮，称为选项设置。所有选项设置均可更改。这里，我们将讨论每一个选项的含义。我们将在本章的以下部分告诉你如何使用它们。

```
% nslookup
Default Server:  bladerunner.fx.movie.edu
Address:  0.0.0.0

> set all
Default Server:  bladerunner.fx.movie.edu
Address:  0.0.0.0

Set options:
  nodebug         defname         search          recurse
```

注 2： BIND 9（如，9.1.0）中的 *nslookup* 不支持帮助功能。

```
    nod2            novc            noignoretc      port=53
    querytype=A     class=IN        timeout=5       retry=4
    root=a.root-servers.net.
    domain=fx.movie.edu
    srchlist=fx.movie.edu

> ^D
```

在讨论选项之前，我们首先介绍一些基本的信息。默认的名字服务器是*bladerunner.fx.movie.edu*。这意味着除非指定其他的名字服务器，*nslookup*将总是把每一个查询都发往 *bladerunner*。地址 0.0.0.0 表示“ 本台主机 ”。当 *nslookup* 使用 0.0.0.0 或 127.0.0.1 作为它的名字服务器时，就意味着正在使用运行在本机上的名字服务器 —— 在本例中就是 *bladerunner*。

选项分为两大类：布尔型选项和值型选项。选项名之后没有等号的选项是布尔型选项。这类选项只有“ 打开 ”和“ 关闭 ”两种属性。而值型选项则可以有不同的值。我们怎么知道哪些布尔型选项是打开的，而哪些是关闭的呢？当选项名前有“ no ”时，该选项是关闭的。*nodebug* 即表示调试是关闭的。正如你可能猜到的那样，选项 search 是打开的。

如何更改布尔型选项或值型选项，取决于你运行*nslookup*的方式。在交互式会话中，通过 set 命令来更改选项设置，如 *set debug* 或 *set domain=classics.movie.edu*。在命令行方式下，略去字 set 并在选项名前加一短线来完成选项设置的更改，如 *nslookup -debug* 或 *nslookup -domain=classics.movie.edu*。选项名可缩写成其最短的惟一字串 —— 即 *nodebug* 可缩写成 *nodeb*。除了缩写形式，*querytype* 选项也可简称为 *type*。

让我们来逐一讨论每一选项：

[no]debug

调试选项在默认情况下是关闭的。如果把它打开，名字服务器就会显示超时时间和响应包。关于 2 级调试的讨论，见*[no]d2* 选项。

[no]defname

在默认情况下，*nslookup*会把本地域名加到那些没有圆点符的名字中去。在搜索列表出现之前，BIND解析器程序仅把本地域名加到那些没有圆点符的名字中去，这一选项反映了这一行为方式。*nslookup*既能实现搜索列表出现前的行

为方式（将 *search* 选项关闭，并打开 *defname* 选项），也可实现搜索列表的行为方式（将 *search* 选项打开）。

[no]search

搜索选项将取代本地域名（*defname*）选项。即，只有当 *search* 选项被关闭的情况下，*defname* 选项才会起作用。默认情况下，*nslookup* 将搜索列表（srchlist）中的域名添加到那些不以圆点符结尾的名字之后。

[no]recurse

nslookup 在默认情况下发送递归查询。这一选项将置位查询消息中的递归位。BIND 解析器以同样的方式发出递归查询。然而，名字服务器在向其他名字服务器查询时，它发出的却是非递归查询。

[no]d2

2 级调试选项默认是关闭的。如果将它打开，除了会看到在常规调试下的输出外，你还会看到发送出去的查询消息。当打开 *d2* 选项时，也同时打开了 *debug* 选项。当关闭 *d2* 时，却仅关闭 *d2*，*debug* 仍为打开状态。当关闭 *debug* 时，则将 *debug* 和 *d2* 都关闭。

[no]vc

默认情况下，*nslookup* 使用 UDP 数据报，而不是通过虚电路（TCP）来发送查询。由于大多数 BIND 解析器使用 UDP 来查询，因此默认情况下 *nslookup* 的行为方式与解析器的行为方式是相一致的。就像解析器能被设置成使用 TCP 一样，*nslookup* 也可以被设置成使用 TCP。

[no]ignoretc

默认情况下，*nslookup* 不忽略被截断的消息。如果收到一个消息，这个消息的“截断”位被置位（表示名字服务器不能在一个 UDP 响应包中包括所有的重要信息），*nslookup* 将不会忽略它，它会使用 TCP 连接而不是 UDP 重新查询。在这一点上，又与 BIND 解析器相同。使用 TCP 连接重新查询的原因是，TCP 响应的大小可以是 UDP 响应的许多倍。

port=53

名字服务器在端口 53 上监听。你可以在另一端口上启动名字服务器（如，为了调试的目的），并且能够指定 *nslookup* 使用那个端口。

querytype=A

默认情况下，*nslookup* 查找 A（地址）资源记录类型。此外，如果你输入的是 IP 地址（且 *nslookup* 的查询类型是 A（地址）或 PTR（指针）），那么 *nslookup* 就会将地址反转，在尾部添加上 *in-add.arpa*，然后查找 PTR 记录。

class=IN

一般只有一种类：Internet（IN）。然而，如果你是一个 MITer 或运行 Ultrix，也存在 Hesiod（HS）类。

timeout=5

如果名字服务器在 5 秒钟内没有响应，*nslookup* 就会重发查询请求并将超时时间加倍（加到 10、20，然后是 40 秒）。大部分 BIND 解析器在查询同一服务器时使用相同的超时时间。

retry=4

在尝试 4 次之后才放弃。每重发一次查询请求，超时时间加倍。这一点与 BIND 解析器相同。

root=a.root-servers.net

有一条称为 *root* 的方便命令，*root* 命令将你的默认服务器切换到这里所给出的服务器。在新版 *nslookup* 的提示符下执行 *root* 命令等价于执行 *server a.root-servers.net* 命令。较老的版本使用 *nic.ddn.mil*（较老的）甚至 *sri-nic.arpa*（更老的）作为默认根名字服务器。你可以用 *set root=server* 命令来更改默认的“根”名字服务器。

domain=fx.movie.edu

如果 *defname* 选项是打开的，这就是将要添加的默认域名。

srchlist=fx.movie.edu

如果 *search* 选项是打开的，这些就是被添加到那些不以圆点结尾的名字尾部的域名。这些域名以其被尝试的先后顺序列出，域名之间用斜杠分隔。（BIND 4.8.3 的 *nslookup* 的搜索列表默认为 *fx.movie.edu/movie.edu*。对 BIND 4.9 及其后续版本 *nslookup* 的默认的搜索列表仅仅包括默认域名（注 3），你不得不

注 3： 这里给出一个非常简单就可以确定你所运行的 *nslookup* 版本的方法：键入 *set all* 并且检查默认的搜索列表是否仅仅包括本地域名（BIND 4.9 及其后续版本）或者是还包括祖先域名（BIND 4.8.3 或者更早版本）。

在 */etc/resolv.conf* 中明确地给出搜索列表设置以得到 *fx.movie.edu* 和 *movie.edu*）。

.nslookuprc 文件

你可以在 *.nslookuprc* 文件中设置新的 *nslookup* 选项值。当启动 *nslookup* 时，不管是以交互式还是以非交互式启动，*nslookup* 都会到你的主目录下查找 *.nslookuprc* 文件。*.nslookuprc* 文件可以包含任何合法的 set 命令，一行一条命令。这是很有用的，例如你的旧版 *nslookup* 仍然认为 *sri-nic.arpa* 是根名字服务器时。你可以在 *.nslookuprc* 文件中包含下面的一行来设置默认根名字服务器为真正的根名字服务器：

```
set root=a.root-servers.net
```

你也可以用 *.nslookuprc* 文件来设置搜索列表与你的主机默认搜索列表不同，或者更改 *nslookup* 的超时时间设置。

避免搜索列表

像解析器那样，*nslookup* 实现同样的搜索列表。当你在调试时，搜索列表会妨碍你的工作。因此，你要么把搜索列表完全关闭（*set nosearch*），要么就在要查询的全称域名后面加上一个圆点。就如你在例子中所看到的那样，我们更喜欢后一种方法。

常见的任务

如果你几乎每天都要使用，*nslookup* 就会显得稍微有点繁杂：查出给定域名的 IP 地址或 MX 记录，或查询特定名字服务器的数据。在讨论一般不常见的任务之前，我们首先讨论这些常见任务。

查找不同的记录类型

默认情况下，*nslookup* 为域名查找对应的地址，或为地址查找对应的域名。像在这个例子中所显示的那样，你通过改变 querytype（查询类型）来查询任何记录类型：

```
% nslookup
Default Server:  terminator.movie.edu
Address:  0.0.0.0

> misery                          —查询地址
Server:  terminator.movie.edu
Address:  0.0.0.0

Name:    misery.movie.edu
Address:  192.253.253.2

> 192.253.253.2                   —查询域名
Server:  terminator.movie.edu
Address:  0.0.0.0

Name:    misery.movie.edu
Address:  192.253.253.2

> set q=mx                        —查询 MX 记录
> wormhole
Server:  terminator.movie.edu
Address:  0.0.0.0

wormhole.movie.edu      preference = 10, mail exchanger = wormhole.movie.edu
wormhole.movie.edu      internet address = 192.249.249.1
wormhole.movie.edu      internet address = 192.253.253.1

> set q=any                       —查询任意类型的记录
> diehard
Server:  terminator.movie.edu
Address:  0.0.0.0

diehard.movie.edu       internet address = 192.249.249.4
diehard.movie.edu       preference = 10, mail exchanger = diehard.movie.edu
diehard.movie.edu       internet address = 192.249.249.4
```

当然，DNS 只有几种合法的记录类型。要得到更多列表，参见附录一。

权威的与非权威的响应

如果你以前使用过 *nslookup*，你或许已经注意到某些奇特的现象：在第一次查询远程名字服务器时，其响应是权威的，但当你第二次再查同一个名字服务器时，其响应却是非权威的。这里有一个例子：

```
% nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2
```

```
>    slate.mines.colorado.edu.
Server:  relay.hp.com
Address:  15.255.152.2

Name:     slate.mines.colorado.edu
Address:  138.67.1.3

>    slate.mines.colorado.edu.
Server:  relay.hp.com
Address:  15.255.152.2

Non-authoritative answer:
Name:    slate.mines.colorado.edu
Address:  138.67.1.3
```

这看上去有些奇怪，其实不然。原来是本地的名字服务器在第一次查询*slate.mines.colorado.edu*时，它与名为*mines.colorado.edu*的名字服务器联系，该服务器以一个权威的应答响应之。本地名字服务器将其收到的权威响应直接转给*nslookup*。同时，它也将该响应缓存起来。当第二次再查*slate.mines.colorado.edu*时，本地名字服务器用它缓存中的数据来响应该查询，从而导致“非权威”响应（注 4）。

注意，我们每次在查询域名时都要在该域名的尾部加上一个标记圆点。如果不加标记圆点，其响应也会是一样的。当进行调试时，是否加上标记圆点是很关键的，但有时却无关紧要。如果已知该名字是一个全称域名，我们总是将其加上一个标记圆点而不去推敲它是否需要，当然，也有例外，如在搜索列表关闭的情况下。

切换名字服务器

有时你想直接查询另一个服务器 —— 例如，你认为它工作异常。使用*nslookup*，你可以通过*server*或*lserver*命令来切换服务器。这两条命令的不同之处在于，*lserver*命令是查询“本地”服务器（你用来开始工作的那台机器）来获得你想切换的服务器地址的，而*server*命令则使用默认服务器而不是本地服务器。了解这一差别很重要，因为你刚刚切换到的服务器可能不会有响应，如下面这个例子：

```
%  nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2
```

注 4： 有趣的是，BIND9 名字服务器甚至将第一个响应也显示为非权威的。

开始时，*relay.hp.com*，我们的第一个名字服务器成为本地服务器。这将在本次会话的后期才有意义。

```
>   server galt.cs.purdue.edu.
Default Server:  galt.cs.purdue.edu
Address:  128.10.2.39

>   cs.purdue.edu.
Server:  galt.cs.purdue.edu
Address:  128.10.2.39

*** galt.cs.purdue.edu can't find cs.purdue.edu.: No response from server
```

这时，我们试着切回到最初的名字服务器。但在*galt.cs.purdue.edu*上却没有正在运行的名字服务器可用来查出 *relay.hp.com* 的地址。

```
>   server relay.hp.com.

*** Can't find address for server relay.hp.com.: No response from server
```

为了解决这一难题，我们使用*lserver*命令让本地服务器来查出*relay.hp.com*的地址：

```
>   lserver relay.hp.com.
Default Server:  relay.hp.com
Address:  15.255.152.2

> ^D
```

由于在 *galt.cs.purdue.edu* 上的名字服务器没有响应（它甚至没有名字服务器在运行），它不可能查出 *relay.hp.com* 的地址以便切回使用 *relay* 的名字服务器。幸好*lserver*提供了补救措施：本地名字服务器*relay*仍然在工作，所以我们利用了它。除了 *lserver*，我们还可以通过直接使用 *relay* 的 IP 地址来恢复 —— 服务器 *15.255.152.2*.

你甚至可以每做一次查询切换一次服务器。为了指定你希望*nslookup*查询特定的服务器来获得给定域名的信息，你可以在要查的域名之后，在命令的第二个参数中给出那台服务器，像这样：

```
%  nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2

>   saturn.sun.com. ns.sun.com.
```

```
Name Server:  ns.sun.com
Address:  192.9.9.3

Name:    saturn.sun.com
Addresses: 192.9.25.2

> ^D
```

当然，你也可在命令行方式下更改服务器。你可以在要查的域名之后作为参数来指定要查询的服务器，就像这样：

```
%   nslookup -type=mx fisherking.movie.edu. terminator.movie.edu.
```

这条命令告诉 *nslookup* 到 *terminator.movie.edu* 上去查找 *fisherking.movie.edu.* 的 MX 记录。

最后，为了指定一个可选的默认服务器并进入交互式模式，你可以在要查域名的位置上加一个短线：

```
%   nslookup - terminator.movie.edu.
```

不太常见的任务

让我们继续讨论一些你平常可能很少使用的技巧，但它们是很容易掌握的。当你在诊断和排除DNS或BIND的故障时，这些技巧大部分都是非常有用的，他们能够使你截获并查看解析器发出和收到的消息，也能够模拟BIND名字服务器查询另一个服务器或传送区数据。

显示查询消息和响应消息

如果需要，你可以让 *nslookup* 显示出它所发出的查询消息和它所接收到的响应消息。打开 *debug* 选项就会显示响应消息。打开 *d2* 选项则不仅会显示响应消息而且还显示查询消息。当你想完全关闭调试时，就不得不设置 *nodebug* 选项，因为设置 *nod2* 选项仅仅关闭 2 级调试。在下面的跟踪信息之后，我们将解释部分输出内容。如果愿意，你可以拿出你的 RFC 1035 文档，翻到第 25 页，一边阅读文档一边看我们的解释。

```
%  nslookup
```

```
Default Server:  terminator.movie.edu
Address:  0.0.0.0

>  set debug
>  wormhole
Server:  terminator.movie.edu
Address:  0.0.0.0

------------
Got answer:
    HEADER:
        opcode = QUERY, id = 6813, rcode = NOERROR
        header flags:  response, auth. answer, want recursion,
        recursion avail.  questions = 1,  answers = 2,
        authority records = 2,  additional = 3

    QUESTIONS:
        wormhole.movie.edu, type = A, class = IN
    ANSWERS:
    ->  wormhole.movie.edu
        internet address = 192.253.253.1
        ttl = 86400 (1D)
    ->  wormhole.movie.edu
        internet address = 192.249.249.1
        ttl = 86400 (1D)
    AUTHORITY RECORDS:
    ->  movie.edu
        nameserver = terminator.movie.edu
        ttl = 86400 (1D)
    ->  movie.edu
        nameserver = wormhole.movie.edu
        ttl = 86400 (1D)
    ADDITIONAL RECORDS:
    ->  terminator.movie.edu
        internet address = 192.249.249.3
        ttl = 86400 (1D)
    ->  wormhole.movie.edu
        internet address = 192.253.253.1
        ttl = 86400 (1D)
    ->  wormhole.movie.edu
        internet address = 192.249.249.1
        ttl = 86400 (1D)

------------
Name:    wormhole.movie.edu
Addresses:  192.253.253.1, 192.249.249.1

>  set d2
>  wormhole
Server:  terminator.movie.edu
Address:  0.0.0.0
```

这一次查询也显示出来了。

```
------------
SendRequest(  ), len 36
    HEADER:
        opcode = QUERY, id = 6814, rcode = NOERROR
        header flags:  query, want recursion
        questions = 1,  answers = 0,  authority records = 0,
      additional = 0

    QUESTIONS:
        wormhole.movie.edu, type = A, class = IN
------------
------------
Got answer (164 bytes):
```

响应与前面的相同。

在虚线之间的文字是查询和响应消息。正如所承诺的那样，我们将讨论消息的内容。DNS 包由五部分组成：首部（Header）、问题（Question）、回答（Answer）、权威（Authority）和附加（Additional）。

首部段（Header section）

首部段出现在每一个查询和响应消息中。*nslookup* 报告的操作码（opcode）总是 QUERY。其他的操作码只有区变更异步通知（NOTIFY）和动态更新（UPDATE），但 *nslookup* 并不对这两个操作码进行识别，因为它们只是发送常规的查询和接收响应。

首部中的 ID 是用来联系查询和响应并检测重复查询和重复响应的。你必须通过查看首部标志（header flag）来确定哪些是查询消息以及哪些是响应消息。字串 want recursion（要递归）意味着这是一个递归查询。同样的标志也出现在响应中。字串 auth.answer（权威回答）表示该回答是权威的。换句话说，就是该回答来自名字服务器的权威数据而不是它的缓存。响应码 *rcode*，可以是 no error（没有错误）、server failure（服务器失败）、*name error*（也称为 *nxdomain* 或 *nonexistent domain*，即名字错误或域名不存在）、not implemented（未实现）或 refused（被拒绝）之一。响应码 *server failure*、*name error*、*not implemented* 和 *refused* 分别导致 *nslookup* 的“服务器失败”、“域名不存在”、“未实现”和“被拒绝”错误。首部段中的最后四项是计数器 —— 它们指明在下面四段中的每一段有多少条资源记录。

问题段（Question section）

在一个 DNS 消息中总是会有一个问题，这个问题包括域名、要查询的数据类

型和资源类。它从不会有多于一个的问题。要让一个DNS 消息能够处理多于一个的问题，需要重新设计消息的格式。例如，只有一位的权威位就必须要修改，因为在回答段中可能会同时包含权威回答和非权威回答。在目前的设计中，设置权威回答位表示对于问题段中的域名这台名字服务器是权威的。

回答段（Answer section）
: 这段包含回答问题的资源记录。在响应中可能会有多于一个的资源记录。例如，如果主机是多宿主机，那么就会有多于一个的地址资源记录。

权威段（Authority section）
: 权威段是返回名字服务器记录的地方。当一个响应表示查询者是别的名字服务器时，那些名字服务器就被列于此处。

附加段（Additional section）
: 附加记录段中的信息是对其他段中的信息的补充。例如，一个名字服务器若被列于权威段中，那么这台服务器的地址就被加到附加段中。毕竟，要与那台服务器联系，你得知道它的地址。

对于那些坚持要了解细节的人，存在一种情况，DNS消息的查询问题数目不是一个：在逆向查询中，问题数目是零。逆向查询时，在查询信息中有一个回答，同时问题段为空。名字服务器将会把问题填上。但是，正如我们所说的那样，逆向查询几乎不存在。

像 BIND 名字服务器那样查询

你可以让*nslookup*发出与名字服务器所发出的一样的查询消息。名字服务器的查询消息与解析器的消息没有多少区别。主要的区别是解析器要求递归查询而名字服务器却很少这么要求。默认情况下，*nslookup*要求递归查询，因此你必须显式地将它关闭。在操作上，解析器和名字服务器之间的差别在于解析器使用搜索列表，而名字服务器却不。默认情况下，*nslookup*使用搜索列表，因此你也得把它关闭。当然，合理地使用标记圆点会得到同样的效果。

*nslookup*最早的意思是你使用*nslookup*的默认设置像解析器那样来查询。要像名字服务器那样查询，需使用*set norecurse*和*set nosearch*命令。在命令行下，那就是*nslookup -norecurse -nosearch*。

当 BIND 名字服务器收到一个查询请求时，它在它的权威数据和缓存中寻找回答。如果没有找到回答且它是这个区的权威名字服务器，那么它就以名字不存在或没有那种类型的记录来响应。如果该名字服务器没有回答而且它不是这个区的权威服务器，它就会在名字空间中向上查找 NS 记录。在名字空间中较高的位置上总会有 NS 记录。作为最后手段，它会使用根区的 NS 记录，即最高层。

如果名字服务器收到一个非递归查询请求，它会用它所发现的 NS 记录来响应查询者。另一方面，如果最初的原始查询是递归查询，那么名字服务器就会查询在它所发现的NS记录中的远程名字服务器。当名字服务器收到来自远程服务器的响应时，它将把这一响应缓存起来，如果必要并重复这一过程。在远程服务器的响应中要么包含问题的答案，要么包含接近答案的名字服务器列表。

例如，假定我们正在设法服务一个递归查询且在未检查*gov*区之前均未查到任何NS记录。在本例中，实际上就是当我们询问在 *relay.hp.com* 上的名字服务器关于*www.whitehouse.gov*的信息时 —— 它未发现任何 NS 记录直至检查 *gov* 区。从那里我们把服务器切换到*gov*名字服务器并问相同的问题。它会引导我们到*whitehouse.gov*的名字服务器。然后，我们切换到 *whitehouse.gov* 名字服务器并问相同的问题：

```
%  nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2

> set norec                    －像名字服务器那样查询：关闭递归
> set nosearch                 －关闭搜索列表
> www.whitehouse.gov     －我们不用圆点结尾，因为我们关闭了搜索列表
Server:  relay.hp.com
Address:  15.255.152.2
Name:    www.whitehouse.gov
Served by:
- I.ROOT-SERVERS.NET
          192.36.148.17
          gov
- E.ROOT-SERVERS.NET
          192.203.230.10
          gov
- D.ROOT-SERVERS.NET
          128.8.10.90
          gov
- B.ROOT-SERVERS.NET
          128.9.0.107
          gov
- C.ROOT-SERVERS.NET
          192.33.4.12
```

```
          gov
- A.ROOT-SERVERS.NET
          198.41.0.4
          gov
- H.ROOT-SERVERS.NET
          128.63.2.53
          gov
- G.ROOT-SERVERS.NET
          192.112.36.4
          gov
- F.ROOT-SERVERS.NET
          192.5.5.241
          gov
```

切换到*gov*名字服务器。(如果你的名字服务器没有已经缓存的*gov*名字服务器的地址，你或许不得不暂时把递归打开。)

```
>   server e.root-servers.net
Default Server:  e.root-servers.net
Address: 192.203.230.10
```

向*gov*名字服务器询问同样的问题。它将把我们引导到接近我们所需答案的名字服务器。

```
>  www.whitehouse.gov.
Server:  e.root-servers.net
Address:  192.203.230.10

Name:    www.whitehouse.gov
Served by:
- DNSAUTH1.SYS.GTEI.NET

          whitehouse.gov
- DNSAUTH2.SYS.GTEI.NET

          whitehouse.gov
- DNSAUTH3.SYS.GTEI.NET

          whitehouse.gov
```

切换到 *whitehouse.gov* 名字服务器 —— 两者之中任何一个均可。

```
>   server dnsauth2.sys.gtei.net.
Default Server:  dnsauth2.sys.gtei.net
Address:  4.2.49.3

>  www.whitehouse.gov.
Server:  sec1.dns.psi.net
Address:  38.8.92.2
```

```
Name:    www.whitehouse.gov
Addresses:  198.137.240.91, 198.137.240.92
```

希望本例能给你一个名字服务器是如何查询域名的印象。如果你需要以图形方式来重新理解这一过程，请翻回到图 2-12 和 2-13。

在我们继续之前，请注意，我们询问每一个名字服务器同一个问题：“ *www.whitehouse.gov* 的地址是什么？ ”如果在 *gov* 名字服务器的缓存中已经有了 *www.whitehouse.gov* 的地址，你认为将会发生什么事呢？*gov* 名字服务器将会用它缓存中的数据来回答你的问题，而不是引导你去 *whitehouse.gov* 服务器。这一点为何重要呢？在你的区数据中，假如你弄错了某台主机的地址。某人指出你的错误，并且你也将该错误改正了。尽管现在你的名字服务器有了正确的数据，但远程主机在查询该主机的域名时得到的却是旧的错误数据。在名字空间中一个处于较高层次的名字服务器（如，根名字服务器），已经缓存了那个不正确的数据，当根服务器接到关于那台主机地址的查询请求时，它将返回不正确的数据而不是将查询者引导到你的名字服务器上来。由于只有一个处于“ 较高 ”层次的名字服务器缓存了不正确的数据就使得这一问题难以跟踪，因为只有部分远程的查询会得到错误的答案 —— 那些使用那台服务器的查询。有趣吧，嗯？尽管那台处于“ 较高 ”层次的名字服务器最终会使那一旧数据超时。如果你时间紧迫，你可以与那台远程服务器的管理员联系并要求他们杀掉并重起 *named* 进程以刷新缓存。当然，如果那台远程服务器是一个重要的、使用频繁的名字服务器，他们或许会告诉你该怎么做。

区传送

nslookup 可以通过使用 *ls* 命令来传送完整的区数据。这一特性对于排除故障、找出远端主机域名如何拼写、或者只是计算某一远程域中有多少台主机是很有用的。由于数据量可能会很大，*nslookup* 允许你将输出结果重定向到一个文件中。如果你想中途放弃，你可以敲中断字符中断传送。

注意：某些主机或是出于安全因素，或是出于限制服务器的工作负载的原因，不允许你下载它们的区数据拷贝。Internet是一个友好的地方，但是管理员不得不保护他们的领地。

让我们来看看 *movie.edu* 区。就像你将在下面的输出中看到的那样，所有的区数据均被列出来了 —— SOA 记录被列出两次，这是由区数据传送过程中的数据交换方

式引起的。由于一些*nslookup*默认条件下只能列出地址和名字服务器记录，我们可以使用 *—d* 选项获取整个区：

```
% nslookup
Default Server:  terminator.movie.edu
Address:  0.0.0.0

> ls -d movie.edu.
[terminator.movie.edu]
$ORIGIN movie.edu.
@                   1D IN SOA      terminator al.robocop (
                                   2000091400      ; 序列号
                                   3H              ; 刷新
                                   1H              ; 重试
                                   4W2D            ; 期满
                                   1H )            ; 最小值

                    1D IN NS       terminator
                    1D IN NS       wormhole
wormhole            1D IN A        192.249.249.1
                    1D IN A        192.253.253.1
wh249               1D IN A        192.249.249.1
robocop             1D IN A        192.249.249.2
bigt                1D IN CNAME    terminator
cujo                1D IN TXT      "Location:" "machine" "room" "dog" "house"
wh253               1D IN A        192.253.253.1
wh                  1D IN CNAME    wormhole
shining             1D IN A        192.253.253.3
terminator          1D IN A        192.249.249.3
localhost           1D IN A        127.0.0.1
fx                  1D IN NS       bladerunner.fx
bladerunner.fx      1D IN A        192.253.254.2
fx                  1D IN NS       outland.fx
outland.fx          1D IN A        192.253.254.3
fx                  1D IN NS       huskymo.boulder.acmebw.com.
                    1D IN NS       tornado.acmebw.com.
dh                  1D IN CNAME    diehard
carrie              1D IN A        192.253.253.4
diehard             1D IN A        192.249.249.4
misery              1D IN A        192.253.253.2
@                   1D IN SOA      terminator al.robocop (
                                   2000091400      ; 序列号
                                   3H              ; 刷新
                                   1H              ; 重试
                                   4W2D            ; 期满
                                   1H )            ; 最小值
```

现在让我们来看区数据开始的一个记录，由于屏幕滚动而使你不能看到它，*nslookup* 允许你把区数据存在文件中：

```
>  ls -d movie.edu  > /tmp/movie.edu      —将所列数据存到文件 /tmp/movie.edu 中
[terminator.movie.edu]
Received 25 answers (25 records).
```

一些版本的*nslookup*甚至支持内置的*view*命令，该命令可以在交互方式下分类并显示区列表的内容。然而，在 BIND 8 版本中删除了 *view* 命令，并且 BIND 9（如，BIND.9.1.0）的 *nslookup* 也不支持 *view*。

nslookup 的故障诊断与排除

最后你想要知道的就是，如果你的排错工具出了问题该怎么办？不幸的是，某些错误使得排错工具几乎毫无用处。多数情况下，某些类型的*nslookup*错误是很让人迷惑的，因为它们没有给出任何清楚的信息帮助你处理。尽管*nslookup*自身会有一些问题，但大多数你所碰到的问题是名字服务器的配置和操作问题。我们将在这里讨论这些问题。

查找正确的数据

这不是一个真正的问题，但它却可让人十分费解。如果你用*nslookup*来查一个域名的一种类型的记录，且这个域名存在但没有你要查的那种类型的记录，你将得到如下错误：

```
%  nslookup
Default Server:  terminator.movie.edu
Address:  0.0.0.0

>  movie.edu.

*** No address (A) records available for movie.edu.
```

所以，到底存在哪些记录类型呢？只需设置 *type=any* 就可查出：

```
>  set type=any
>  movie.edu.
Server:  terminator.movie.edu
Address:  0.0.0.0

movie.edu
      origin = terminator.movie.edu
      mail addr = al.robocop.movie.edu
```

```
        serial = 42
        refresh = 10800 (3H)
        retry   = 3600 (1H)
        expire  = 604800 (7D)
        minimum ttl = 86400 (1D)
movie.edu    nameserver = terminator.movie.edu
movie.edu    nameserver = wormhole.movie.edu
movie.edu    nameserver = zardoz.movie.edu
movie.edu    preference = 10, mail exchanger = postmanrings2x.movie.edu
postmanrings2x.movie.edu        internet address = 192.249.249.66
```

服务器没有响应

如果你的服务器不能查出它自己的名字，到底什么地方可能出了错呢？

```
% nslookup
Default Server:  terminator.movie.edu
Address:  0.0.0.0

> terminator
Server:  terminator.movie.edu
Address:  0.0.0.0

*** terminator.movie.edu can't find terminator: No response from server
```

错误消息“服务器没有响应”的准确意思是：解析器没有收到返回来的响应。当它启动时，*nslookup* 没有必要查看任何信息。如果你看到你的服务器地址是 0.0.0.0，*nslookup* 就用这台名字服务器所在主机的主机名（*hostname* 命令所返回的名字）作为 *Default Server* 字段的值，并给出它的提示符。只有当你做查询时，你才发现没有服务器响应。本例中，这是相当明显的，即没有名字服务器正在运行 —— 因为一个名字服务器应该能够查出它自己的名字。然而，如果你是在查询某些远时信息，名字服务器可能失败而不会给你响应，因为在它还在查询数据时，*nslookup* 却放弃了等待。你如何知道服务器不在运行，以及服务器在运行但未能响应这两者的区别呢？使用 *ls* 命令来指出其不同：

```
% nslookup
Default Server:  terminator.movie.edu
Address:  0.0.0.0

> ls foo.       —试图列出一个不存在的区
*** Can't list domain foo.: No response from server
```

在本例中，没有名字服务器在运行。如果主机不可达到，则错误将是“超时”。如果名字服务器正在运行，你将会看到如下的错误消息：

```
% nslookup
Default Server: terminator.movie.edu
Address: 0.0.0.0

> ls foo.
[terminator.movie.edu]
*** Can't list domain foo.: No information
```

那就是说，除非在你的世界中存在一个顶级 *foo* 域。

没有对应名字服务器地址的 PTR 记录

这是一个最让人头疼的问题之一：某处出错了，*nslookup* 一启动就退出：

```
% nslookup

*** Can't find server name for address 192.249.249.3: Non-existent host/domain
*** Default servers are not available
```

“域不存在”的意思是名字 *3.249.249.192.in-addr.arpa* 不存在。换句话说，就是 *nslookup* 不能映射 192.249.249.3（名字服务器的地址）到它所对应的域名。可是，我们不是才说过 *nslookup* 在启动时不查任何东西码？在未配置好之前，*nslookup* 是不会做任何查询的，但那并不是规则。如果你创建了一个包含一个或者多个 *nameserver* 行的 *resolv.conf* 文件，*nslookup* 就会试图反向映射地址以获取名字服务器的域名。在前面的例子中，有一个运行在 192.249.249.3 上的名字服务器，但它却说没有与地址 192.249.249.3 相对应的 PTR 记录。显然，反向映射区被弄乱了，至少 *249.249.192.in-addr.arpa* 区的数据被弄乱了。

在本例中，“默认服务器不可用”消息使人容易误解。毕竟，有一个名字服务器在那里却说地址不存在。通常情况下，如果那台主机上的名字服务器没有运行或那台主机不可达，你会看到“服务器上没有响应”的错误。只有那时，“默认服务器不可用”消息才有意义。

查询被拒绝

查询被拒绝可能会在 *nslookup* 启动时带来问题，而且它们可能会导致某个会话中的查询错误。这里是由于查询被拒绝而引起 *nslookup* 在启动时退出的情况：

```
% nslookup
```

```
*** Can't find server name for address 192.249.249.3: Query refused
*** Default servers are not available
%
```

这一错误有两种可能的原因。或是你的名字服务器不支持逆向查询（仅限于较老的*nslookup*版本），或是访问列表终止了这一查询。

*nslookup*的旧版本（早于4.8.3版）在启动时使用逆向查询。逆向查询从来没有被广泛地使用过 —— *nslookup*是少数几个使用它的应用程序之一。在BIND 4.9版，对逆向查询的支持被删除。为适应这些老版本客户，增加了一条配置文件选项。

在 BIND 4 中，这条语句如下所示：

```
options fake-iquery
```

在 BIND 8 中，这条语句如下所示：

```
options { fake-iquery yes; };
```

（BIND 9，如 BIND 9.1.0, 不支持 *fake-iquery*。）

这条语句使你的名字服务器用一个“伪造的”应答来响应逆向查询，这一响应对*nslookup*来说已足够好，从而使它可继续执行下去（注 5）。

访问列表也可能导致*nslookup*的启动错误。当*nslookup*企图查找它的名字服务器的域名时（用 PTR 查询，而非逆向查询），查询可能被拒绝。如果你认为是访问列表的问题，确保你允许正在运行的主机可以查询名字服务器。如果*nslookup*与名字服务器在同一主机上运行，你需要检查所有*secure_zone* TXT记录或者用*allow_query*子语句检查本地主机的 IP 地址或者回送地址。

访问列表不仅仅引起*nslookup*的启动错误。当你将*nslookup*指向一个远程名字服务器时，它也可能在会话过程中导致查询和区传送失败。这就是你将要看到的：

```
%  nslookup
Default Server:  hp.com
Address:  15.255.152.4
```

注 5： 对一个逆向查询的伪造响应，也就是说，拥有地址 192.249.249.3 的域名就是方括号中的地址：[192.249.249.3]。

```
>   server terminator.movie.edu
Default Server:  terminator.movie.edu
Address:  192.249.249.3

>   carrie.movie.edu.
Server:  terminator.movie.edu
Address:  192.249.249.3

*** terminator.movie.edu can't find carrie.movie.edu.: Query refused

> ls movie.edu                          —这条命令试图进行一次区数据传送
[terminator.movie.edu]
*** Can't list domain movie.edu: Query refused
>
```

resolv.conf 文件中的第一个名字服务器没有响应

这是上一个问题的另一种表现形式：

```
%  nslookup
*** Can't find server name for address 192.249.249.3: No response from server
Default Server:  wormhole.movie.edu
Address:  192.249.249.1
```

这一次，在*resolv.conf*文件中的第一个名字服务器没有响应。我们在*resolv.conf*文件中加入了第二个*nameserver*指令，且第二个名字服务器确实响应了。从现在开始，*nslookup* 只向 *wormhole.movie.edu* 发出查询请求，而不再尝试与 192.249.249.3 的名字服务器联系。

找出正在查找的是什么

在前一个例子中，我们曾提到*nslookup*查询名字服务器的地址但未证实这一点。这里是我们的证明。这一次在启动 *nslookup* 时，我们从命令行上打开 *d2* 调试。这将引起 *nslookup* 输出它发出的查询信息，同时也输出何时查询超时，以及何时重传：

```
%  nslookup -d2
------------
SendRequest( ), len 44
    HEADER:
        opcode = QUERY, id = 1, rcode = NOERROR
        header flags:  query, want recursion
        questions = 1,  answers = 0,  authority records = 0,
        additional = 0
```

```
        QUESTIONS:
            3.249.249.192.in-addr.arpa, type = PTR, class = IN

    ------------
    timeout (5 secs)
    timeout (10 secs)
    timeout (20 secs)
    timeout (40 secs)
    SendRequest failed

    *** Can't find server name for address 192.249.249.3: No response from server
    *** Default servers are not available
```

正如你所看到的超时时间，*nslookup*经过75秒才放弃。如果没有调试输出，你就不可能在那75秒内的屏幕上看到任何输出，就好像*nslookup*已经挂起了似的。

未指定的错误

你可能会遇到一个更令人不安的问题，称为“未指定的错误”。这里有一个这类错误的例子。我们只包括了输出的最后部分，因为我们这时只想讨论这一错误。（你将在第十四章中找到其完整的*nslookup*会话输出。）

```
Authoritative answers can be found from:
(root)  nameserver = NS.NIC.DDN.MIL
(root)  nameserver = B.ROOT-SERVERS.NET
(root)  nameserver = E.ROOT-SERVERS.NET
(root)  nameserver = D.ROOT-SERVERS.NET
(root)  nameserver = F.ROOT-SERVERS.NET
(root)  nameserver = C.ROOT-SERVERS.NET
(root)  nameserver =
*** Error: record size incorrect (1050690 != 65519)

*** relay.hp.com can't find .: Unspecified error
```

这里所出现的问题就是，数据太多而无法装进一个UDP数据报中。当数据报耗尽空间时，名字服务器停止了数据的填充。名字服务器在其响应包中没有设置截断位，或者是*nslookup*已经使用TCP连接重新查询了，而名字服务器一定是认为已经包含了足够的“重要”信息。你不会经常看到这类错误。当你给这个域创建太多的NS记录时你才会看到它，因此不要创建太多的NS记录。（像这样的建议使你明白为什么要购买这本书了，是这样吧？）多少才算是太多取决于在该包中的名字的“压缩”程度，反过来，也就是取决于有多少名字服务器在它们的域名中共享同一个域。出于这一原因，多个根名字服务器被重新命名整合在一个*root-servers.net*域中——

这就允许有更多的根名字服务器（十三个）来支持整个Internet。为了了解是什么引起了这一错误，你得阅读第十四章。对那些刚刚读了第九章的读者来说，或许你已经知道了这一点。

网络中的无名英雄

系统管理员的工作是一种吃力不讨好的工作。老有那么些问题，通常是一些很简单的问题，得一遍又一遍地回答。有时，他们则以一种创造性的聪明方式来帮助用户巧妙地解决问题。当我们中有人发现他们的智慧时，我们只有坐在他们身后，羡慕地微笑着，并且责怪自己为什么事先没有想出来。这样的例子很多，比如，系统管理员是这样终止有时让人感到困惑的 *nslookup* 会话的：

```
% nslookup
Default Server:  envy.ugcs.caltech.edu
Address:  131.215.134.135

> quit
Server:  envy.ugcs.caltech.edu
Addresses:  131.215.134.135, 131.215.128.135

Name:    ugcs.caltech.edu
Addresses:  131.215.128.135, 131.215.134.135
Aliases:  quit.ugcs.caltech.edu
          use.exit.to.leave.nslookup.-.-.-.ugcs.caltech.edu

> exit
%
```

使用 dig

一种方法是处理 *nslookup* 的缺点；另一种是放弃 *nslookup*，并使用 *dig*（Domain Information Groper，域名信息探索者）。

早先我们说过 *dig* 不如 *nslookup* 应用得广泛，所以我们最好从何处可以获得它开始。你可以从 BIND 4 发布的 *tools* 目录，BIND 8 发布的 *src/bin/dig* 目录，BIND 9 发布的 *bin/dig* 目录中获取 *dig* 的源代码。如果你编译了整个发布，你也可以获得一个非常好的、新的 *dig* 备份。

用 *dig*，你可以在命令行方式下指定查询的任何方面；*dig* 没有交互方式。你可以指定参数为你想要查询的域名，指定另一个参数为你想要发送查询的类型（比如，*a* 为地址记录，*mx* 为 MX 记录）；默认情况为查询地址记录。你可以在 @ 后面指定你想要查询的名字服务器。你可以使用域名或者 IP 地址来表示一个名字服务器。默认查询的名字服务器在 *resolv.conf* 中。

对于参数，*dig* 是很聪明的。你可以以任何你喜欢的方式指定你的参数顺序，并且 *dig* 可以辨认出 *mx* 可能是指记录的类型，而不是你想要查询的域名（注 6）。

nslookup 和 *dig* 的一个主要区别是 *dig* 不使用查询列表，因此经常使用全称域名作为参数。例如：

```
%  dig plan9.fx.movie.edu
```

是使用 *resolv.conf* 中的第一个名字服务器查询 *plan9.fx.movie.edu* 的地址记录，而：

```
%  dig acmebw.com mx
```

在相同的名字服务器上查询 *acmebw.com* 的 MX 记录，而：

```
%  dig @wormhole.movie.edu. movie.edu. soa
```

在 *wormhole.movie.edu* 服务器上查询 *movie.edu* 中的 SOA 记录。

dig 的输出格式

最引以为荣的是，*dig* 以各种不同的段（首部段、问题段、回答段、权威段、附加段）很清楚地显示完整的 DNS 响应消息，每个段中的资源记录以主文件格式输出。如果你想使用你的排错工具的输出（输出可以是区数据文件或你的根线索文件的内容）的一部分，这也是非常方便的。例如，下面语句：

```
%  dig @a.root-servers.net ns .
```

输出结果为：

注 6： 实际上，*dig* 的 BIND9 早期版本（9.1.0 之前）的排序能力很弱，并要求你在类型之前指定域名参数。你可以将服务器指定为在任何地方查询。

```
; <<>> DiG 8.3 <<>> @a.root-servers.net . ns
; (1 server found)
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 6
;; flags: qr aa rd; QUERY: 1, ANSWER: 13, AUTHORITY: 0, ADDITIONAL: 13
;; QUERY SECTION:
;;      ., type = NS, class = IN

;; ANSWER SECTION:
.                       6D IN NS        A.ROOT-SERVERS.NET.
.                       6D IN NS        H.ROOT-SERVERS.NET.
.                       6D IN NS        C.ROOT-SERVERS.NET.
.                       6D IN NS        G.ROOT-SERVERS.NET.
.                       6D IN NS        F.ROOT-SERVERS.NET.
.                       6D IN NS        B.ROOT-SERVERS.NET.
.                       6D IN NS        J.ROOT-SERVERS.NET.
.                       6D IN NS        K.ROOT-SERVERS.NET.
.                       6D IN NS        L.ROOT-SERVERS.NET.
.                       6D IN NS        M.ROOT-SERVERS.NET.
.                       6D IN NS        I.ROOT-SERVERS.NET.
.                       6D IN NS        E.ROOT-SERVERS.NET.
.                       6D IN NS        D.ROOT-SERVERS.NET.

;; ADDITIONAL SECTION:
A.ROOT-SERVERS.NET.     6D IN A         198.41.0.4
H.ROOT-SERVERS.NET.     6D IN A         128.63.2.53
C.ROOT-SERVERS.NET.     6D IN A         192.33.4.12
G.ROOT-SERVERS.NET.     6D IN A         192.112.36.4
F.ROOT-SERVERS.NET.     6D IN A         192.5.5.241
B.ROOT-SERVERS.NET.     6D IN A         128.9.0.107
J.ROOT-SERVERS.NET.     5w6d16h IN A    198.41.0.10
K.ROOT-SERVERS.NET.     5w6d16h IN A    193.0.14.129
L.ROOT-SERVERS.NET.     5w6d16h IN A    198.32.64.12
M.ROOT-SERVERS.NET.     5w6d16h IN A    202.12.27.33
I.ROOT-SERVERS.NET.     6D IN A         192.36.148.17
E.ROOT-SERVERS.NET.     6D IN A         192.203.230.10
D.ROOT-SERVERS.NET.     6D IN A         128.8.10.90

;; Total query time: 116 msec
;; FROM: terminator.movie.edu to SERVER: a.root-servers.net  198.41.0.4
;; WHEN: Fri Sep 15 09:47:26 2000
;; MSG SIZE  sent: 17  rcvd: 436
```

让我们按段来检查这个输出结果。

第一行，以主文件注释提示符（;）和 <<>>*DIG 8.3*<<>> 开始，简单的模仿在命令行中指定的选项，也就是，我们对 *a.root-servers.net* 根区中的 NS 记录感兴趣。

下一行（*1 server found*），告诉我们 *dig* 在查找我们在 @ 后面指定域名 *a.root-servers.net* 的对应地址的时候，发现了一个记录。（如果 *dig* 发现了多于三个的记录，它只会报告三个记录，因为大部分的解析器最多只能查询三个名字服务器。）

以 *->>HEADER<<-* 开头的行是 *dig* 从远程名字服务器接收到的回答信息的首部段的第一部分。首部中的操作码总是 QUERY，就像 *nslookup* 中的那样。状态是 NOERROR；它可以是本章前面一节中提到的任何状态。ID 是消息的 ID，通常用来匹配查询的响应。

标志（flag）告诉我们更多的响应信息。*qr* 表示消息是响应而不是查询。*dig* 只对响应解码而不对查询解码，所以一般都会显示 *qr*。然而，对 *aa* 和 *rd* 不是这样的，*aa* 表示响应是权威的，而 *rd* 表示设置在查询消息中的递归位（因为响应名字服务器简单地把查询消息中的该位拷贝到响应消息中）。大部分时间，*rd* 都设置在查询消息中，不过你在响应中也能看到 *ra*，这表明远程名字服务器支持递归查询。然而，*a.rootservers.net* 是根名字服务器不允许递归查询，如同我们在第十一章所讲的那样，它只是把递归查询当做重复查询处理。因此，远程名字服务器会忽略掉 *rd* 位并且以不设置 ra 位来表示递归不可行。

首部段的最后一个字段表明 *dig* 问了一个问题，并且在回答段中得到 13 个记录，0 个记录在权威段，13 个记录在附加数据段。

下一行后面包含 *QUERY SECTION:* 显示 *dig* 发送的查询信息：根区中 IN 类的 NS 记录。后面是 *ANSWER SECTION:*，我们可以看到 13 个根名字服务器的 NS 记录。接下来是 *ADDITION SECTION:*，包括了对应于 13 个 NS 记录的 13 个 A 记录。如果响应包括权威段，我们也会看到 *AUTHORITY SECTION:*。

最后，*dig* 显示查询和响应的汇总（summary）信息。第一行显示 *dig* 发出查询后并从远程服务器得到响应的时间。第二行显示发送查询的主机和查询被发往的名字服务器。第三行是收到响应的时间戳。最后一行，第四行表示查询和响应消息的大小（以字节计算）。

利用 dig 进行区传送

像使用 *nslookup* 那样，我们也可以用 *dig* 执行区传送。不同于 *nslookup* 的是 *dig* 没

有专门的命令来请求区传送。你只要简单地指定 *axfr*（作为查询类型）和用区的域名作为参数就可以了。记住，你只能从区的权威名字服务器传送区。

如果你从 *wormhole.movie.edu* 传送 *movie.edu* 区，可以像下面这样：

```
$ dig @wormhole.movie.edu movie.edu axfr

; <<>> DiG 8.3 <<>> @wormhole.movie.edu movie.edu axfr
; (1 server found)
$ORIGIN movie.edu.
@                   1D IN SOA      terminator al.robocop (
                                   2000091402      ; 序列号
                                   3H              ; 刷新
                                   1H              ; 重试
                                   1W              ; 期满
                                   1H )            ; 最小值

                    1D IN NS       terminator
                    1D IN NS       wormhole
                    1D IN NS       outland.fx
outland.fx          1D IN A        192.253.254.3
wormhole            1D IN A        192.249.249.1
                    1D IN A        192.253.253.1
wh249               1D IN A        192.249.249.1
robocop             1D IN A        192.249.249.2
bigt                1D IN CNAME    terminator
cujo                1D IN TXT      "Location:" "machine" "room" "dog" "house"
wh253               1D IN A        192.253.253.1
wh                  1D IN CNAME    wormhole
shining             1D IN A        192.253.253.3
terminator          1D IN A        192.249.249.3
localhost           1D IN A        127.0.0.1
fx                  1D IN NS       bladerunner.fx
bladerunner.fx      1D IN A        192.253.254.2
fx                  1D IN NS       outland.fx
outland.fx          1D IN A        192.253.254.3
dh                  1D IN CNAME    diehard
carrie              1D IN A        192.253.253.4
diehard             1D IN A        192.249.249.4
misery              1D IN A        192.253.253.2
@                   1D IN SOA      terminator al.robocop (
                                   2000091402      ; 序列号
                                   3H              ; 刷新
                                   1H              ; 重试
                                   1W              ; 期满
                                   1H )            ; 最小值

;; Received 25 answers (25 records).
;; FROM: terminator.movie.edu to SERVER: wormhole.movie.edu
;; WHEN: Fri Sep 22 11:02:45 2000
```

注意，和 *nslookup* 一样，SOA 记录出现两次，分别在区的开始和结束。对于所有的 *dig* 输出，区传送的结果以主文件的格式输出，所以，如果需要，你可以把输出作为一个区数据文件（注 7）。

dig 选项

需要给出的 *dig* 的命令行选项太多，可以参照 *dig* 使用手册中的详细列表。这里只是列出一些比较重要的，并给出它们的意义：

−x address

nslookup 足够聪明以至于能辨认一个 IP 地址，并在 *in-addr.arpa* 中查找合适的域名，为什么 *dig* 不能呢？如果使用 *−x* 选项，*dig* 就假定你给出的域名是真实的 IP 地址，它就会反转 IP 地址并且附加 *in-addr.arpa*。利用 *dig* 还可以把默认记录类型改为 ANY，这样你就可以使用 *dig −x 10.0.0.1* 反向映射一个 IP 地址。

−p port

把查询发往指定端口而不是默认的 53 端口。

+norec[urse]

关闭递归查询（递归默认是打开）。

+vc

发送基于 TCP 的查询（查询默认使用 UDP）。

注 7： 你需要先删除额外的 SOA 记录。

第十三章

阅读 BIND 的调试输出

本章内容：

- 调试级别
- 打开调试
- 阅读调试输出
- 解析器搜索算法和否定缓存（BIND 8）
- 解析器搜索算法和否定缓存（BIND 9）
- 工具

“噢，卷丹！”爱丽丝向着随风微摆的
卷丹说道，“我希望能与你交谈！”
“我们可以交谈，”卷丹说道，
“当有值得一谈的人的时候。”

可以利用名字服务器的调试结果来进行诊断和排错。只要你的名字服务器在编译时定义了编译选项 DEBUG，你就可以得到一个个查询的内部操作报告。你所得到的消息通常是很难理解的，它们是给有源程序的人看的。本章中，我们将解释部分调试输出。我们的目标只是涵盖足够多的内容，让你了解名字服务器在做什么，而不是企图解释所有的调试消息。

当你在浏览本章中的解释时，请回想前面章节的内容。在另一上下文中再次看到这些信息，有助于你更完整地理解名字服务器是如何工作的。

调试级别

名字服务器提供的信息数量取决于调试级别的高低。调试级别越低，信息越少。较高的调试级别能给你更多的信息，但它们同时也会更快地填满你的硬盘。经过阅读大量的调试输出信息之后，你就会找到到底需要多少信息才可以解决某一特定故障的感觉。当然，如果你能很容易地重现故障，你可以从级别 1 开始，并逐步提高调

试级别直到你获得足够的信息。对于最基本的问题，例如，为什么找不到一个名字，级别 1 就足够了，因此你应从调试级别 1 开始。

每一个调试级别的信息是什么？

这里是 BIND 8 和 BIND 9 名字服务器中每一调试级别将会给出的信息列表。调试信息是累积的，例如，级别 2 包括级别 1 的调试信息。数据被划分为以下几种基本部分：启动、更新数据库、处理查询、和维护区数据。我们不打算讨论名字服务器的内部数据库的更新问题，因为问题总是出现在别的地方。然而，名字服务器从它的内部数据库中添加或删除哪些数据可能会是一个问题，就像你在第十四章中将会看到的那样。

BIND 8 和 9 共有 99 个调试级别，但是大多数调试消息都记录在少数几个调试级别上。这里我们将看到这部分调试级别。

BIND 8 的调试级别

级别 1

这一级别的信息相当简洁。名字服务器可以处理许多查询，这些查询会产生大量的调试输出。由于输出是压缩的，你可以长期收集数据。使用这一调试级别来获得基本的启动信息和观察查询事务。在这一级别，你可以看到一些记录的错误信息，包括语法错误和 DNS 数据报格式错误。这一级别也会显示引用（referral）。

级别 2

级别2 提供许多有用的东西 :它列出在一次查找中用到的远程名字服务器的IP地址，以及往返时间值；当它收到错误的响应时会显示报警信息，而且它会在其响应中打上标记来表示它所回答的是何种查询类型，也就是SYSTEM（系统查询）还是 USER（用户）查询。当你跟踪辅名字服务器的加载区错误时，这一级别将向你展示区数据值：序列号、刷新时间、重试时间、期满时间和剩余时间，因为辅名字服务器要检查它的主名字服务器以确定它的数据是否是最新数据。

级别 3

调试级别3的信息要详细得多，因为它将产生许多关于更新名字服务器的数据

库的信息。如果打算收集在级别3或更高级别的调试输出，你得确信有足够的磁盘空间。在级别3，你将会看到：发送出的重复查询、产生的系统查询（sysquery）、在查询时所使用的远程服务器的名字以及对每一个服务器所找到的地址数。

级别4

当你想看名字服务器收到的查询和响应包时，使用调试级别4。这一级别还显示缓存数据的可信度。

级别5

级别5有多种不同的消息，但对一般的调试，它们都不是特别有用。在这一级中包括一些错误消息，如，何时*malloc()*失败或者何时名字服务器放弃了一个查询。

级别6

级别6显示发给原始查询的响应。

级别7

级别7显示几个配置和分析消息。

级别8

在这一级别没有重要的调试信息。

级别9

本级没有重要的调试信息。

级别10

当你想看名字服务器所发出的查询和响应包时，使用调试级别10。这些包的格式与在级别4中的格式相同。你不必经常使用这一级别，因为你可以通过*nslookup*来查看名字服务器的查询和响应包。

级别11

本级以及本级以上只有几个调试消息，而且还很少涉及到。

BIND 9的调试级别

级别1

级别1给出基本的名字服务器操作：区加载、维护（包括，SOA记录、区传送、

区期满和缓存清除）、NOTIFY 消息、接收的查询、分配的高级别任务（比如，查询名字服务器的地址）。

级别 2

级别 2 记录多播（multicast）请求。

级别 3

级别 3 给出了低级别任务的创建和操作。不幸的是，这些任务大部分没有特别的描述名字（*requestmgr_detach*?）并且它们给出的参数也非常模糊。级别 3 也显示日志（journal）活动，例如，当名字服务器把区变更的记录写到区的日志中，或者在启动时名字服务器使一个日志对应一个区。DNSSEC 验证器（validator）的操作和 TSIG 签名的检查也在本级别中给出。

级别 4

当传输来的区的日志文件不可用时，主名字服务器将被迫使用 AXFR。级别 4 将记录这一事件。

级别 5

当服务一个特定的请求时，级别 5 将记录使用哪个视图（view）。

级别 6

级别 6 将记录对外区传送消息，包括检查启动传送的查询。

级别 7

该级别只记录新的调试信息（记录日志的添加或删减，区传送返回的字节数）。

级别 8

级别 8 记录许多动态更新信息：必备（prerequisite）检查、写日志条目、回滚（rollback）。一些低级别的区传送信息，包括区传送中发送的资源记录，也会在这里出现。

级别 10

级别 10 显示有关区计时器活动的一些信息。

级别 20

级别 20 报告一个更新给区刷新计时器。

级别 90

级别 90 记录 BIND 9 的任务分配器的低级别操作。

使用 BIND 8 和 BIND 9，你可以配置名字服务器在打印调试消息时附上调试级别。就像在第七章的“在 BIND 8 和 9 中记录日志”一节中所解释的那样，只需将记录日志选项 *print-severity* 打开即可。

请时刻牢记这是调试信息 —— 它是被 BIND 代码的开发者用来调试程序的，因此它不会像你所希望的那样容易阅读。你也可以用它来判断，为什么名字服务器没有做你认为它应该做的事，或者只是学习名字服务器是如何工作的，但不要指望它是经过很好地设计、精心地格式化了的输出。

打开调试

名字服务器的调试既可以从命令行开始，也可以用控制消息（control message）来开始。如果你需要看启动信息来诊断你的当前问题，你必须使用命令行选项。如果你想在一个正在运行的名字服务器上开始调试，或者你想关闭调试，就必须使用控制消息。名字服务器把它的调试信息写入文件 *named.run*。BIND 4 名字服务器将在 */usr/tmp*（或 */var/tmp*）目录下创建 *named.run* 文件。BIND 8 和 9 名字服务器将在它的工作目录下创建 *named.run* 文件。

调试命令行选项

在故障诊断和排错时，你有时需要查看分类列表，了解哪个接口与一个文件描述符相绑定，或找出当名字服务器在初始化阶段就退出时的位置（如果 *syslog* 错误消息不是足够清楚的话）。若要看到这类调试信息，你必须用命令行选项来启动调试，因为当你发控制消息时，已经太晚了。用于调试的命令行选项是 *–d* 级别。当使用命令行选项打开调试时，BIND 4 名字服务器将不会像通常情况那样在后台运行，你必须在命令行的最后加上“ & ”才能返回到你的命令行提示符。以下是如何启动处于调试级别 1 的 BIND 4 名字服务器：

```
#  /etc/named -d 1 &
```

当你设定选项 *–d* 的时候，BIND 8 和 9 名字服务器会在后台运行，所以你不必加“ & ”。

用控制消息来改变调试级别

如果你不需要查看名字服务器的启动信息，你可以以不带调试命令行选项的方式来启动你的名字服务器。以后，你可以用 *ndc* 向名字服务器进程发送相应的控制消息来打开和关闭调试。这里，我们将调试级别设置成级别 3，然后关闭调试。

```
#  ndc trace 3
#  ndc notrace
```

而且，正如你所希望的那样，如果你是从命令行打开的调试，你仍然可以用 *ndc* 改变名字服务器的调试级别。

然而，BIND 9.1.0 的 *rndc* 不支持 *trace* 和 *notrace* 参数（9.1.0 的 *named* 也不支持），但是将来的版本会支持的。所以，如果你运行的是 BIND 9，你应该使用 *–d* 命令行选项。

阅读调试输出

我们将讨论五个调试输出的例子。第一个例子演示名字服务器的启动。接下来的两个例子演示成功的名字查询。第四个例子演示辅名字服务器保持它的区数据的更新情况。在最后一个例子中，我们从演示名字服务器的行为转到演示解析器的行为：解析器搜索算法。除了最后一个例子外，在每一次跟踪之后，我们都重启名字服务器，因此，每一次跟踪均是从一个新的、缓存几乎为空的情况下开始的。

你或许想知道，我们为什么要选择名字服务器在正常情况下的行为作为我们的例子，毕竟本章是关于调试的。我们向你介绍名字服务器的正常行为是因为，在你跟踪非正常操作之前，你必须了解正常的操作是什么。另一个原因就是想帮助你理解前面章节中所描述的概念（重传、往返时间，等等）。

名字服务器的启动（BIND 8，调试级别 1）

我们将通过观察名字服务器的初始化来开始调试例子。第一个名字服务器是 BIND 8。我们在命令行中使用了 *–d 1* 选项，以下是在 *named.run* 中产生的输出：

```
1) Debug level 1
```

```
2) Version = named 8.2.3-T7B Mon Aug 21 19:21:21 MDT 2000
3) cricket@abugslife.movie.edu:/usr/local/src/bind-8.2.3-T7B/src/bin/named
4) conffile = ./named.conf
5) starting.  named 8.2.3-T7B Mon Aug 21 19:21:21 MDT 2000
6) cricket@abugslife.movie.edu:/usr/local/src/bind-8.2.3-T7B/src/bin/named
7) ns_init(./named.conf)
8) Adding 64 template zones
9) update_zone_info('0.0.127.in-addr.arpa', 1)
10) source = db.127.0.0
11) purge_zone(0.0.127.in-addr.arpa,1)
12) reloading zone
13) db_load(db.127.0.0, 0.0.127.in-addr.arpa, 1, Nil, Normal)
14) purge_zone(0.0.127.in-addr.arpa,1)
15) master zone "0.0.127.in-addr.arpa" (IN) loaded (serial 2000091500)
16) zone[1] type 1: '0.0.127.in-addr.arpa' z_time 0, z_refresh 0
17) update_zone_info('.', 3)
18) source = db.cache
19) reloading hint zone
20) db_load(db.cache, , 2, Nil, Normal)
21) purge_zone(,1)
22) hint zone "" (IN) loaded (serial 0)
23) zone[2] type 3: '.' z_time 0, z_refresh 0
24) update_pid_file(  )
25) getnetconf(generation 969052965)
26) getnetconf: considering lo [127.0.0.1]
27) ifp->addr [127.0.0.1].53 d_dfd 20
28) evSelectFD(ctx 0x80d8148, fd 20, mask 0x1, func 0x805e710, uap 0x40114344)
29) evSelectFD(ctx 0x80d8148, fd 21, mask 0x1, func 0x8089540, uap 0x4011b0e8)
30) listening on [127.0.0.1].53 (lo)
31) getnetconf: considering eth0 [192.249.249.3]
32) ifp->addr [192.249.249.3].53 d_dfd 22
33) evSelectFD(ctx 0x80d8148, fd 22, mask 0x1, func 0x805e710, uap 0x401143b0)
34) evSelectFD(ctx 0x80d8148, fd 23, mask 0x1, func 0x8089540, uap 0x4011b104)
35) listening on [206.168.194.122].53 (eth0)
36) fwd ds 5 addr [0.0.0.0].1085
37) Forwarding source address is [0.0.0.0].1085
38) evSelectFD(ctx 0x80d8148, fd 5, mask 0x1, func 0x805e710, uap 0)
39) evSetTimer(ctx 0x80d8148, func 0x807cbe8, uap 0x40116158, due 969052990.
812648000, inter 0.000000000)
40) exit ns_init(  )
41) update_pid_file(  )
42) Ready to answer queries.
43) prime_cache: priming = 0, root = 0
44) evSetTimer(ctx 0x80d8148, func 0x805bc30, uap 0, due 969052969.000000000,
inter 0.000000000)
45) sysquery: send -> [192.33.4.12].53 dfd=5 nsid=32211 id=0 retry=969052969
46) datagram from [192.33.4.12].53, fd 5, len 436
47) 13 root servers
```

我们在调试输出中加入了行号，在你的输出中将不会有行号。第 2 行到第 6 行是正在运行的 BIND 的版本信息和配置文件的名称。版本 8.2.3-T7B 是由 ISC（Internet

软件协会）于 2000 年 8 月发布的。在这次运行中，我们使用了当前目录下的 *./named.conf* 配置文件。

第 7 行至第 23 行显示了 BIND 读取配置文件和区数据文件。这台名字服务器是一台只缓存名字服务器 —— 所读的文件只有 *db.127.0.0*（9 行至 16 行）和 *db.cache*（17 行至 23 行）。第 9 行列出了被更新的区（*0.0.127.IN-ADDR.ARPA*），而第 10 行则显示包含区数据的文件（*db.127.0.0*）。第 11 行指明在加入任何新数据之前先将区中的旧数据删除。第 12 行说的是区数据被重新装入了，尽管区数据实际上是被首次装入。区数据在第 13 行到 15 行之间被装入。在第 16 行和第 23 行中，*z_time* 是检查本区何时更新的时间，*z_refresh* 是区刷新时间。这些值只有当名字服务器是区的辅名字服务器才有意义。

第 25 行至 39 行是文件描述符的初始化（在本例中，它们实际上是套接字描述符）。文件描述符 20 和 21（27 行至 29 行）被绑定到回送地址 127.0.0.1。描述符 20 是数据报套接字，而描述符 21 是数据流套接字。文件描述符 22 和 23（32 行至 34 行）被绑定到 IP 地址为 192.249.249.3 的网络接口。每一个接口地址都被考虑和使用 —— 如果接口没有被初始化或地址已存在于列表中，它们将不会被使用。文件描述符 5（36 行至 39 行）被绑定到通配地址 0.0.0.0。大多数网络守护进程只将一个套接字绑定到通配地址，而不是将多个套接字绑定到各个接口。通配地址接受送往该主机的任意接口的数据报。让我们暂时脱离主题，解释为什么 *named* 既要把一个套接字绑定到通配地址，同时也要把套接字绑定到特定的接口上。

当 *named* 收到一个来自某一应用程序或另一名字服务器的请求时，它将在一个绑定到特定接口的套接字上收到这一请求。如果 *named* 没有把套接字绑定到特定接口的话，它就会在已绑定到通配地址上的套接字上接收该请求。当 *named* 回发响应时，它使用请求到达的同一个套接字描述符。为什么 *named* 要这么做呢？当响应通过绑定到通配地址的套接字发送出去时，内核将会把该响应实际送出的接口的地址填入到发送者地址字段。该地址可能是，也可能不是那个请求所发往的地址。当响应通过绑定到特定地址的套接字发送出去时，内核将会用该特定地址来填入发送者地址，与该请求所发往的地址相同。如果名字服务器收到来自一个它所不知道的 IP 地址的响应时，该响应就会被打上一个“火星”（martian）标记并将其丢弃。*named* 通过将其响应发送到被绑定到特定接口的文件描述符上来设法避免火星响应，从而，响应中的发送者地址与相应请求所发往的地址一致。然而，当 *named* 发出查询请求时，它使用通配描述符，因为没有一定得使用一个特定 IP 地址的要求。

第43行至47行显示的是名字服务器发出一个系统查询以找出当前正在运行的根名字服务器。这被称为“预准备缓存”(priming the cache)。第一个被查询的服务器发回了一个包含十三个名字服务器的响应。

这时该名字服务器被初始化了，它已准备就绪随时响应查询。

名字服务器的启动（BIND 9，调试级别1）

下面是BIND 9名字服务器启动时的输出：

```
1)  Sep 15 15:34:53.878 starting BIND 9.1.0 -d1
2)  Sep 15 15:34:53.883 using 1 CPU
3)  Sep 15 15:34:53.899 loading configuration from './named.conf'
4)  Sep 15 15:34:53.920 the default for the 'auth-nxdomain' option is now 'no'
5)  Sep 15 15:34:54.141 no IPv6 interfaces found
6)  Sep 15 15:34:54.143 listening on IPv4 interface lo, 127.0.0.1#53
7)  Sep 15 15:34:54.151 listening on IPv4 interface eth0, 192.249.249.3#53
8)  Sep 15 15:34:54.163 command channel listening on 0.0.0.0#953
9)  Sep 15 15:34:54.180 now using logging configuration from config file
10) Sep 15 15:34:54.181 dns_zone_load: zone 0.0.127.in-addr.arpa/IN: start
11) Sep 15 15:34:54.188 dns_zone_load: zone 0.0.127.in-addr.arpa/IN: loaded
12) Sep 15 15:34:54.189 dns_zone_load: zone 0.0.127.in-addr.arpa/IN: dns_journal
_rollforward: no journal
13) Sep 15 15:34:54.190 dns_zone_maintenance: zone 0.0.127.in-addr.arpa/IN: enter
14) Sep 15 15:34:54.190 dns_zone_maintenance: zone version.bind/CHAOS: enter
15) Sep 15 15:34:54.190 running
```

比较BIND 9的调试输出和BIND8的调试输出，你可能首先注意到的不同之处是BIND 9的调试输出的精简。不过你要知道，BIND 8已经有大约三年的历史了，作者有足够的时间在代码中增加调试消息，但是BIND 9还仅仅是刚出道的新产品，所以不可能加入太多的调试消息。

你可能也注意到，BIND 9的每个调试消息都包括时间戳，这可以使我们更容易地把消息和现实中的事件联系起来。

第1行和第2行给出正在运行的BIND的版本信息和配置文件的名称。像前面例子中所讲的，我们使用当前目录下的*named.conf*文件。第3行告诉我们只用一个CPU——即主机只使用了一个处理器。

第4行给出*auth-nxdomain*子语句的默认值已经改变的简单警告(第十章中讲到过)。

第 5 行提醒我们的主机没有任何 IPv6 的网络接口；如果有的话，BIND 9 能够监听到对这些接口的查询。

第 6 到 7 行显示名字服务器监听在以下两个接口：*lo*，回送接口；*ethO*，以太网接口。与 BIND 8 不同的是，BIND 9 以 *address#port* 的格式显示地址和接口，而 BIND 8 的格式为 *[address].port*。第 8 行显示 *named* 监听来自默认端口（即 53 端口）上的控制消息。

10 行至 12 行显示名字服务器装载了 *0.0.127.in-addr.arpa* 区的数据。*start* 和 *loaded* 消息是不言自明的。*no journal* 消息表明当前没有日志。（在第十章中讲过，日志是名字服务器为区接收的动态更新的记录。）

最后，第 13 和 14 行显示名字服务器维护 *0.0.127.in-address.arpa* 和 *version.bind* 区。（*version.bind* 是一个内置的 CHAOSNET 区，CHAOSNET 区包含一个单个的 TXT 记录。）区维护是一个安排周期任务的过程，如针对辅名字服务器的 SOA 查询、存根区（stub zone）或 NOTIFY 消息。

成功的查找（BIND 8，调试级别 1）

假如你想观察名字服务器查找一个名字的情况，且你的名字服务器不是以调试方式启动的，那么就用 *udc* 打开调试，查找名字，然后再关闭调试，就像这样：

```
#   ndc trace 1
#    /etc/ping galt.cs.purdue.edu.
#   ndc notrace
```

我们这么做了，下面是产生的 *named.run* 文件结果：

```
datagram from [192.249.249.3].1162, fd 20, len 36

req: nlookup(galt.cs.purdue.edu) id 29574 type=1 class=1
req: missed 'galt.cs.purdue.edu' as '' (cname=0)
forw: forw -> [198.41.0.10].53 ds=4 nsid=40070 id=29574 2ms retry 4sec
datagram from [198.41.0.10].53, fd 4, len 343

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40070
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 9, ADDITIONAL: 9
;;              galt.cs.purdue.edu, type = A, class = IN
EDU.                        6D IN NS    A.ROOT-SERVERS.NET.
EDU.                        6D IN NS    H.ROOT-SERVERS.NET.
```

```
EDU.                        6D IN NS    B.ROOT-SERVERS.NET.
EDU.                        6D IN NS    C.ROOT-SERVERS.NET.
EDU.                        6D IN NS    D.ROOT-SERVERS.NET.
EDU.                        6D IN NS    E.ROOT-SERVERS.NET.
EDU.                        6D IN NS    I.ROOT-SERVERS.NET.
EDU.                        6D IN NS    F.ROOT-SERVERS.NET.
EDU.                        6D IN NS    G.ROOT-SERVERS.NET.

A.ROOT-SERVERS.NET.             5w6d16h IN A    198.41.0.4
H.ROOT-SERVERS.NET.             5w6d16h IN A    128.63.2.53
B.ROOT-SERVERS.NET.             5w6d16h IN A    128.9.0.107
C.ROOT-SERVERS.NET.             5w6d16h IN A    192.33.4.12
D.ROOT-SERVERS.NET.             5w6d16h IN A    128.8.10.90
E.ROOT-SERVERS.NET.             5w6d16h IN A    192.203.230.10
I.ROOT-SERVERS.NET.             5w6d16h IN A    192.36.148.17
F.ROOT-SERVERS.NET.             5w6d16h IN A    192.5.5.241
G.ROOT-SERVERS.NET.             5w6d16h IN A    192.112.36.4
resp: nlookup(galt.cs.purdue.edu) qtype=1
resp: found 'galt.cs.purdue.edu' as 'edu' (cname=0)
resp: forw -> [192.36.148.17].53 ds=4 nsid=40071 id=29574 1ms
datagram from [192.36.148.17].53, fd 4, len 202

;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40071
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 4
;;   galt.cs.purdue.edu, type = A, class = IN
PURDUE.EDU.                 2D IN NS    NS.PURDUE.EDU.
PURDUE.EDU.                 2D IN NS    MOE.RICE.EDU.
PURDUE.EDU.                 2D IN NS    PENDRAGON.CS.PURDUE.EDU.
PURDUE.EDU.                 2D IN NS    HARBOR.ECN.PURDUE.EDU.
NS.PURDUE.EDU.                  2D IN A     128.210.11.5
MOE.RICE.EDU.                   2D IN A     128.42.5.4
PENDRAGON.CS.PURDUE.EDU.            2D IN A  128.10.2.5
HARBOR.ECN.PURDUE.EDU.              2D IN A     128.46.199.76
resp: nlookup(galt.cs.purdue.edu) qtype=1
resp: found 'galt.cs.purdue.edu' as 'cs.purdue.edu' (cname=0)
resp: forw -> [128.46.199.76].53 ds=4 nsid=40072 id=29574 8ms
datagram from [128.46.199.76].53, fd 4, len 234

send_msg -> [192.249.249.3].1162 (UDP 20) id=29574
Debug off
```

首先，注意，被记录的是IP地址而不是名字，对一个名字服务器来说，你不认为有点奇怪吗？然而，这并不奇怪。如果你试图用查询名字的方法来排除故障，你不会仅仅为了产生更易阅读的调试输出而让名字服务器查询额外名字的 —— 额外的查询会干扰排错。没有一种调试级别将IP地址翻译成名字。你不得不使用工具（比如，稍后我们所提供的那种）来转换它们。

让我们一行一行地来逐一解释这一调试输出。如果你想理解每一行的含义，这种详

尽的方式是很重要的。如果你打开调试，可能是试图发现为什么某些名字不能被查到的原因，那么你将不得不去设法理解那些跟踪信息的含义。

```
datagram from [192.249.249.3].1162, fd 20, len 36
```

一个数据报来自 IP 地址为 192.249.249.3（*terminator.movie.edu*）的主机。如果发送者和名字服务器在同一台主机上，你可能会看到数据报来自 127.0.0.1。发送端应用程序使用端口 1162。名字服务器从文件描述符（fd）20 上接收数据报。如前所示，启动调试输出将告诉你文件描述符 20 被绑定到哪个接口上。这一数据报的长度（len）是 36 个字节。

```
req: nlookup(galt.cs.purdue.edu) id 29574 type=1 class=1
```

由于下一行以 *req* 开头，因此我们知道该数据报是请求数据报。在这一请求中，要查询的名字是 *galt.cs.purdue.edu*。请求 id 是 29574。*type=1* 的意思是该请求要查地址信息。*class=1* 表示类是 IN。你可以在头文件 */usr/include/arpa/nameser.h* 中找到查询类型和类的完整列表。

```
req: missed 'galt.cs.purdue.edu' as '' (cname=0)
```

名字服务器查找了所请求的名字但未找到。然后它试图找出一个它可以询问的远程名字服务器，但没有找到这样的服务器直到根区（空引号）。*cname=0* 的意思是名字服务器未碰到一个 CNAME 记录。如果确实看到了 CNAME 记录，规范名而不是原始名将被查找，同时 *cname* 将不为零。

```
forw: forw -> [198.41.0.10].53 ds=4 nsid=40070 id=29574 2ms retry 4sec
```

该查询被转发给在 198.41.0.10（*j.root-servers.net*）主机上的名字服务器（端口 53）。该名字服务器使用了文件描述符 4（它被绑定到通配地址）来发送查询。该名字服务器将该查询标上 ID 号 40070（*nsid=40070*），因而它能够将其响应匹配到初始问题。正如你在 *nslookup* 行中看到的那样，应用程序使用了 ID 号 29574（*id=29574*）。该名字服务器将在尝试下一名字服务器之前等待 4 秒钟。

```
datagram from [198.41.0.10].53, fd 4, len 343
```

在 *j.root-servers.net* 上的名字服务器有了响应。由于该响应是一个授权，在调试日志中它被完整打印出来了。

```
resp: nlookup(galt.cs.purdue.edu) qtype=1
```

当响应信息被缓存之后，该名字被再次查询。如前所述，*qtype=1* 的意思是名字服务器正在查找地址信息。

```
resp:found'galt.cs.purdue.edu'as'edu'(cname=0)
resp:forw→[192.36.148.17].53 ds=4 nsid=40071 id=29574/ms
datagram from[192.36.148.17].53,fd4,len202
```

根名字服务器以一个到 *edu* 域服务器的授权来响应。同样的查询请求被送往一个在 *edu* 域中的服务器 192.36.148.17 （*i.root-servers.net*）。*i.root-servers.net* 以关于 *purdue.edu* 名字服务器的信息来响应。

```
resp: found 'galt.cs.purdue.edu' as 'cs.purdue.edu' (cname=0)
```

这次在 *cs.purdue.edu* 层次上有了一些信息。

```
resp: forw -> [128.46.199.76].53 ds=4 nsid=40072 id=29574 8ms
```

一个查询请求被发往在 128.46.199.76 （*harbor.ecn.purdue.edu*）上的名字服务器。这时的名字服务器 ID 是 40072。

```
datagram from [128.46.199.76].53, fd 4, len 234
```

在 *harbor.ecn.purdue.edu* 上的服务器有了响应。我们得看看随后发生的事情才能理解这一响应的内容。

```
send_msg -> [192.249.249.3].1162 (UDP 20) id=29574
```

上一响应一定含有所请求的地址，因为名字服务器回应给应用程序（回头看看最初的查询，它使用端口 1162）。该响应是一个 UDP 包（相对于 TCP 连接），而且它使用了文件描述符 20。

当我们跟踪时，这台名字服务器是“ 安静的 ”，它没有在我们跟踪的同时处理其他的查询。当你在一个在线名字服务器上进行跟踪时，你就不会这么幸运了。你将不得不仔细地检查输出结果，并将那些你所关心的与你的查询有关的部分放到一起。然而这并非那样难。打开你所喜欢的编辑器，查找含有你所查找的名字的 *nslookup* 行，然后追踪那些 *nsid* 与该行的 *nsid* 相同的行。在下一个 BIND 8 跟踪中，你将会看到如何利用 *nsid*。

成功的查找（BIND 9，调试级别 1）

我们将显示处于级别 1 的 BIND 9 服务器在查找同一域名时的调试输出，但是，结果短得令人发笑。我们说过，在正确的操作下知道调试结果的含义是非常重要的。不管怎样，我们看：

```
Sep 16 17:20:57.193 client 192.249.249.3#1090: query: galt.cs.purdue.edu A
Sep 16 17:20:57.194 createfetch: galt.cs.purdue.edu. A
```

第 1 行告诉我们运行在 192.249.249.3（也就是本地主机）的 1090 端口上的一个客户为 *galt.cs.purdue.edu* 的地址发送了一个查询。第 2 行记录了名字服务器的名字解析部分，它让我们知道到底做了什么。

有重传的成功查找（BIND 8，调试级别 1）

并不是所有的查找都像前一个那样“ 利索 ” —— 有时查询必须被重传。只要是查询成功了，用户就看不出有何异同，尽管一个查询由于涉及到重传而需要更长的时间。下面是一个有重传的跟踪。在完成跟踪之后，我们把 IP 地址转换成对应的名字。注意，使用名字阅读起来要容易得多！

```
1)  Debug turned ON, Level 1
2)
3)  datagram from terminator.movie.edu port 3397, fd 20, len 35
4)  req: nlookup(ucunix.san.uc.edu) id 1 type=1 class=1
5)  req: found 'ucunix.san.uc.edu' as 'edu' (cname=0)
6)  forw: forw -> i.root-servers.net port 53  ds=4 nsid=2 id=1 0ms retry 4 sec
7)
8)  datagram from i.root-servers.net port 53, fd 4, len 240
    <delegation lines removed>
9)  resp: nlookup(ucunix.san.uc.edu) qtype=1
10) resp: found 'ucunix.san.uc.edu' as 'san.uc.edu' (cname=0)
11) resp: forw -> uceng.uc.edu port 53 ds=4 nsid=3 id=1 0ms
12) resend(addr=1 n=0) - > ucbeh.san.uc.edu port 53 ds=4 nsid=3 id=1 0ms
13)
14) datagram from terminator.movie.edu port 3397, fd 20, len 35
15) req: nlookup(ucunix.san.uc.edu) id 1 type=1 class=1
16) req: found 'ucunix.san.uc.edu' as 'san.uc.edu' (cname=0)
17) resend(addr=2 n=0) - > uccba.uc.edu port 53 ds=4 nsid=3 id=1 0ms
18) resend(addr=3 n=0) - > mail.cis.ohio-state.edu port 53 ds=4 nsid=3 id=1 0ms
19)
20) datagram from mail.cis.ohio-state.edu port 53, fd 4, len 51
21) send_msg -> terminator.movie.edu (UDP 20 3397) id=1
```

这一跟踪在开始部分与前一个跟踪相似（行 1 到行 11）：名字服务器收到一个对 *ucunix.san.uc.edu*的查询请求，发送该查询到一个*edu*的名字服务器（*i.root-servers.net*），收到一个包含*uc.edu* 的名字服务器列表，然后发送该查询到在*uc.edu*的其中一个名字服务器（*uceng.uc.edu*）。

这一跟踪的新功能就是*resend*行（行12、17和18）。在11行的*forw*就是*resend(addr=0 n=0)*的意思，CS 总是从零开始计数。由于 *uceng.uc.edu* 没有响应，名字服务器继续尝试 *ucbeh.san.uc.edu*（行 12）、*uccba.uc.edu*（行 17）和 *mail.cis.ohio-state.edu*（行 18）。在 *mail.cis.ohio-state.edu* 上的远程名字服务器终于有了响应（行 20）。注意，你可以通过查找*nsid=3*来找出所有的重传行，知道这一点是重要的，因为在这些行之间可能插入了许多其他的查询。

也请注意来自 *terminator.movie.edu* 的第二个数据报（行 14）。它与行 3 中的查询有相同的端口、文件描述符、长度、ID 和类型。应用程序未能及时地收到响应，因此它重传了其最初的查询。由于名字服务器还在处理第一个查询，因此这一个查询是重复的。在本输出中没有显示这一点，但名字服务器检测到重复并将其丢弃。我们可以这么说是因为并非像在行 4 至行 6 中那样：*req:* 行后有 *forw：*行。

如果名字服务器在查询名字时出现了故障，你能猜到输出会是什么样吗？你会看到许多重传，因为名字服务器不停地查询那一名字（你可以通过匹配 *nsid=* 行来找出它们）。你会看到应用程序发送大量的重传，因为它们认为名字服务器没有收到应用程序的初始查询。最终名字服务器会放弃，通常是在应用程序自己放弃之后。

使用 BIND 9.1.0 名字服务器，在级别 3 以前你不会看到重新发送，并且你也很难在 BIND 9 的其他日志信息中找到你想要的信息。并且，即使在级别 3，BIND 9.1.0 也不能告诉你正在重新发往哪个名字服务器。

检查区数据的辅名字服务器（BIND 8，调试级别 1）

除了用名字服务器的查询来跟踪问题外，你可能会不得不跟踪为什么辅名字服务器没有从它的主名字服务器上装载数据。跟踪这种问题的通常做法就是使用*nslook-up* 或 *dig*，正如我们在第十四章中所讲述的那样来比较在两个服务器上区的 SOA 序列号。如果你的问题更难以捉摸，或许你不得不去查看调试信息。我们将向你讲述名字服务器在正常工作时的调试输出信息会是什么样子。

这一调试输出产生于一个“安静的”名字服务器，它不接收任何查询，我们用它来向你准确地说明哪些行与区维护密切相关。你是否还记得在读入数据之前，BIND 4 或者 8 的辅名字服务器是用一个子进程来将区数据传送到本地磁盘的。在辅名字服务器将其调试信息记录到 *named.run* 文件的同时，辅名字服务器的子进程把它的调试信息写到 *xfer.ddt.PID* 文件。默认情况下，这里的文件名后缀 PID 是子进程的进程号，它可以改变以保证文件名的惟一性。当心 —— 打开辅名字服务器的调试选项将会生成文件 *xfer.ddt.PID*，即使你仅仅只是跟踪一个查询。我们的跟踪是在调试级别 1 上进行的，并且打开了 BIND 8 的日志选项 *print-time*（打印时间）。调试级别 3 能给你更多的信息，如果一次传输真正发生，其信息比你所想要的还多。在调试级别 3 下，跟踪几百条资源记录的区传送将会产生一个几百兆字节大小的 *xfer.ddt.PID* 文件。

```
21-Feb 00:13:18.026 do_zone_maint for zone movie.edu (class IN)
21-Feb 00:13:18.034 zone_maint('movie.edu')
21-Feb 00:13:18.035 qserial_query(movie.edu)
21-Feb 00:13:18.043 sysquery: send -> [192.249.249.3].53 dfd=5
                         nsid=29790 id=0 retry=888048802
21-Feb 00:13:18.046 qserial_query(movie.edu) QUEUED
21-Feb 00:13:18.052 next maintenance for zone 'movie.edu' in 2782 sec
21-Feb 00:13:18.056 datagram from [192.249.249.3].53, fd 5, len 380
21-Feb 00:13:18.059 qserial_answer(movie.edu, 26739)
21-Feb 00:13:18.060 qserial_answer: zone is out of date
21-Feb 00:13:18.061 startxfer(  ) movie.edu
21-Feb 00:13:18.063 /usr/etc/named-xfer -z movie.edu -f db.movie
                    -s 26738 -C 1 -P 53 -d 1 -l xfer.ddt 192.249.249.3
21-Feb 00:13:18.131 started xfer child 390
21-Feb 00:13:18.132 next maintenance for zone 'movie.edu' in 7200 sec

21-Feb 00:14:02.089 endxfer: child 390 zone movie.edu returned
                         status=1 termsig=-1
21-Feb 00:14:02.094 loadxfer(  ) "movie.edu"
21-Feb 00:14:02.094 purge_zone(movie.edu,1)

21-Feb 00:14:30.049 db_load(db.movie, movie.edu, 2, Nil)
21-Feb 00:14:30.058 next maintenance for zone 'movie.edu' in 1846 sec

21-Feb 00:17:12.478 slave zone "movie.edu" (IN) loaded (serial 26739)
21-Feb 00:17:12.486 no schedule change for zone 'movie.edu'

21-Feb 00:42:44.817 Cleaned cache of 0 RRs

21-Feb 00:45:16.046 do_zone_maint for zone movie.edu (class IN)
21-Feb 00:45:16.054 zone_maint('movie.edu')
21-Feb 00:45:16.055 qserial_query(movie.edu)
21-Feb 00:45:16.063 sysquery: send -> [192.249.249.3].53 dfd=5
                         nsid=29791 id=0 retry=888050660
```

```
21-Feb 00:45:16.066 qserial_query(movie.edu) QUEUED
21-Feb 00:45:16.067 next maintenance for zone 'movie.edu' in 3445 sec
21-Feb 00:45:16.074 datagram from [192.249.249.3].53, fd 5, len 380
21-Feb 00:45:16.077 qserial_answer(movie.edu, 26739)
21-Feb 00:45:16.078 qserial_answer: zone serial is still OK
21-Feb 00:45:16.131 next maintenance for zone 'movie.edu' in 2002 sec
```

与前面的跟踪记录不一样，在这次跟踪记录中每一行都有一个时间戳。这些时间戳能更清晰地把调试语句分成组。

该服务器是一个单个区 *movie.edu* 的辅名字服务器。时刻显示为 00:13:18.026 的行是检查主名字服务器的时间。在决定装载区数据之前，该服务器查询区的SOA 记录并比较序列号。时刻显示为从 00:13:18.059 到 00:13:18.131 的行是区的序列号（26739），告诉你区数据已过期，并启动一个子进程（pid 390）来传送区数据。在 00:13:18.132 时刻，一个计时器被设置成 7200 秒后到期，这是服务器允许一次传送完成的时间。在 00:14:02.089 时刻，你看到了该子进程的退出状态。状态 1 表明区数据已被成功地传送。旧的区数据被清除（时刻 00:14:02.094），新数据被装载。

下一次维护（见时刻 00:14:30.058）被安排在 1846 秒后进行。对这个区，刷新时间间隔是 3600，但该名字服务器选择在 1846 秒后再次检查。为什么？该名字服务器是为了设法避免刷新计时器变成同步。它使用了一个在一半刷新间隔时间（1800）和完全刷新间隔时间（3600）之间的一个随机时间，而不是严格地使用 3600。在 00:45:16.046 时刻区数据被再次检查，但这一次数据是新的。

如果你的跟踪足够长，你会看到更多的如在 00:42:44.817 时刻所示的行 —— 每隔 1 小时一行。原因是服务器遍历检查它的缓存，释放掉那些已过期的数据以减少占用的内存空间。

这个区的主名字服务器是一台BIND 4名字服务器。如果主名字服务器是一台BIND 8 的名字服务器的话，当区数据改变时，辅名字服务器将会被通知而不是等待刷新周期到期。辅名字服务器的调试输出几乎是完全一样的，但触发检查区状态的事件却是 NOTIFY：

```
rcvd NOTIFY(movie.edu, IN, SOA) from [192.249.249.3].1059
qserial_query(movie.edu)
sysquery: send -> [192.249.249.3].53 dfd=5
          nsid=29790 id=0 retry=888048802
```

检查区数据的辅名字服务器（BIND 9，调试级别 1）

通常，处于级别 1 的 BIND 9.1.0 的等价调试输出更简洁些。如下所示：

```
Sep 18 15:05:00.059 zone_timer: zone movie.edu/IN: enter
Sep 18 15:05:00.059 dns_zone_maintenance: zone movie.edu/IN: enter
Sep 18 15:05:00.059 queue_soa_query: zone movie.edu/IN: enter
Sep 18 15:05:00.059 soa_query: zone movie.edu/IN: enter
Sep 18 15:05:00.061 refresh_callback: zone movie.edu/IN: enter
Sep 18 15:05:00.062 refresh_callback: zone movie.edu/IN: Serial: new 2000010923,
old 2000010922
Sep 18 15:05:00.062 queue_xfrin: zone movie.edu/IN: enter
Sep 18 15:05:00.070 zone_xfrdone: zone movie.edu/IN: success
Sep 18 15:05:00.070 transfer of 'movie.edu' from 192.249.249.3#53: end of transfer
Sep 18 15:05:01.089 zone_timer: zone movie.edu/IN: enter
Sep 18 15:05:01.089 dns_zone_maintenance: zone movie.edu/IN: enter
Sep 18 15:05:19.121 notify_done: zone movie.edu/IN: enter
Sep 18 15:05:19.621 notify_done: zone movie.edu/IN: enter
```

在 15:05:00:059时刻的消息表示刷新计时器开始计时，引发名字服务器开始维护下一行所指的区。首先，名字服务器对它所发送的对区movie.edu（*queue_soa_query*）中的 IN 类的 SOA 记录的查询进行排队。在 15:05:00:062 时刻，名字服务器发现主名字服务器的序列号比它的序列号更高（主名字服务器是 2000010923，它是 2000010922），于是它对向内的区传送（*queue_xfrin*）进行排队。8 毫秒以后（即，15:05:00:070）区传送完成，在 15:05:01:089 时刻，名字服务器重置刷新计时器（*zone_timer*）。

接下来 3 行显示了名字服务器再次对 *movie.edu* 的维护。例如，如果 *movie.edu* 的一些名字服务器在 *moive.edu* 区之外，名字服务器可以利用这次机会来查找它们的地址（不仅是 A 记录，还有 A6 和 AAAA 记录！），使得可以在将来的响应中包括这些记录。最后两行，我们的名字服务器向 *movie.edu* 的 NS 记录中列出的名字服务器发出 NOTIFY 消息 —— 确切地说是两个。

解析器搜索算法和否定缓存(BIND 8)

在本次跟踪中，我们将以 BIND 8 名字服务器为例，向你描述 BIND 4.9 和后续版本的解析器搜索算法和否定缓存是什么。像上次跟踪一样，我们可以查询 *galt.cs.purdue.edu*，但是不能向你展示搜索算法。为此，我们将查询 *foo.bar*，一个不存在的名字。实际上，我们将查找两次：

```
1)  datagram from cujo.horror.movie.edu 1109, fd 6, len 25
2)  req: nlookup(foo.bar) id 19220 type=1 class=1
3)  req: found 'foo.bar' as '' (cname=0)
4)  forw: forw -> D.ROOT-SERVERS.NET 53 ds=7 nsid=2532 id=19220 0ms retry 4sec
5)
6)  datagram from D.ROOT-SERVERS.NET 53, fd 5, len 25
7)  ncache: dname foo.bar, type 1, class 1
8)  send_msg -> cujo.horror.movie.edu 1109 (UDP 6) id=19220
9)
10) datagram from cujo.horror.movie.edu 1110, fd 6, len 42
11) req: nlookup(foo.bar.horror.movie.edu) id 19221 type=1 class=1
12) req: found 'foo.bar.horror.movie.edu' as 'horror.movie.edu' (cname=0)
13) forw: forw -> carrie.horror.movie.edu 53 ds=7 nsid=2533 id=19221 0ms
                                                          retry 4sec
14) datagram from carrie.horror.movie.edu 53, fd 5, len 42
15) ncache: dname foo.bar.horror.movie.edu, type 1, class 1
16) send_msg -> cujo.horror.movie.edu 1110 (UDP 6) id=19221
```

再一次查找 *foo.bar*：

```
17) datagram from cujo.horror.movie.edu 1111, fd 6, len 25
18) req: nlookup(foo.bar) id 15541 type=1 class=1
19) req: found 'foo.bar' as 'foo.bar' (cname=0)
20) ns_req: answer -> cujo.horror.movie.edu 1111 fd=6 id=15541 size=25 Local
21)
22) datagram from cujo.horror.movie.edu 1112, fd 6, len 42
23) req: nlookup(foo.bar.horror.movie.edu) id 15542 type=1 class=1
24) req: found 'foo.bar.horror.movie.edu' as 'foo.bar.horror.movie.edu' (cname=0)
25) ns_req: answer -> cujo.horror.movie.edu 1112 fd=6 id=15542 size=42 Local
```

让我们来看看解析器的搜索算法。第一个被查找的名字（行 2）正是我们输入的名字。由于该名字至少有一个圆点，所以它未经修改地被查找。当该查找失败时，将 *horror.movie.edu* 添加到该名字尾部之后再查。（比 BIND 4.9 更早版本的解析器会尝试添加 *horror.movie.edu* 和 *movie.edu*。）

行 7 显示将否定回答（ncache）放入缓存。如果在接下来的几分钟内再次查找同一名字（行 19），名字服务器的缓存中还有该否定响应，则该服务器可以立即回答该名字不存在。（如果拿不准，你可对行3和行19做一个比较。在行3中，对名字*foo.bar*没有找到任何东西，但行 19 却显示了要查找的完整名字。）

解析器搜索算法和否定缓存(BIND 9)

这里是对 *foo.bar* 两次查找时，BIND 9.1.0 名字服务器的调试输出：

```
Sep 18 15:45:42.944 client cujo.horror.movie.edu#1044: query: foo.bar A
Sep 18 15:45:42.945 createfetch: foo.bar. A
Sep 18 15:45:42.945 createfetch: . NS
Sep 18 15:45:43.425 client cujo.horror.movie.edu#1044: query: foo.bar.horror.
movie.edu A
Sep 18 15:45:43.425 createfetch: foo.bar.horror.movie.edu. A
```

比起 BIND 8 来，这个输出结果更细致、简洁，并且你能够从中得到你需要的信息。第 1 行，在 15:45:42:944 时刻，给出了来自客户 *cujo.horror.movie.edu* 对 *foo.bar* 地址的初始查询（记住，我们通过一个从 IP 到名字的过滤器运行这个操作，这将在后面做介绍）。下面两行显示了名字服务器分派两个查找 *foo.bar* 的任务（*createfetch*）：第一个是实际查找 *foo.bar* 地址；而第二个是查找根区 NS 记录的辅助任务，这对于完成 *foo.bar* 的查找是必须的。一旦名字服务器有了根区的当前 NS 记录，它就会为 *foo.bar* 的地址查询根名字服务器并且得到一个响应，该响应表明没有叫做 *bar* 的顶级域名存在。不幸的是，你看不到该响应。

在 15:45:43:425 时刻的行表示 *cujo.horror.movie.edu* 使用查询列表来查找 *foo.bar.horror.movie.edu*，这使得名字服务器分派一个任务（*createfetch*）去查找那个域名。

当我们再次查找 foo.bar 时，将会看到：

```
Sep 18 15:45:46.557 client cujo.horror.movie.edu#1044: query: foo.bar A
Sep 18 15:45:46.558 client cujo.horror.movie.edu#1044: query: foo.bar.horror.
movie.edu A
```

注意，这里没有 *createfetch* 条目？因为我们的名字服务器具有缓存的否定回答。

工具

我们曾提起过一种把 IP 地址转换成名字，使调试输出信息更容易阅读的工具。下面的程序就可以完成这样的功能，它是用 Perl 写成的：

```
#!/usr/bin/perl -n

use "Socket";

if (/\b)(\d+\.\d+\.\d+\.\d+)\b/) {
     $addr = pack('C4', split(/\./, $1));
     ($name, $rest) = gethostbyaddr($addr, &AF_INET);
     if($name) {s/$1/$name/;
```

```
}

print;
```

最好不要在调试选项被打开时将*named.run*的输出导入到这个脚本程序，因为该脚本程序将向名字服务器生成它自己的查询。

本章内容：

- NIS 确实是你的问题吗？
- 故障诊断与排除的工具和技术
- 潜在问题列表
- 版本升级带来的问题
- 互操作性和版本问题
- TSIG 错误
- 故障症状

第十四章

DNS 和 BIND 排错

“当然不，”假海龟说道。
“当一条鱼向我游来并告诉我它要去旅行时，
我为什么应该说‘With what porpoise?’（译注 1）”
“你说的是否是‘purpose’（译注 2）？”
爱丽丝说道。
“我的意思就是我所说的，”
假海龟用一种不友好的语调回答道。
那只半狮半鹫的怪兽加上一句，
“来，让我们听听你的历险故事。”

在前两章中，我们讨论了如何使用 *nslookup* 和 *dig*，还有如何阅读名字服务器的调试信息。在本章中，我们将告诉你如何使用这些工具，还有如何使用 *ping* 这样可靠的传统 Unix 网络工具来诊断和排除现实生活中 DNS 和 BIND 的故障。

故障诊断与排除，就其本质而言，是一个很难讲授的课题。你从一种故障症状开始，设法追溯到引起该故障的原因。我们不可能涵盖你在 Internet 上可能遇到的所有问题，但我们会尽最大的努力来向你讲述如何诊断其中最常见的故障。而且同时，我

译注 1： 该句意为“与什么海豚去？”

译注 2： purpose 是“目的”的意思，其读音与 porpoise“海豚”相似。

们希望教你一些故障诊断与排除技术，这些技术在解决我们未介绍的更困难的问题时会很有价值的。

NIS 确实是你的问题吗？

在我们开始讨论如何诊断与排除DNS和BIND故障之前，我们要确信你能分辨一个故障是由DNS还是由NIS引起的。在运行NIS的主机上，推断出引起故障的罪魁祸首是DNS还是NIS可能会很困难。例如，常见的BSD *nslookup*就没有考虑到NIS。你可以在一台Sun机器上运行*nslookup*并查询名字服务器，与此同时其他的服务却使用NIS。

怎么知道故障是由谁引起的呢？有些厂商修改了*nslookup*使其在配置了NIS的情况下使用NIS来提供名字服务。例如，HP-UX的*nslookup*在启动时将报告它正在查询一个NIS服务器：

```
% nslookup
Default NIS Server:  terminator.movie.edu
Address:  192.249.249.3

>
```

在vanilla版*nslookup*的主机上，你常常可以使用*ypmatch*来判定你正在使用的是DNS还是NIS。如果*ypmatch*收到了来自名字服务器的数据，它将在主机信息之后打印出一个空行。所以在本例中，回答来自NIS：

```
% ypmatch ruby hosts
140.186.65.25   ruby ruby.ora.com
%
```

而在这个例子中，回答来自一个名字服务器：

```
% ypmatch harvard.harvard.edu hosts
128.103.1.1     harvard.harvard.edu

%
```

注意，它在SunOS 4.1.1上是有效的，但不能保证在SunOS的未来所有版本上均能工作。据我们所知，这个附带有bug的特性可能将在下一版本中消失。

一种用来判断回答是否来自NIS的更可靠的方法是用*ypcat*来列出*hosts*数据库。例如，若想找出 *andrew.cmu.edu* 是否在你的NIS 主机映射表中，可以执行：

```
% ypcat hosts | grep andrew.cmu.edu
```

如果你在NIS中找到了答案（并且你知道首先被查询的是NIS），你就发现了问题的原因。

最后，在那些使用*nsswitch.conf*文件的Unix版本中，你可以通过查找该文件中*hosts*数据库的条目来判定不同名字服务的使用顺序。例如，下面这样的设置项表明首先检查 NIS：

```
hosts:    nis dns files
```

而下面的设置项则让名字解析器首先查询 DNS：

```
hosts:    dns nis files
```

关于 *nsswitch.conf* 文件的语法和语义的详细信息，请参阅第六章。

上面提到的这些线索将帮助你识别有问题的部分，或者至少帮助你排除怀疑的对像。如果你已缩小了怀疑的范围，而DNS仍在受怀疑的范围之内，那你就只好阅读本章了。

故障诊断与排除的工具和技术

在前两章中我们讨论了 *nslookup* 和 *dig*，还有名字服务器的调试输出。在继续讨论之前，让我们先介绍几个在故障诊断与排除时可能有用的新工具：*named-xfer*、名字服务器数据库转储和查询日志。

如何使用 named-xfer

named-xfer 是 BIND 4 和 8 名字服务器开始执行区数据传送的程序。（不过要注意，BIND 9名字服务器是多线程的，所以不需要一个单独程序来进行向内的区传送：只是启动一个新线程。）程序*named-xfer*检查slave服务器的区数据副本是否是最新的，

如果需要就传送新的区数据。(在 BIND 4.9 和 8 中，*named* 首先检查区数据是否是最新的以避免在没有必要传送时启动一个子进程。)

在第十三章中，我们向你显示过当一个 BIND 8 辅名字服务器在检查它的区数据时所记录下来的调试输出信息。当辅名字服务器传送区数据时，它启动一个子进程(*named-xfer*) 来把数据下载到本地文件系统。然而，我们没有告诉你也可以用手工的方式来启动*named-xfer*, 而不必等待*named*来启动它, 而且你也可以不通过*named*来让它产生调试输出信息。

当你在跟踪区传送故障而又不想等待由 *named* 来安排一个区传送时，这是很有用的。为了用手工的方式来测试区数据传送，你需要指定几个命令行选项：

```
%   /usr/sbin/named-xfer
Usage error: no domain
Usage: named-xfer
        -z zone_to_transfer
        -f db_file
        [-i ixfr_file]
        [-s serial_no]
        [-d debug_level]
        [-l debug_log_file]
        [-t trace_file]
        [-p port]
        [-S] [-Z]
        [-C class]
        [-x axfr-src]
        [-T tsig_info_file]
        servers [-ixfr|-axfr]...
```

这是 BIND 8.2.3 版本的 *named-xfer* 的输出结果。早期版本的 *named-xfer* 并不支持所有这些选项。

当 *named* 启动 *named-xfer* 时，它指定 *-z* 选项（*named* 想要检查的区)、*-f* 选项（该区相应的区数据文件名，来自 *named.boot* 或者 *named.conf*)、*-s* 选项（slave 服务器上该区的序列号，来自当前的SOA 记录）和辅名字服务器被指示去获得数据的服务器的地址（来自 *named.conf* 文件中 *zone* 语句的 *masters* 子语句或 *named.boot* 文件中 *secondary* 指令中的 IP 地址)。如果 *named* 在调试模式下运行，它还用 *-d* 选项来指定 *named-xfer* 的调试级别。排错时通常用不到其他选项；那些选项与增量区传送、TSIG 签名区传送等有关。

当你用手工方式来运行 *named-xfer* 时，你也可以在命令行中用 *–d* 选项来指定调试级别。（不过不要忘了，如果传送成功，高于级别 3 的调试将会产生大量的调试输出信息！）你也可以用 *–l* 选项来为调试输出文件指定一个可选文件名。默认的日志文件是 */var/tmp/xfer.ddt.XXXXXX*，这里的 *XXXXXX* 是一个附加的以保证文件名惟一性的后缀，或者是在目录 */usr/tmp* 下的同名文件。而且你可以指定要去获得数据的服务器的主机名，而不是它的 IP 地址。

例如，用下面的命令你可以检查从 *terminator.movie.edu* 传送区数据的工作是否正常：

```
%   /usr/sbin/named-xfer -z movie.edu -f /tmp/db.movie -s 0 terminator
% echo $?
4
```

在这条命令中，我们指定了一个为零的序列号，因为我们想强制 *named-xfer* 进行区数据传送，即使这个传送是不必要的。零是一个特殊的序列号 —— *named-xfer* 将传送区数据而不管实际的序列号是多少。另外，我们告诉 *named-xfer* 把新的区数据放在 */tmp* 目录下的文件，而不是覆盖该区的工作区数据文件。

我们可以通过查看 *named-xfer* 的返回值来判断传送是否成功。如果你运行的是 BIND 8.1.2 或者更老的版本，*named-xfer* 有四个可能的返回值：

0 区数据是最新的且不需要传送。

1 表示传送成功。

2 *named-xfer* 查询的主机不可达，或出现了错误且 *named-xfer* 在系统日志文件中已记录了一个出错消息。

3 出现了错误且 *named-xfer* 在系统日志文件中已记录了一个出错消息。

在 BIND 8.2 中，又新增了三个返回值以适应增量区传送：

4 表示成功的 AXFR（完全）区传送。

5 表示成功的 IXFR（增量）区传送。

6 表示主名字服务器对 *named-xfer* 的 IXFR 请求返回了一个 AXFR。

对于一个名字服务器来说，对增量区传送请求返回一个完全区传送也是很正常的，即使该服务器是支持 IXFR 的。例如，主名字服务器可能漏掉了部分对该区更改的记录。

注意，BIND 8.2 和后续版本的 *named-xfer* 不再使用返回值 1，它已经被返回值 4 到 6 取代了。

如果没有 named-xfer 怎么办？

如果你已经升级到 BIND 9，没有 *named-xfer* 程序，你仍然可以使用 *nslookup* 或 *dig* 来进行区传送。这两种查询工具都可以给你一些 *named-xfer* 能提供的信息。

例如，用 *dig* 来完成我们前面讲到过的那个区传送，你可以按如下操作：

```
%  dig @terminator.movie.edu movie.edu axfr
```

利用 *nslookup* 你可以在交互模式下改变名字服务器并使用 *ls –d* 命令。

不幸的是，在报告出错方面，*nslookup* 和 *dig* 比 *named-xfer* 要模糊得多。如果 *nslookup* 不能传送一个区，它通常会报告“未指定的错误”：

```
>  ls movie.edu
[terminator.movie.edu]
*** Can't list domain movie.edu: Unspecified error
```

这可能是由 *allow-transfer* 访问列表引起的，或者是因为 *terminator.movie.edu* 不是 *movie.edu* 的权威，或者由于别的一些问题。想分辨到底是哪一个，你可能需要发送其他一些相关的查询或者检查一下主名字服务器上的系统日志文件。

如何阅读数据库转储

细读名字服务器的内部数据库转储，包括缓存的信息，也可以帮助你找到故障。*ndc dumpdb* 或者 *rndc dumpdb* 命令会使 *named* 将它的权威数据、缓存数据和线索数据写到 BIND 工作目录下的 *named_dump.db* 文件中（在 BIND 4 中，写到 */user/tmp/named_dump.db* 或者 */var/tmp/named_dump.db* 文件中）（注 1）。下面是一个

注 1： BIND9.1.0 是最早支持转储数据库的 BIND9 版本。

named.dump.db 文件的例子。权威数据和缓存数据，混合在一起，出现在文件的开始部分。最后是线索数据：

```
; Dumped at Tue Jan  6 10:49:08 1998
;; ++zone table++
; 0.0.127.in-addr.arpa (type 1, class 1, source db.127.0.0)
;   time=0, lastupdate=0, serial=1,
;   refresh=0, retry=3600, expire=608400, minimum=86400
;   ftime=884015430, xaddr=[0.0.0.0], state=0041, pid=0
;; --zone table--
; Note: Cr=(auth,answer,addtnl,cache) tag only shown for non-auth RR's
; Note: NT=milliseconds for any A RR which we've used as a nameserver
; --- Cache & Data ---
$ORIGIN .
.    518375  IN      NS  G.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  J.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  K.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  L.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  M.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  A.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  H.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  B.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  C.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  D.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  E.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  I.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
     518375  IN      NS  F.ROOT-SERVERS.NET.   ;Cr=auth [128.8.10.90]
EDU  86393  IN      SOA A.ROOT-SERVERS.NET.  hostmaster.INTERNIC.NET. (
         1998010500 1800 900 604800 86400 )   ;Cr=addtnl [128.63.2.53]
$ORIGIN  0.127.in-addr.arpa.
0        IN    SOA cujo.movie.edu. root.cujo.movie.edu. (
         1998010600 10800 3600 608400 86400 )        ;Cl=5
         IN    NS  cujo.movie.edu.   ;Cl=5
$ORIGIN  0.0.127.in-addr.arpa.
1        IN    PTR localhost.    ;Cl=5
$ORIGIN EDU.
PURDUE   172787  IN  NS  NS.PURDUE.EDU.           ;Cr=addtnl [192.36.148.17]
         172787  IN  NS  MOE.RICE.EDU.            ;Cr=addtnl [192.36.148.17]
         172787  IN  NS  PENDRAGON.CS.PURDUE.EDU.  ;Cr=addtnl [192.36.148.17]
         172787  IN  NS  HARBOR.ECN.PURDUE.EDU.    ;Cr=addtnl [192.36.148.17]
$ORIGIN  movie.EDU.
;cujo    593     IN  SOA  A.ROOT-SERVERS.NET. hostmaster.INTERNIC. NET. (
;        1998010500 1800 900 604800 86400 );EDU.; NXDOMAIN  ;-$
   ;Cr=auth [128.63.2.53]
$ORIGIN   RICE.EDU.
MOE      172787  IN  A   128.42.5.4        ;NT=84 Cr=addtnl [192.36.148.17]
$ORIGIN   PURDUE.EDU.
CS       86387   IN  NS  pendragon.cs.PURDUE.edu.    ;Cr=addtnl [128.42.5.4]
         86387   IN  NS  ns.PURDUE.edu.              ;Cr=addtnl [128.42.5.4]
         86387   IN  NS  harbor.ecn.PURDUE.edu.      ;Cr=addtnl [128.42.5.4]
         86387   IN  NS  moe.rice.edu.               ;Cr=addtnl [128.42.5.4]
```

```
NS      172787   IN  A  128.210.11.5       ;NT=4 Cr=addtnl [192.36.148.17]
$ORIGIN   ECN.PURDUE.EDU.
HARBOR   172787  IN   A  128.46.199.76      ;NT=6 Cr=addtnl [192.36.148.17]
$ORIGIN   CS.PURDUE.EDU.
galt     86387   IN   A  128.10.2.39               ;Cr=auth [128.42.5.4]
PENDRAGON  172787  IN  A  128.10.2.5        ;NT=20 Cr=addtnl [192.36.148.17]
$ORIGIN   ROOT-SERVERS.NET.
K        604775    IN  A  193.0.14.129       ;NT=10 Cr=answer [128.8.10.90]
A        604775    IN  A  198.41.0.4         ;NT=20 Cr=answer [128.8.10.90]
L        604775    IN  A  198.32.64.12       ;NT=8 Cr=answer [128.8.10.90]
B        604775    IN  A  128.9.0.107        ;NT=9 Cr=answer [128.8.10.90]
M        604775    IN  A  202.12.27.33       ;NT=20 Cr=answer [128.8.10.90]
C        604775    IN  A  192.33.4.12        ;NT=17 Cr=answer [128.8.10.90]
D        604775    IN  A  128.8.10.90        ;NT=11 Cr=answer [128.8.10.90]
E        604775    IN  A  192.203.230.10     ;NT=9 Cr=answer [128.8.10.90]
F        604775    IN  A  192.5.5.241        ;NT=73 Cr=answer [128.8.10.90]
G        604775    IN  A  192.112.36.4       ;NT=14 Cr=answer [128.8.10.90]
H        604775    IN  A  128.63.2.53        ;NT=160 Cr=answer [128.8.10.90]
I        604775    IN  A  192.36.148.17      ;NT=102 Cr=answer [128.8.10.90]
J        604775    IN  A  198.41.0.10        ;NT=21 Cr=answer [128.8.10.90]
; --- Hints ---
$ORIGIN .
.   3600           IN  NS  A.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  B.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  C.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  D.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  E.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  F.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  G.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  H.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  I.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  J.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  K.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  L.ROOT-SERVERS.NET.     ;Cl=0
    3600           IN  NS  M.ROOT-SERVERS.NET.     ;Cl=0
$ORIGIN   ROOT-SERVERS.NET.
K     3600         IN   A  193.0.14.129      ;NT=11 Cl=0
L     3600         IN   A  198.32.64.12      ;NT=9 Cl=0
A     3600         IN   A  198.41.0.4        ;NT=10 Cl=0
M     3600         IN   A  202.12.27.33      ;NT=11 Cl=0
B     3600         IN   A  128.9.0.107       ;NT=1288 Cl=0
C     3600         IN   A  192.33.4.12       ;NT=21 Cl=0
D     3600         IN   A  128.8.10.90       ;NT=1288 Cl=0
E     3600         IN   A  192.203.230.10    ;NT=19 Cl=0
F     3600         IN   A  192.5.5.241       ;NT=23 Cl=0
G     3600         IN   A  192.112.36.4      ;NT=18 Cl=0
H     3600         IN   A  128.63.2.53       ;NT=11 Cl=0
I     3600         IN   A  192.36.148.17     ;NT=21 Cl=0
J     3600         IN   A  198.41.0.10       ;NT=13 Cl=0
```

生成这个*named.dump.db*文件的名字服务器仅仅对*0.0.127.in-addr.arpa*是权威的。用这个服务器能查到的名字只有两个：*galt.cs.purdue.edu*和*cujo.movie.edu*。在查

找*galt.cs.purdue.edu*的过程中，该服务器不仅缓存了*galt*的地址，而且还缓存了域*purdue.edu*的名字服务器列表和这些服务器的地址。然而，名字*cujo.movie.edu*实际上并不存在（除了在我们的例子中，域*movie.edu*也不存在），所以该服务器缓存了这个否定响应。在转储文件中，否定响应被注释掉了（用分号开始的行），而且列出了原因（NXDOMAIN）而不是真实数据。你将注意到 TTL 时间值是相当低的（593）。在BIND 8.2及其后续版本的名字服务上，否定响应放在缓存中的时间是由SOA 记录中最后一个字段的值决定的，通常比该区的默认 TTL 小得多。

在该文件底部的线索数据部分包含了来自文件*db.cache*的数据。线索数据的TTL时间值被减小，并可减小到零，但是线索数据永远不会被扔掉。

注意，某些资源记录跟着一个分号和*NT=*。只有在名字服务器的地址记录上你才会看到这些符号。数字是名字服务器保存的往返时间计算值，因而它知道过去哪一个服务器的响应最快，在下一次查询时，会首先尝试往返时间值最低的那个服务器。

缓存数据很容易分辨，就是那些有可信度标记（*Cr=*）的数据项，有时还有数据来源服务器的IP 地址（注2）。区数据和线索数据由（*Cl=*）标记，该标记只是一个在域树中的层次计数。（根域是层次 0，*foo* 是层次 1，*foo.foo* 是层次 2，等等。）让我们暂时脱离一下主题来解释可信度的概念。

在4.9和4.8.3版之间的进步之一就是增加了可信度的度量。这允许名字服务器在如何处理来自远程服务器的数据时能够进行更智能的决策。

4.8.3 名字服务器只有两个可信度级别 —— 本地权威数据和其他。本地权威数据就是从你的区数据文件中得来的数据，你的名字服务器知道最好不去更新来自你自己的区文件的内部数据。但是所有来自远程名字服务器的数据均被同等地看待。

注 2： 如果可以得到的话，名字服务器会给出远程名字服务器的IP地址。在BIND 8.2及其后续版本的名字服务器上，只有当*host-statistics*是打开的时候才能得到IP地址，在第八章中我们曾介绍过。在早期的BIND 4.9和BIND 8名字服务器上，默认情况就是打开的。*host-statistics*会保存大量的统计数据，关于每个与你联系过的名字服务器和解析器，这从某种角度来说是非常有用的（比如，指出你的服务器从哪个名字服务器那里得到过记录），但这会耗费相当多的内存。

下面我们将讲述一种可能会发生的情况，以及在该情况下4.8.3版的服务器可能的处理方式。假如你的服务器查询名字 *terminator.movie.edu* 的地址，并且从 *movie.edu* 名字服务器上收到一个权威响应。（记住，权威响应是你可以得到的最好响应。）在稍后某个时刻你查询 *foo.ora.com*，你的服务器收到另一个 *terminator.movie.edu* 的地址记录，但是这一次它是作为 *oreilly.com* 的授权信息的一部分（*terminator.movie.edu* 是它的一个辅名字服务器）。尽管数据来自 *com* 域的名字服务器而不是 *movie.edu*的权威服务器，4.8.3版的名字服务器将会更新其缓存中*terminator.movie.edu*的地址记录。当然，*com*和*movie.edu*名字服务器所拥有的关于*terminator.movie.edu* 的数据将会完全相同，因此这不会有问题，对吧？啊哈，就像在南加州从不会下雨。

4.9或更新版本的名字服务器更聪明些。像4.8.3服务器那样，它对你的区数据毫不怀疑。但4.9或更新的名字服务器会区分来自远程名字服务器的不同数据。这里是远程数据可信度的层次结构，从最可信到最不可信：

auth
: 这些记录是权威应答数据 —— 响应消息的响应段中权威响应位被置位。

answer
: 这些记录是来自非权威的或缓存中的响应数据 —— 在响应消息的回答段中权威回答位未被置位。

addtnl
: 这些记录是来自响应消息中其他段的数据 —— 权威段和附加段。响应中的权威段包含将一个区授权给一个权威名字服务器的NS记录。附加段中包含地址记录，它们是对其他段中信息的补充（例如，与权威段中NS记录对应的地址记录）。

对这一规则有个例外：当服务器正在启动它的根名字服务器缓存时，可信度为 *addtnl* 的记录的可信度被改为 *answer*，这样就不会那么容易被误修改了。注意，在该转储中，根名字服务器的地址记录的可信度是 *answer*，而 *purdue.edu* 名字服务器的地址记录的可信度是 *addtnl*。

在刚刚描述过的情况中，4.9或较新版本的名字服务器将不会用 *terminator.movie.edu* 的授权数据（可信度 =*addtnl*）去替代权威数据（可信度 =*auth*），因为权威回答具有较高的可信度。

记录查询日志

BIND 4.9 版增加了一个特性，称为记录查询日志（query logging），这可以用来帮助诊断某些问题。当记录查询日志被打开时，运行的名字服务器将会把每一个查询都记录到系统日志中。这一特性可以帮助你找出解析器的配置错误，因为你可以确认你正在查询的名字正是你想要查询的名字。

首先你必须确信级别为LOG_INFO的消息在相应的日志设备守护进程中确实能够通过 *syslog* 记录下来。接下来你需要打开记录查询日志。这可以通过以下几种方法来完成：针对 BIND 4.9，在你的名字服务器的启动文件中设置 *options query-log* ；针对 BIND 4.9 或 BIND 8，在命令行中用 *–q* 选项来启动名字服务器，或向正在运行的名字服务器发送一个 *ndc querylog* 命令；针对 BIND 9.1.0 及其后续版本的名字服务器（早期版本不支持记录查询日志），使用 *rndc querylog*。你将开始看到如下所示的系统日志消息：

```
Feb 20 21:43:25 terminator named[3830]:
                    XX+ /192.253.253.2/carrie.movie.edu/A
Feb 20 21:43:32 terminator named[3830]:
                    XX+ /192.253.253.2/4.253.253.192.in-addr.arpa/PTR
```

如果你运行的是 BIND 9，将会看到：

```
Jan 13 18:32:25 terminator named[13976]: info: client 192.253.253.2#1702: query:
carrie.movie.edu IN A
Jan 13 18:32:42 terminator named[13976]: info: client 192.253.253.2#1702: query:
4.253.253.192.in-addr.arpa IN PTR
```

这些消息包括产生查询的主机的 IP 地址和查询本身。由于第一个例子来自 BIND 8.2.3 名字服务器，而且查询是递归的，所以它们以 XX+ 开头。重复查询只以 XX 开头。（BIND 8.2.1 以前版本的名字服务器对递归查询和非递归查询不做区分）。逆向查询在查询类型前有一个破折号（例如，对地址记录的逆向查询将会被记录成“ -A ”而不只是“ A ”）。当记录了足够多的查询后，你可以通过再发送一个 *ndc querylog* 或者 *rndc querylog* 命令给你的名字服务器来关闭记录查询日志。

如果你仍在坚持使用着旧的 BIND 9 名字服务器，则可以在级别上的 *named* 的调试输出中看到所接收的查询。

潜在问题列表

既然我们已经给了你一些很好的工具，那么就让我们来谈谈如何使用它们来诊断实际的问题。有的故障很容易识别和改正。我们将把它们作为一个课程来讨论 —— 它们是一些最常见的故障，因为它们是由一些最常见的错误造成的。下面所说的故障是不分先后次序的，总共有十三种。

1、忘了增加序列号

这个问题的主要症状是辅名字服务器没有获得你在主名字服务器上对区数据文件所进行的任何修改。辅名字服务器认为区数据没有被修改过，因为序列号没变。

怎样检查你是否记得增加序列号了呢？不幸得很，这不那么容易。如果你不记得旧的序列号，而你的序列号又没有任何指示表明它是何时被更新的，那么就没有任何直接的方法可以知道它是否被更改过（注3）。当你重新加载主名字服务器时，无论你是否修改过序列号，它都将装载更新过的区数据文件。它将检查文件的时间戳，如果自上一次装载数据以来该文件被修改过，就读入该文件。你最好是用 *nslookup* 来比较主名字服务器和辅名字服务器所返回的数据。如果它们返回不同的数据，那可能就是你忘了增加序列号。如果你能记得最近所做过的修改，可以查询那个最近修改过的数据。如果你忘记了最近所做的修改，可以尝试从主名字服务器和辅名字服务器上传送区数据，对结果排序，再用 *diff* 命令来比较它们。

所幸的是，尽管判断区数据是否被传送过比较困难，但是保证传送了区数据却很简单。只需增加主名字服务器区数据文件的序列号，并重新装载主名字服务器上的区。辅名字服务器将会在其刷新周期内；或在更短的时间内得到新数据，如果它们使用 NOTIFY 的话。如果想确信辅名字服务器传送了新数据，你可以用手工方式执行 *named-xfer* 命令（当然是在辅名字服务器上）：

```
#    /usr/sbin/named-xfer -z movie.edu -f db.movie -s 0 terminator.movie.edu
#   echo $?
```

注3： 另一方面，如果你要将日期解码为序列号，正如多数人所做的（例如，2001010500就是2001年1月5日第一次修改数据），在修改数据时你看一眼就能分辨出是否更新了序列号。

如果 *named-xfer* 返回 1 或者 4，那么区数据已被成功地传送。其他返回值表明没有区数据被传送，或是因为出错，或是因为辅名字服务器认为其区数据是最新的。（详见本章前面的小节，“ 如何使用 *named-xfer* ”。）

还有一种错误也是因为忘了增加序列号。如果系统管理员使用像 *h2n* 这样的工具从主机表来创建区数据文件时，我们可能会看到这种情况。使用像 *h2n* 这样的脚本程序，删除旧的区数据文件和从零开始创建新的区数据文件，是十分容易的。某些系统管理员会偶尔这么做，因为他们错误地认为旧的区数据文件中的数据会包含在新的文件中。删除这些区数据文件所带来的问题是，由于没有旧数据文件可读出当前的序列号，*h2n* 将以序列号 1 从头开始。如果你的主名字服务器上区的序列号从 598 或其他你所使用的数值重新回到 1 时，辅名字服务器（4.8.3 或更早版本）不会发现问题，它们将认为自己的数据是最新的而不需要进行区传送。然而，4.9 或更新版本的辅名字服务器很警觉，会发出一条系统日志出错消息来警告你可能某处出了错：

```
Jun  7 20:14:26 wormhole named[29618]: Zone "movie.edu"
                (class 1) SOA serial# (1) rcvd from [192.249.249.3]
                is < ours (112)
```

所以，如果主名字服务器上的序列号看上去小得让人怀疑，那么也检查一下辅名字服务器上的序列号，并将它们做一个比较：

```
% nslookup
Default Server:  terminator.movie.edu
Address:  192.249.249.3

> set q=soa
> movie.edu.
Server:  terminator.movie.edu
Address:  192.249.249.3

movie.edu
        origin = terminator.movie.edu
        mail addr = al.robocop.movie.edu
        serial = 1
        refresh = 10800 (3 hours)
        retry   = 3600 (1 hour)
        expire  = 604800 (7 days)
        minimum ttl = 86400 (1 day)
> server wormhole.movie.edu.
Default Server:  wormhole.movie.edu
Addresses:  192.249.249.1, 192.253.253.1

> movie.edu.
Server:  wormhole.movie.edu
```

```
Addresses:  192.249.249.1, 192.253.253.1

movie.edu
        origin = terminator.movie.edu
        mail addr = al.robocop.movie.edu
        serial = 112
        refresh = 10800 (3 hours)
        retry   = 3600 (1 hour)
        expire  = 604800 (7 days)
        minimum ttl = 86400 (1 day)
```

作为 *movie.edu* 的辅名字服务器，*wormhole.movie.edu* 应该永远不会有比主名字服务器还大的序列号，所以显然是某个地方出了问题的。

顺便说一句，使用第十五章中我们所写的工具来发现这一问题实际上是很容易的。

2、忘了重新加载主名字服务器

偶尔你也会在对配置文件或区数据文件做了修改后，忘了重载你的主名字服务器。名字服务器不会知道要去装载新的配置或者新的区数据 —— 它不会自动检查文件的时间戳来发现它已被修改。结果是你所做过的任何更改都不会在名字服务器的数据中反映出来：新的区数据没有被装载，而且新的记录不会传播到辅名字服务器。

对 BIND 9 名字服务器来说，要查出上一次重载名字服务器的时间，可以在系统日志输出中查找这样一条消息：

```
Mar  8 17:22:08 terminator named[22317]: loading configuration from '/etc/named.
conf'
```

对 BIND 4.9 或 BIND 8 名字服务器来说，消息是这样的：

```
Mar  8 17:22:08 terminator named[22317]: reloading nameserver
```

这些信息告诉你最近一次向名字服务器发送重载命令的时间。如果你关闭并重启名字服务器，在 BIND 9 名字服务器上，你会看到：

```
Mar  8 17:22:08 terminator named[22317]: starting BIND 9.1.0
```

或在 BIND 8 名字服务器上看到：

```
Mar  8 17:22:08 terminator named[22317]: restarted
```

或在 4.9 名字服务器上看到：

```
Mar  8 17:22:08 terminator named[22317]: starting
```

如果重启或者重载的时间与你最近的修改时间没有关系，那么就重载名字服务器。同时也检查你是否在区数据文件中增加了序列号。如果你不确定你编辑区数据文件的时间，可以用 *ls –l* 命令列出文件列表来检查文件的修改时间。

3、辅名字服务器不能装载区数据

如果辅名字服务器不能从其主名字服务器获得区的当前序列号，它会通过*syslog* 记录一条消息。在 BIND 9 名字服务器上，是这样的：

```
Sep 25 22:02:38 wormhole named[21246]: refresh_callback: zone movie.edu/IN:
failure for 192.249.249.3#53: timed out
```

在 BIND 8 上，是这样的：

```
Jan  6 11:55:25 wormhole named[544]: Err/TO getting serial# for "movie.edu"
```

在 BIND 4 上，是这样的：

```
Mar  3 8:19:34 wormhole named[22261]: zoneref: Masters for secondary
       zone movie.edu unreachable
```

如果不理会这个问题，辅名字服务器将会使该区期满。BIND 9 名字服务器将报告：

```
Sep 25 23:20:20 wormhole named[21246]: zone_expire: zone movie.edu/IN: expired
```

BIND 4.9 或 8 名字服务器将会记录如下日志：

```
Mar  8 17:12:43 wormhole named[22261]: secondary zone
       "movie.edu" expired
```

一旦区已期满，当你向名字服务器查询在该区的数据时，将会得到SERVFAIL错误：

```
%  nslookup robocop wormhole.movie.edu.
Server:  wormhole.movie.edu
Addresses:  192.249.249.1, 192.253.253.1

*** wormhole.movie.edu can't find robocop.movie.edu: Server failed
```

有三种引起该错误的主要原因：由于网络错误而失去与主名字服务器的通信连接，在配置文件中设置了错误的主名字服务器IP地址，以及在主名字服务器上区数据文件中有语法错误。首先检查配置文件中该区的数据项，看看辅名字服务器是从哪个IP 地址装载数据的：

```
zone "movie.edu" {
                type slave;
                masters { 192.249.249.3; };
                file "bak.movie.edu";
};
```

在 BIND 4 版本的服务器上，该指令是这样的：

```
secondary        movie.edu        192.249.249.3        bak.movie.edu
```

要保证那就是主名字服务器的 IP 地址。如果是，就检查与那个 IP 地址的网络连接：

```
%   ping 192.249.249.3 -n 10
PING 192.249.249.3: 64 byte packets

----192.249.249.3 PING Statistics----
10 packets transmitted, 0 packets received, 100% packet loss
```

如果主名字服务器不可达，看看该名字服务器所在的主机是否确实在运行（比如，电源是开着的，等等），或者查找网络问题。如果主名字服务器是可达的，确信该主机上的 *named* 正在运行，而且你可以手工地传送区数据：

```
#    /usr/sbin/named-xfer -z movie.edu -f /tmp/db.movie.edu -s 0 192.249.249.3
#   echo $?
2
```

返回码 2 表明一个错误出现了。查看一下是否有系统日志消息。在本例中有一个消息：

```
Jan  6 14:56:07 zardoz named-xfer[695]: record too short from [192.249.249.3],
zone movie.edu
```

乍一看，这一错误像是一个截断错误。如果你使用 *nslookup*，真正的问题会比较容易看出来：

```
% nslookup - terminator.movie.edu
Default Server:  terminator.movie.edu
Address:  192.249.249.3
```

```
> ls movie.edu                          —尝试一个区传送
[terminator.movie.edu]
*** Can't list domain movie.edu: Query refused
```

这说明 *named* 正在拒绝你传送它的区数据。该远程服务器已经对其区数据通过用 *allow-transfer* 子语句、*secure_zone* 资源记录或 *xfrnets* 启动文件指令进行了安全保护。

如果主名字服务器对该区的响应是非权威的，在 BIND 9 名字服务器上你将会看到这样的消息：

```
Sep 26 13:29:23 zardoz named[21890]: refresh_callback: zone movie.edu/IN:
non-authoritative answer from 192.249.249.3#53
```

在 BIND 8 上，将会显示：

```
Jan  6 11:58:36 zardoz named[544]: Err/TO getting serial# for "movie.edu"
Jan  6 11:58:36 zardoz named-xfer[793]: [192.249.249.3] not authoritative for
     movie.edu, SOA query got rcode 0, aa 0, ancount 0, aucount 0
```

如果这是正确的主名字服务器，那么该服务器应该是这个区的权威。这可能表明该主名字服务器装载区数据有问题，通常是因为区数据文件中有语法错误。与该 主名字服务器的管理员联系，并让他检查系统日志输出是否有语法错误的提示（见后面的问题 5）。

4、增加名字到区数据文件，但忘了增加 PTR 记录

在 DNS 中，由于从主机名到 IP 地址的映射与从 IP 地址到主机名的映射脱节，很容易忘记为新主机增加 PTR 记录。凭直觉知道该增加 A 记录，但是许多使用主机表的人会以为增加地址记录也会对反向映射起作用。事实并不是这样的 —— 你需要增加 PTR 记录，将主机同适当的反向映射区联系起来。

忘记给一个主机增加 PTR 记录常常会导致该主机验证检查失败。例如，如果未指定口令，该主机上的用户将不能 *rlogin* 到其他主机，而且也不能 *rsh* 或 *rcp* 到其他主机。与这些命令会话的服务器需要能够将客户端的 IP 地址映射成域名，以检查 *.rhosts* 和 *hosts.equiv*。这些用户的连接将会引起系统日志记录下这样的条目：

```
Aug 15 17:32:36 terminator inetd[23194]: login/tcp:
       Connection from unknown (192.249.249.23)
```

另外，许多大的FTP文档站点，包括*ftp.uu.net*，对那些IP地址不能映射回域名的主机，会拒绝其匿名FTP访问。*ftp.uu.net*的FTP服务器会发出一条消息，部分内容如下：

```
530- Sorry, we're unable to map your IP address 140.186.66.1 to a hostname
530- in the DNS.  This is probably because your nameserver does not have a
530- PTR record for your address in its tables, or because your reverse
530- nameservers are not registered.  We refuse service to hosts whose
530- names we cannot resolve.
```

该消息十分清楚地说明了你不能使用匿名FTP的原因。然而，其他的FTP站点却懒地去打印这些消息，它们只是简单地拒绝服务。

用*nslookup*很方便就能检查出你是否忘了PTR记录：

```
% nslookup
Default Server:  terminator.movie.edu
Address:  192.249.249.3

> beetlejuice          —查找名字 - 地址映射
Server:  terminator.movie.edu
Address:  192.249.249.3

Name:    beetlejuice.movie.edu
Address:  192.249.249.23

> 192.249.249.23     —现在查找相应的地址 - 名字映射
Server:  terminator.movie.edu
Address:  192.249.249.3

*** terminator.movie.edu can't find 192.249.249.23: Non-existent domain
```

在*249.249.192.in-addr.arpa*的主名字服务器上，快速检查一下*db.192.249. 249*文件，就会知道是否已经把PTR记录加入到区数据文件中了，或该名字服务器是否已被重载。如果有问题的名字服务器是该区的辅名字服务器，那么就检查主名字服务器上的序列号是否已经增加，还有辅名字服务器是否有足够的时间来加载该区数据。

5、配置文件或区数据文件中有语法错误

在名字服务器的配置文件和区数据文件中的语法错误相对来说也很常见（或多或少，取决于管理员的经验）。通常情况下，在配置文件中的错误将导致名字服务器装载一

个或多个区数据失败。某些在 *options* 语句中的拼写错误将导致服务器根本不能启动，并通过 *syslog* 记录如下错误日志消息(BIND 9)：

```
Sep 26 13:39:30 terminator named[21924]: change directory to '/var/name' failed:
file not found
Sep 26 13:39:30 terminator named[21924]: options configuration failed: file not
found
Sep 26 13:39:30 terminator named[21924]: loading configuration: failure
Sep 26 13:39:30 terminator named[21924]: exiting (due to fatal error)
```

BIND 8 名字服务器会记录：

```
Jan  6 11:59:29 terminator named[544]: can't change directory to /var/name: No
     such file or directory
```

注意，当用命令行方式或者在机器启动时启动 *named*，你不会看到错误消息，但 *named* 不会长久保持运行。

如果语法错误是在配置文件中的一个不太重要的行中（如，在 *zone* 语句中），受到影响的将只有那一个区。通常，名字服务器将不能装载那个区的数据（如，你拼错“masters”或区数据文件的文件名，或者你忘了用引号将文件名或域名引起来）。在 BIND 9 中这将产生类似如下的系统日志输出：

```
Sep 26 13:43:03 terminator named[21938]: /etc/named.conf:80: parse error near
'masters'
Sep 26 13:43:03 terminator named[21938]: loading configuration: failure
Sep 26 13:43:03 terminator named[21938]: exiting (due to fatal error)
```

或者在 BIND 8 中：

```
Jan  6 12:01:36 terminator named[841]: /etc/named.conf:10: syntax error near
     'movie.edu'
```

如果区数据文件包含语法错误，但名字服务器成功地装载了区数据，那么对该区中所有数据的查询要么它的响应是非权威的，要么返回一个 SERVFAIL 错误：

```
%  nslookup carrie
Server:  terminator.movie.edu
Address:  192.249.249.3

Non-authoritative answer:
Name:    carrie.movie.edu
Address:  192.253.253.4
```

下面是在 BIND 9 中由引起该问题的语法错误所产生的系统日志消息：

```
Sep 26 13:45:40 terminator named[21951]: error: dns_rdata_fromtext: db.movie.edu:
11: near 'postmanrings2x': unexpected token
Sep 26 13:45:40 terminator named[21951]: error: dns_zone_load: zone movie.edu/IN:
database db.movie.edu: dns_db_load failed: unexpected token
Sep 26 13:45:40 terminator named[21951]: critical: loading zones: unexpected token
Sep 26 13:45:40 terminator named[21951]: critical: exiting (due to fatal error)
```

下面是在 BIND 8 中的错误信息：

```
Jan  6 15:07:46 terminator named[693]: db.movie.edu:11: Priority error
     (postmanrings2x.movie.edu.)
Jan  6 15:07:46 terminator named[693]: master zone "movie.edu" (IN) rejected due
     to errors (serial 1997010600)
```

如果你在区数据文件中查找问题，你将会发现这样的记录：

```
postmanrings2x     IN     MX     postmanrings2x.movie.edu.
```

MX 记录缺少了优先级字段，导致了该错误。

注意，除非你将缺乏权威（当你希望该名字服务器是权威时）视为错误，或者你很认真地检查系统日志文件，否则你可能永远也注意不到这个语法错误！

从 BIND 4.9.4 开始，一个“非法的”主机名可以是语法错误：

```
Jan  6 12:04:10 terminator named[841]: owner name "ID_4.movie.edu" IN (primary)
     is invalid - rejecting
Jan  6 12:04:10 terminator named[841]: db.movie.edu:11: owner name error
Jan  6 12:04:10 terminator named[841]: db.movie.edu:11: Database error near (A)
Jan  6 12:04:10 terminator named[841]: master zone "movie.edu" (IN) rejected
     due to errors (serial 1997010600)
```

然而，BIND 9（如，9.1.0）并没有实现名字检查。将来的版本或许会支持。

6、在区数据文件中域名尾部缺少了圆点

在编辑区数据文件时很容易忘了加尾部的圆点。由于何时要在域名尾部加上圆点各个地方的要求很不一样（不要在配置文件中使用它们，不要在*resolv.conf*文件中使用它们，在区数据文件中要使用它们来覆盖$ORIGIN…），要都记住是很困难的。这些资源记录：

```
zorba          IN     MX     10 zelig.movie.edu
movie.edu      IN     NS     terminator.movie.edu
```

对一双未经训练的眼睛来说，这些记录看上去确实没有什么奇怪的，但是它们可能没有起到它们应该起到的作用。在 *db.movie.edu* 文件中，他们应等价于：

```
zorba.movie.edu.         IN    MX    10 zelig.movie.edu.movie.edu.
movie.edu.movie.edu.     IN    NS    terminator.movie.edu.movie.edu.
```

除非起点（origin）已被明确地修改了。

如果你省略了资源记录数据中域名尾部的圆点（相对于在资源记录的名字中未加尾部圆点），你通常会得到一些奇怪的 NS 或 MX 记录：

```
%   nslookup -type=mx zorba.movie.edu.
Server:  terminator.movie.edu
Address:  192.249.249.3

zorba.movie.edu      preference = 10, mail exchanger
                     = zelig.movie.edu.movie.edu
zorba.movie.edu      preference = 50, mail exchanger
                     = postmanrings2x.movie.edu.movie.edu
```

从 *nslookup* 的输出来看，引起该情况的原因是很清楚的。但是如果你在记录中的域名字段中忘了尾部圆点（就像在上面列出的 *movie.edu* 的 NS 记录那样），要发现你的错误可不是那么容易的了。如果你试图用 *nslookup* 来查询那个记录，你将不会在你认为的域中发现那个名字。拷贝出你的名字服务器的转储或许会帮助你把它找出来：

```
$ORIGIN edu.movie.edu.
movie    IN    NS    terminator.movie.edu.movie.edu.
```

$ORIGIN 行与其他行相比，看上去显得特别奇怪。

7、遗失根线索数据

由于某种原因，如果你在你的主机上忘了安装根线索文件，或者你不小心把它删了，你的名字服务器将不能解析它权威数据之外的名字。用 *nslookup* 这种情况很容易识别，但要小心使用完全的、以圆点结尾的域名，否则搜索列表可能会引起误导性的失败：

```
% nslookup
Default Server: terminator.movie.edu
Address: 192.249.249.3

> ftp.uu.net.        —查找你的名字服务器权威数据以外的一个名字，导致 SERVFAIL 错误
Server: terminator.movie.edu
Address: 192.249.249.3

*** terminator.movie.edu can't find ftp.uu.net.: Server failed
```

查询你的名字服务器的权威数据中的名字将返回一个响应：

```
> wormhole.movie.edu.
Server: terminator.movie.edu
Address: 192.249.249.3

Name:   wormhole.movie.edu
Addresses: 192.249.249.1, 192.253.253.1

> ^D
```

要证实你对根线索数据遗失的怀疑，检查系统日志输出，看有没有像这样的错误：

```
Jan  6 15:10:22 terminator named[764]: No root nameservers for class IN
```

你应该记得类 1 就是 IN 或 Internet 类。这一错误表明由于没有根线索数据可用，因而找不到根名字服务器。

在 BIND 9 中你就不会碰到这样的问题，因为它自带有根线索。

8、网络连接丢失

尽管今天的 Internet 比其前身 ARPAnet 网可靠得多，但网络不可用的情况相对而言仍然是很普遍的。如果不揭开它的面纱，在调试输出中来回寻找原因，这些错误通常看上去就像是性能很差：

```
% nslookup nisc.sri.com.
Server: terminator.movie.edu
Address: 192.249.249.3

*** Request to terminator.movie.edu timed out ***
```

但是，如果打开了名字服务器的调试，你会发现你的名字服务器其实很正常的。它

接收解析器的查询请求，发送出必要的查询，并耐心地等待响应。它只是没有收到响应而已。这里是 BIND 8 中可能的调试输出结果：

```
Debug turned ON, Level 1
```

为了查找 *nisc.sri.com* 的 IP 地址，*nslookup* 向我们的本地名字服务器发出了第一个查询请求。然后查询被转发给另一名字服务器，如果没收到响应，就重发给另一个不同的名字服务器：

```
datagram from [192.249.249.3].1051, fd 5, len 30
req: nlookup(nisc.sri.com) id 18470 type=1 class=1
req: missed 'nisc.sri.com' as 'com' (cname=0)
forw: forw -> [198.41.0.4].53 ds=7 nsid=58732 id=18470 0ms retry 4 sec
resend(addr=1 n=0) -> [128.9.0.107].53 ds=7 nsid=58732 id=18470 0ms
```

现在 *nslookup* 开始不耐烦了，它再一次查询本地服务器。注意，它使用的是相同的源端口。本地服务器忽略该重复查询并再次尝试转发该查询两次：

```
datagram from [192.249.249.3].1051, fd 5, len 30
req: nlookup(nisc.sri.com) id 18470 type=1 class=1
req: missed 'nisc.sri.com' as 'com' (cname=0)
resend(addr=2 n=0) -> [192.33.4.12].53 ds=7 nsid=58732 id=18470 0ms
resend(addr=3 n=0) -> [128.8.10.90].53 ds=7 nsid=58732 id=18470 0ms
```

nslookup 再次查询本地名字服务器，服务器发出了更多的查询请求：

```
datagram from [192.249.249.3].1051, fd 5, len 30
req: nlookup(nisc.sri.com) id 18470 type=1 class=1
req: missed 'nisc.sri.com' as 'com' (cname=0)
resend(addr=4 n=0) -> [192.203.230.10].53 ds=7 nsid=58732 id=18470 0ms
resend(addr=0 n=1) -> [198.41.0.4].53 ds=7 nsid=58732 id=18470 0ms
resend(addr=1 n=1) -> [128.9.0.107].53 ds=7 nsid=58732 id=18470 0ms
resend(addr=2 n=1) -> [192.33.4.12].53 ds=7 nsid=58732 id=18470 0ms
resend(addr=3 n=1) -> [128.8.10.90].53 ds=7 nsid=58732 id=18470 0ms
resend(addr=4 n=1) -> [192.203.230.10].53 ds=7 nsid=58732 id=18470 0ms
resend(addr=0 n=2) -> [198.41.0.4].53 ds=7 nsid=58732 id=18470 0ms
Debug turned OFF
```

在 BIND 9 名字服务器上，调试级别 1 给出的信息非常少。但是你仍然能够看到名字服务器在不断地查询 *nisc.sri.com*。

```
Sep 26 14:33:27.486 client 192.249.249.3#1028: query: nisc.sri.com A
Sep 26 14:33:27.486 createfetch: nisc.sri.com. A
Sep 26 14:33:32.489 client 192.249.249.3#1028: query: nisc.sri.com A
Sep 26 14:33:32.490 createfetch: nisc.sri.com. A
```

```
Sep 26 14:33:42.500 client 192.249.249.3#1028: query: nisc.sri.com A
Sep 26 14:33:42.500 createfetch: nisc.sri.com. A
Sep 26 14:34:02.512 client 192.249.249.3#1028: query: nisc.sri.com A
Sep 26 14:34:02.512 createfetch: nisc.sri.com. A
```

在更高级别的调试中，你能明显地看到超时，但是BIND 9.1.0仍然没有给出服务器尝试查询的远程名字服务器的地址。

从BIND 8的调试输出中你可以抽出你的名字服务器试图查询的名字服务器的IP地址列表，然后检查到这些服务器的网络连接。奇怪的是，*ping*不会比你的名字服务器幸运多少：

```
%  ping 198.41.0.4 -n 10     －ping 要查询的第一个名字服务器
PING 198.41.0.4: 64 byte packets

----198.41.0.4 PING Statistics----
10 packets transmitted, 0 packets received, 100% packet loss
%  ping 128.9.0.107 -n 10    －ping 要查询的第二个名字服务器
PING 128.9.0.107: 64 byte packets

----128.9.0.107 PING Statistics----
10 packets transmitted, 0 packets received, 100% packet loss
```

如果真的*ping*不通，你应该检查远程名字服务器是否确实在运行。你或许还应检查你的Internet防火墙是不是不小心禁止了你的名字服务器的查询。如果你最近才升级到BIND 8或者9，看看第十一章中“选读：理解BIND 8或9与包过滤防火墙”，看它是否适用于你。

如果还是*ping*不通，剩下的事情就是确定网络中的断点。像*traceroute*和*ping*这样的工具的记录路由选项在确定问题是出在你的网络、目的网络还是在中途某处时，是很有帮助的。

当寻找断点时，你也应该利用你自己的常识来判断。例如，在这个例子中，你的名字服务器尝试查询的远程名字服务器均是根名字服务器。（你可能已经有了它们的PTR缓存记录，所以你能找到它们的域名。）不过不大可能每一个根的本地网络都崩溃了，也不大可能互联网的主干网会完全瘫痪。可能导致这一故障（也就是你的网络失去了与互联网的连接）的最简单的原因就是最有可能的原因。

9、遗失子域授权

尽管登记员会尽其最大的努力来尽快处理你的请求，可能也会需要一两天时间才能把你的子域授权信息放到它的父区的名字服务器上。如果你的父区不是通用顶级域，所需的时间可能会更难确定。有的父域处理速度很快而且负责，有的则较慢且步调不够协调一致。然而，就像在现实生活中那样，你只有承受这一切。

直到你的授权信息出现在你的父区的名字服务器中，你的名字服务器才能够查询 Internet 域名空间中的数据，但是 Internet 中（在你的域之外）的其他人将不知道如何查询你的名字空间中的数据。

这意味着，即使你能够向你的域外发送电子邮件，收信人也不能回复你。更糟糕的是，没有人能够通过域名来 *telnet*、*ftp* 或 *ping* 到你的主机。

记住，这对任何你运行的 *in-addr.arpa* 区效果都是一样的。直到父区将授权加到你的服务器，Internet 上的名字服务器将不能反向映射你的网络上的地址。

要确定你的区授权是否已加到父区的名字服务器中，可查询父名字服务器中你的区的 NS 记录。如果父名字服务器有你的区 NS 记录，任何在 Internet 上的名字服务器均能找到你：

```
%  nslookup
Default Server:  terminator.movie.edu
Address:  192.249.249.3

>   server a.root-servers.net.   －查询一个根名字服务器
Default Server:  a.root-servers.net
Address:  198.41.0.4

> set norecurse                  －指示服务器不根据自己的数据回答
> set type=ns                    －查找 249.249.192.in-addr.arpa 的 NS 记录
> 249.249.192.in-addr.arpa.
Server:  a.root-servers.net
Address:  198.41.0.4

*** a.root-servers.net can't find 249.249.192.in-addr.arpa.: Non-existent domain
```

这里，授权显然还没有被加进去。你可以耐心地等待，如果你请求父区授权已经很长时间了，还是没有结果，你可以与你的父区的管理员联系，问问发生了什么事。

10、不正确的子域授权

在 Internet 中，不正确的子域授权是另一常见故障。保持授权信息的实时更新需要人为地介入 —— 告诉你的父区的系统管理员关于你的权威名字服务器的更改情况。由于系统管理员们做了修改之后没有告诉它们的父区，结果授权信息常常变得不准确。太多的管理员认为设置授权只是个一次性的工作 :他们只是在建立他们的区时，告诉父区哪些名字服务器是权威服务器，之后他们再不谈此事。甚至在父亲节时他们也不打个电话。

管理员可能会增加一个新的名字服务器，而让另一个退役，并且修改第三个服务器的 IP 地址，所有这些都没有告诉其父区的管理员。渐渐地，被父区正确授权的名字服务器的数目逐渐减少。如果幸运的话，这只会使解析时间变长，因为进行查询的名字服务器不得不花费一番功夫才能找出一个该区的权威名字服务器。如果授权信息变得太过时了，为了维护的需要而将最后一个权威名字服务器主机关掉，那么在该区中的信息将不可访问。

如果你怀疑从你的父区到你的区、从你的区到你的子区，或者从一个远程区到它的子区的授权坏了，可以用 *nslookup* 来检查：

```
% nslookup
Default Server:  terminator.movie.edu
Address:  192.249.249.3

>   server a.root-servers.net.      —把服务器设成你怀疑有错误授权的
                                    —父区的名字服务器
Default Server:  a.root-servers.net
Address:  198.41.0.4
> set type=ns                       —查找问题区的 NS 记录
>  hp.com.
Server:  a.root-servers.net
Address:  198.41.0.4

Non-authoritative answer:
hp.com          nameserver = RELAY.HP.COM
hp.com          nameserver = HPLABS.HPL.HP.COM
hp.com          nameserver = NNSC.NSF.NET
hp.com          nameserver = HPSDLO.SDD.HP.COM

Authoritative answers can be found from:
hp.com          nameserver = RELAY.HP.COM
hp.com          nameserver = HPLABS.HPL.HP.COM
hp.com          nameserver = NNSC.NSF.NET
hp.com          nameserver = HPSDLO.SDD.HP.COM
```

```
RELAY.HP.COM    internet address = 15.255.152.2
HPLABS.HPL.HP.COM       internet address = 15.255.176.47
NNSC.NSF.NET    internet address = 128.89.1.178
HPSDLO.SDD.HP.COM       internet address = 15.255.160.64
HPSDLO.SDD.HP.COM       internet address = 15.26.112.11
```

假设你怀疑到 *hpsdlo.sdd.hp.com* 的授权不正确。你现在向 *psdlo.sdd.hp.com* 查询 *hp.com* 区中的数据（例如，*hp.com* 的 SOA 记录），并检查其响应：

```
>   server hpsdlo.sdd.hp.com.
Default Server:  hpsdlo.sdd.hp.com
Addresses:  15.255.160.64, 15.26.112.11

>   set norecurse
>   set type=soa
>  hp.com.
Server:  hpsdlo.sdd.hp.com
Addresses:  15.255.160.64, 15.26.112.11

Non-authoritative answer:
hp.com
        origin = relay.hp.com
        mail addr = hostmaster.hp.com
        serial = 1001462
        refresh = 21600 (6 hours)
        retry   = 3600 (1 hour)
        expire  = 604800 (7 days)
        minimum ttl = 86400 (1 day)

Authoritative answers can be found from:
hp.com          nameserver = RELAY.HP.COM
hp.com          nameserver = HPLABS.HPL.HP.COM
hp.com          nameserver = NNSC.NSF.NET
RELAY.HP.COM    internet address = 15.255.152.2
HPLABS.HPL.HP.COM       internet address = 15.255.176.47
NNSC.NSF.NET    internet address = 128.89.1.178
```

如果 *hpsdlo.sdd.hp.com* 确实是 *hp.com* 的权威，它将会用一个权威的应答来响应。*hp.com*区的管理员可以告诉你*hpsdlo.sdd.hp.com*是否是*hp.com*的权威名字服务器，所以那个人就是你应该联系的人。

这一问题的另一个常见症状是“残缺的服务器”（lame server）错误消息：

```
Oct 1 04:43:38 terminator named[146]: Lame server on '40.234.23.210.in-addr.arpa'
(in '210.in-addr.arpa'?): [198.41.0.5].53 'RS0.INTERNIC.NET': learnt(A=198.41.0.
21,NS=128.63.2.53)
```

该错误消息的含义是：为了查询在域 *210.in-addr.arpa* 中的一个名字，具体地说就

是*40.234.23.210.in-addr.arpa*，你的名字服务器被在128.63.2.53上的名字服务器指给在198.41.0.5上的名字服务器。来自198.41.0.5上服务器的响应指出，事实上它不是*210.in-addr.arpa*的权威，因此要么是128.63.2.53给你的授权信息错了，要么是198.41.0.5上服务器的配置不对。

11、resolv.conf中的语法错误

尽管*resolv.conf*文件的语法简单，但是编辑时人们偶尔也会犯错误。而且，不幸的是，文件*resolv.conf*中有语法错误的行只是被解析器默默地忽略掉了。结果通常是你打算配置的某些部分没有生效：你的本地域名或搜索列表设置不正确，或者是解析器没有像你所配置的那样去查询一个你指定的名字服务器；依赖于搜索列表的命令不能工作，你的解析器不去查询正确的名字服务器，或者它根本就不去查询名字服务器。

检查你的*resolv.conf*文件是否起到你所希望的效果，最简单的方法就是运行*nslookup*。当你键入*set all*时，*nslookup*将会友好地报告从*resolv.conf*得来的本地域名和搜索列表，加上它正在查询的名字服务器，就像在第十二章中我们向你演示的那样：

```
% nslookup
Default Server:  terminator.movie.edu
Address:  192.249.249.3

> set all
Default Server:  terminator.movie.edu
Address:  192.249.249.3

Set options:
  nodebug         defname          search         recurse
  nod2            novc             noignoretc     port=53
  querytype=A     class=IN         timeout=5      retry=4
  root=ns.nic.ddn.mil.
  domain=movie.edu
  srchlist=movie.edu

>
```

检查*set all*命令的输出，看它是否是你所希望在你的*resolv.conf*文件中设置的值。例如，如果你在*resolv.conf*文件中设置了*search fx.movie.edu movie.edu*，你将希望在输出结果中看到：

```
domain=fx.movie.edu
srchlist=fx.movie.edu/movie.edu
```

如果你没有看到你所希望的结果，请仔细检查*resolv.conf*文件。如果未发现任何明显的不妥之处，请查找非打印字符（例如，使用*vi*的*set list*命令）。特别要注意结尾的空格，在较老版本的解析器中，域名结尾的空格将把本地域名设置成包含一个空格。当然，真正的顶级域名实际上不会以空格结尾，所以所有不带尾部圆点的名字查询都将会失败。

12、未设置本地域名

未设置本地域名是另一个古老的常见失误。你可以通过将主机名设为主机的全称域名来隐含地设置本地域名，或者在*resolv.conf*文件中明确地给出本地域名。未设置本地域名的特征是直观的：在命令中使用单标号名字（或者缩写域名）的人们将得不到任何满意的结果：

```
%  telnet br
br: No address associated with name
%  telnet br.fx
br.fx: No address associated with name
%  telnet br.fx.movie.edu
Trying...
Connected to bladerunner.fx.movie.edu.
Escape character is '^]'.

HP-UX bladerunner.fx.movie.edu A.08.07 A 9000/730 (ttys1)
login:
```

你可以用*nslookup*来检查，就像当你怀疑在*resolv.conf*文件中有语法错误时所做的那样：

```
%  nslookup
Default Server:  terminator.movie.edu
Address:  192.249.249.3

>  set all
Default Server:  terminator.movie.edu
Address:  192.249.249.3

Set options:
  nodebug         defname         search          recurse
  nod2            novc            noignoretc      port=53
  querytype=A     class=IN        timeout=5       retry=4
```

```
root=ns.nic.ddn.mil.
domain=
srchlist=
```

注意，这里既未设置本地域名也未设置搜索列表。你也可以通过打开名字服务器的调试选项来找出该故障。（当然，这需要访问名字服务器，而该服务器可能就运行在受故障影响的主机上。）以下是在BIND 9名字服务器尝试了那些*telnet*命令之后可能的调试输出结果：

```
Sep 26 16:17:58.824 client 192.249.249.3#1032: query: br A
Sep 26 16:17:58.825 createfetch: br. A
Sep 26 16:18:09.996 client 192.249.249.3#1032: query: br.fx A
Sep 26 16:18:09.996 createfetch: br.fx. A
Sep 26 16:18:18.677 client 192.249.249.3#1032: query: br.fx.movie.edu A
```

在BIND 8名字服务器上，你可能会看到如下结果：

```
Debug turned ON, Level 1

datagram from [192.249.249.3].1057, fd 5, len 20
req: nlookup(br) id 27974 type=1 class=1
req: missed 'br' as '' (cname=0)
forw: forw -> [198.41.0.4].53 ds=7 nsid=61691 id=27974 0ms retry 4 sec

datagram from [198.41.0.4].53, fd 5, len 20
ncache: dname br, type 1, class 1
send_msg -> [192.249.249.3].1057 (UDP 5) id=27974

datagram from [192.249.249.3].1059, fd 5, len 23
req: nlookup(br.fx) id 27975 type=1 class=1
req: missed 'br.fx' as '' (cname=0)
forw: forw -> [128.9.0.107].53 ds=7 nsid=61692 id=27975 0ms retry 4 sec

datagram from [128.9.0.107].53, fd 5, len 23
ncache: dname br.fx, type 1, class 1
send_msg -> [192.249.249.3].1059 (UDP 5) id=27975

datagram from [192.249.249.3].1060, fd 5, len 33
req: nlookup(br.fx.movie.edu) id 27976 type=1 class=1
req: found 'br.fx.movie.edu' as 'br.fx.movie.edu' (cname=0)
req: nlookup(bladerunner.fx.movie.edu) id 27976 type=1 class=1
req: found 'bladerunner.fx.movie.edu' as 'bladerunner.fx.movie.edu'
     (cname=1)
ns_req: answer -> [192.249.249.3].1060 fd=5 id=27976 size=183 Local
Debug turned OFF
```

将这一调试输出与在第十三章中搜索列表应用程序所产生的调试输出相比较。只查

询了用户输入的名字，而没有在这个名字结尾添加任何域名。很明显，没有使用搜索列表。

13、响应来自预料之外

在 DNS 新闻组中，我们越来越多地看到的一个问题是“响应来自预料之外。”这曾被称为火星的响应：该响应来自一个与你的服务器查询所发往的地址不同的地址。当BIND名字服务器发送查询到一个远程服务器时，BIND谨慎地保证应答仅来自那个服务器的 IP 地址。这有助于减少接受欺骗应答的可能性。BIND 对它自己一样严格要求：BIND 服务器将尽可能地通过它收到查询请求的同一网络接口发回响应。

这里是当收到一个可能的意外响应时你会看到的错误消息：

```
Mar  8 17:21:04 terminator named[235]: Response from unexpected source ([205. 199.
4.131].53)
```

这可能有两种含义：或者某人正在试图欺骗你的名字服务器，不过更有可能是你的查询请求被发往一个较老的 BIND 服务器，或是一个服务器不够认真，没有从其所收到查询请求的同一接口发回响应。

版本升级带来的问题

随着 BIND 8 和现在 BIND 9 的发布，许多 Unix 操作系统都更新了它们的解析器和名字服务器。然而，当升级到新版本之后，最近的 BIND 版本的某些特性在你看来似乎是错误的。我们将试着给你一些思路，你可能会在升级之后的名字服务器和名字服务中注意到这些变化。

解析器的行为

像我们在第六章中描述的那样，解析器默认搜索列表的变化对你的用户来说可能是个问题。回忆一下，把本地域名设置为*fx.movie.edu*，你的默认搜索列表将不再包括*movie.edu*。因此，对某些用户来说，他们已习惯于使用像*telnet db.personnel*这样的命令，不完整的域名会被展开成*db.personnel.movie.edu*，在使用这些命令时将会

失败。要解决这一问题，你可以使用 *search* 指令来明确定义一个搜索列表，它包括你的本地域名的父亲。或者只是告诉你的用户去适应新的操作。

名字服务器的行为

在4.9版之前，BIND名字服务器乐于从它的主名字服务器上的任何区数据文件中加载任何区的数据。如果你将该服务器配置为 *movie.edu* 的主名字服务器并告诉它 *movie.edu* 的数据在 *db.movie.edu* 中，你能够在 *db.movie.edu* 中放入 *hp.com* 的数据，而你的名字服务器将会装载 *hp.com* 的资源记录到缓冲区中。有些书甚至建议将所有的 *in-addr.arpa* 区数据放到一个文件中。

所有BIND 4.9以及后续版本的名字服务器会忽略区数据文件中任何“ 区以外的 ”资源记录。所以，如果你把所有你的 *in-addr.arpa* 域的 PTR 记录放入一个文件中，并用一个 *zone* 语句或 *primary* 指令来装载它，名字服务器将忽略所有不在命名区中的记录。当然，那将意味着装载丢失的 PTR 记录和失败的 *gethostbyaddr()* 调用。

BIND 会在系统日志中记录它忽略了某些记录。在 BIND 9 中，日志消息看上去像这样：

```
Sep 26 13:48:19 terminator named[21960]: dns_master_load: db.movie.edu:16:
ignoring out-of-zone data
```

在 BIND 8 中则像下面这样：

```
Jan  7 13:58:01 terminator named[231]: db.movie.edu:16: data "hp.com" outside zone
     "movie.edu" (ignored)
Jan  7 13:58:01 terminator named[231]: db.movie.edu:17: data "hp.com" outside zone
     "movie.edu" (ignored)
```

解决方案是每个区使用一个区文件和一个 *zone* 语句或 *primary* 指令。

互操作性和版本问题

随着 BIND 9 的出现和微软 DNS Server 的引入，越来越多的互操作性问题出现在名字服务器之间。也许有些问题是针对某一个 BIND 版本或底层操作系统的，但其中许多问题是容易发现并改正的，如果我们不讲讲这些问题的话，那可就是我们的疏忽了。

由于专有的 WINS 记录导致区数据传送失败

当微软 DNS Server 被配置成在给定区中找不到名字时去查询 WINS 服务器时，它将在区数据文件中插入一个特殊的记录。该记录如下所示：

```
@   IN   WINS   &IP address of WINS server
```

不幸的是在 IN 类中，WINS 不是标准记录类型。结果，如果有 BIND 的辅名字服务器传送该区的数据，它们会被阻塞在 WINS 记录处并拒绝装载该区：

```
May 23 15:58:43 terminator named-xfer[386]: "fx.movie.edu IN 65281" - unknown type
 (65281)
```

该问题的解决办法就是设置微软DNS Server在传送区之前把它专有的记录过滤掉。你可以通过在 DNS Manager 屏幕的左边选中该区，右击它，选择 Properties（属性）来完成这一任务。在导出的 Zone Properties（区属性）窗口中，点击 WINS Lookup（WINS 查询）标签，如图 14-1 所示。

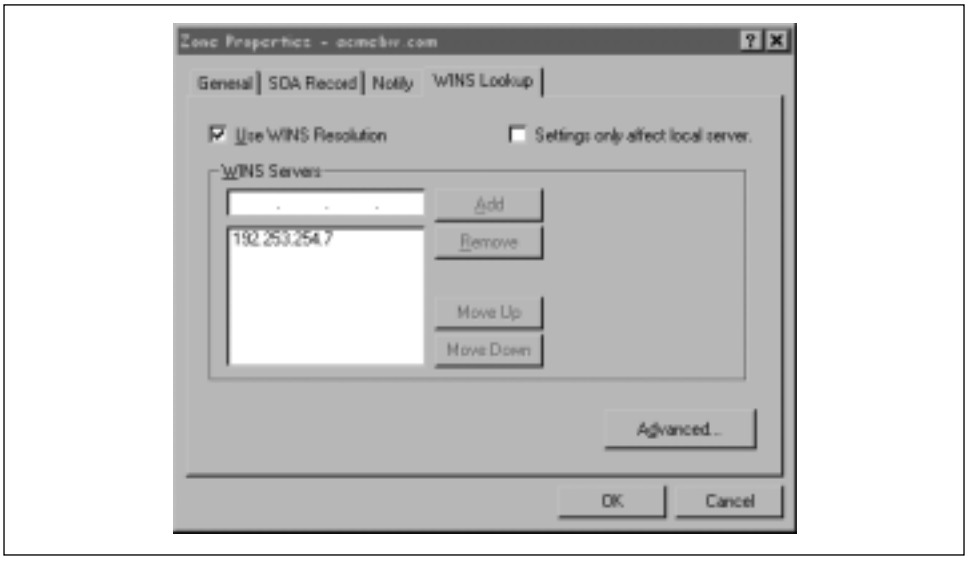


图 14-1　Zone Properties 窗口

选中 Settings only affect local server（设置只影响本地服务器）选项将过滤掉那个区的 WINS 记录。然而即使是微软 DNS Server 的辅名字服务器也看不到那个记录，即使它们是可以使用这个记录的。

名字服务器报告“对 SOA MNAME 无 NS 记录”

你只会在 BIND 8.1 服务器中看到这一错误：

```
May 8 03:44:38 terminator named[11680]: no NS RR for SOA MNAME "movie.edu" in
     zone "movie.edu"
```

8.1 版服务器对 SOA 记录中的第一个字段很敏感。记得在第四章中我们说过的，按惯例该字段是该区的主名字服务器的域名。BIND 8.1 将假定它确实是的，并且检查相应的 NS 记录，该记录将该区域名指向那个字段中的服务器。如果不存在这样的 NS 记录，BIND 将发出那个错误消息。这也会导致 NOTIFY 消息工作不正常。该问题的解决方案是将 MNAME 字段值改成在 NS 记录中名字服务器的域名，或者将版本升级到 BIND 8 的更新版本。由于 BIND 8.1 已经很老了，所以升级是一个很好的选择。这一检查在 BIND 8.1.1 中被去除。

名字服务器报告“打开的文件太多”

在有多个 IP 地址或允许用户可打开的最大文件数较少的主机上，BIND 将报告：

```
Dec 12 11:52:06 terminator named[7770]: socket(SOCK_RAW): Too many open files
```

由于 BIND 试图 *bind( )* 到并监听主机上的每一个网络接口，它可能会用光文件描述符。这种情况特别常见于那些大量使用虚拟接口的主机上，这类主机通常用于支持 Web 的虚拟主机服务。对该问题的两种可能的解决方案是：

- 使用基于名字的虚拟主机，它不需要额外的 IP 地址。
- 用 *listen-on* 子句将 BIND 8 或者 9 名字服务器配置成只监听一个或几个主机的网络接口。如果 *terminator.movie.edu* 是有这一问题的主机，下面的语句：

  ```
  options {
       listen-on { 192.249.249.3; };
  };
  ```

  将告诉在 *terminator.movie.edu* 上的 *named*：只要 *bind( )* 到 IP 地址 192.249.249.3。
- 重新配置你的操作系统使之允许一个进程同时打开更多的文件描述符。

解析器报告“查询 PTR，却找到了 CNAME”

这是与 BIND 的严密性有关的另一个问题。在某些查询中，解析器将记录日志：

```
Sep 24 10:40:11 terminator syslog: gethostby*.getanswer: asked for
     "37.103.74.204.in-addr.arpa IN PTR", got type "CNAME"
Sep 24 10:40:11 terminator syslog: gethostby*.getanswer: asked for
     "37.103.74.204.in-addr.arpa", got "37.32/27.103.74.204.in-addr.arpa"
```

这里所发生的事是解析器要求名字服务器把IP地址204.74.103.37反向映射到域名。服务器这么做了，但是在处理过程中发现 *37.103.74.204.in-addr.arpa* 实际上是 *37.32/27.103.74.204.in-addr.arpa* 的别名。那几乎是肯定的，因为运行 *103.74.204.in-addr.arpa* 的人正使用我们在第九章中所描述的方案去授权他们的部分名字空间。然而，BIND 4.9.3-BETA 版解析器不理解这种情况，而将其标记为一个错误，认为它没有得到域名或它所查的类型。并且，信不信由你，一些操作系统把 BIND 4.9.3-BETA 解析器当做它们的系统解析器。

解决这个问题的惟一方法就是将 BIND 解析器升级到一个更新的版本。

由于禁止使用 UDP 校验和而导致名字服务器启动失败

在一些运行 SunOS 4.1.X 的主机上，你会看到这个错误：

```
Sep 24 10:40:11 terminator named[7770]: ns_udp checksums NOT turned on: exiting
```

在该系统中，*named* 要检查以确保 UDP 校验和是打开的，但却发现不是的，因此退出。*named* 如此坚持要求打开 UDP 校验和运算是有原因的：它大量使用 UDP，需要这些 UDP 数据报正确到达目的地。

该问题的解决方法是在你的系统中打开 UDP 校验和。在 BIND 软件包目录 *shres/sunos/INSTALL* 和 *shres/sunos/ISSUES* 中（BIND 4）或在目录 *src/port/sunos/shres/ISSUES* 中（BIND 8）有关于该问题的文档资料。

SunOs 解析器已配置，但主机未使用 DNS

这个问题是与实现有关的。某些 SunOs 4 主机的管理员用 *resolv.conf* 文件来配置他们的解析器并天真地假设 *ping*、*telnet* 及类似的程序应该会正常工作。然而，在第

六章中我们讨论过SunOs 4是如何实现解析器的（在*ypserv*中，你应该还记得）。如果该主机没有运行NIS，配置解析器将不会有效果。管理员要么不得不至少设置一个空的主机映射，要么就替换解析器例程。关于这两个选项的详细说明，见第六章“Sun的SunOS 4.x”一节。

其他名字服务器没有缓存你的否定响应

你需要有一双敏锐的眼睛才能注意到这一问题。如果你运行的是BIND 8，你将不得不关掉引起该问题的一个重要特性。不过，如果你正在运行BIND 9，该特性的默认设置是关闭的。如果你运行的是BIND 8或者9名字服务器并且其他解析器和服务器似乎忽略了你的服务器缓存的否定响应，你可能是关掉了*auth-nxdomain*。

*auth-nxdomain*是一个*options*子语句，它告诉BIND 8或者9服务器将缓存的否定响应标记为权威的，即使它们不是。那就是说，如果你的名字服务器已经缓存了来自*movie.edu*的权威名字服务器的*titanic.movie.edu*并不存在的消息，*auth-nxdomain*选项告诉你的服务器将那个缓存的响应传递给查询它的解析器和服务器，就好像它是*movie.edu*的权威名字服务器一样。

有时需要这么做是因为某些名字服务器要检查以确保否定响应（如，NXDOMAIN的返回码或带NOERROR返回码的无记录响应）被标记为权威的。在否定缓存被引入之前，否定响应必须是权威的，因此这是一个明智的检查。然而，随着否定缓存的出现，否定响应可能来自缓存。可是，为确保那些旧服务器不会忽略这样的响应或认为它们是错误，BIND 8和9使你错误地将那些响应标记为权威的。事实上那是BIND 8默认的行为，因此你应该看不到远程的查询者会忽略你的服务器的否定响应，除非你关闭了*auth-nxdomain*选项。另一方面，默认情况下BIND 9名字服务器的*auth-nxdomain*是关闭的，因此，如果你没有修改过配置文件，查询者会忽略它们的响应。

未设置TTL

在第四章我们提到，BIND 8.2发布之前出版了RFC 2308。RFC 2308将SOA记录最后一个字段的语意改成了否定缓存TTL，并且引入了一个新的控制语句$TTL来设置区数据文件默认的TTL。

如果你升级到比 BIND 8.2 更新的 BIND 8 版本，而没有在你区数据文件中添加必要的 $TTL 控制语句，你会在名字服务器的系统日志文件中看到如下信息：

```
Sep 26 19:34:39 terminator named[22116]: Zone "movie.edu" (file db.movie.edu): No
default TTL ($TTL <value>) set, using SOA minimum instead
```

BIND 8 非常宽宏大量，它假设你没有阅读过 RFC 2308，并且把 SOA 的最后一个字段当做区的默认 TTL 和它的否定缓存 TTL。然而，BIND 9 却没有这么好：

```
Sep 26 19:35:54 terminator named[22124]: dns_master_load: db.movie.edu:7: no TTL
specified
Sep 26 19:35:54 terminator named[22124]: dns_zone_load: zone movie.edu/IN:
database db.movie.edu: dns_db_load failed: no ttl
Sep 26 19:35:54 terminator named[22124]: loading zones: no ttl
Sep 26 19:35:54 terminator named[22124]: exiting (due to fatal error)
```

所以在升级到 BIND 9 以前一定要确定添加了必要的 $TTL 控制语句。

TSIG 错误

就像我们在第十一章中讲的，事务签名需要时间同步和密钥同步（事务两端要有同样的密钥和密钥名）才能工作。下面是如果你失去了时间同步或者使用了不同的密钥和密钥名可能出现的一些错误。

首先，在 BIND 8 上，如果你配置了 TSIG，但是在你的主名字服务器和辅名字服务器之间有太大的时钟偏差，你会看到下面的错误：

```
Sep 27 10:47:49 wormhole named[22139]: Err/TO getting serial# for "movie.edu"
Sep 27 10:47:49 wormhole named-xfer[22584]: SOA TSIG verification from server
[192.249.249.3], zone movie.edu: message had BADTIME set (18)
```

这里，你的名字服务器试图检查在 *terminator.movie.edu*(192.249.249.3)上 *movie.edu* 区的序列号。因为 *wormhole.movie.edu* 的时钟和响应所标记的时钟有大于 10 分钟的偏差，所以来自 *terminator.movie.edu* 的响应没有得到确认。*Err/TO* 信息是带 TSIG 签名的响应所确认的失败的副产品。

如果你在事务的两端使用了不同的密钥名，即使密钥名所指的数据是相同的，你还是会在你的 BIND 8 名字服务器中看到如下错误：

```
Sep 27 12:02:44 wormhole named-xfer[22651]: SOA TSIG verification from server
[209.8.5.250], zone movie.edu: BADKEY(-17)
```

这次，因为认证者没有找到TSIG记录中指定名字的密钥，所以没有检查带TSIG签名的响应。如果密钥名匹配但是指向不同的数据，你可以看到同样的错误。

通常，对于TSIG错误，BIND 9给出的信息非常少，在调试级别3，对前面两种情况它只是报告：

```
Sep 27 13:35:42.804 client 192.249.249.1#1115: query: movie.edu SOA
Sep 27 13:35:42.804 client 192.249.249.1#1115: error
```

故障症状

不幸的是，某些问题不是像我们所列出的那些问题那样容易识别。你会遇到某些异常情况但又找不出直接的原因，这通常是因为有多种问题都能引起你所看到的症状。对于这样的案例，我们将给出这些问题的常见原因和区分它们的方法。

无法查找本地名字

当*telnet*或*ftp*这样的程序无法查找本地名字时，要做的第一件事就是用*nslookup*或者*dig*来尝试查询同样的名字。当我们说“同样的名字”时，意思是字面上完全相同——不加域和尾部圆点，如果用户没有输入的话。不会查询与用户所查询的不同的服务器。

有时，用户敲错了名字或不了解搜索列表是如何工作的，那只需指出即可。偶尔，你会发现真正的主机配置错误：

- 在*resolv.conf*中的语法错误（本章前面提到的“潜在故障列表”中的问题11）
- 未设置默认域名（问题12）

你可以用*nslookup*的*set all*命令来检查这两个问题。

如果*nslookup*指出名字服务器有故障而不是主机配置有问题，那么就检查与名字服务器类型相关的问题。如果该名字服务器是该区的主名字服务器，但它响应的数据与你想像中的不一样：

- 检查包含有问题数据的区数据文件，并检查装载了该文件的服务器（问题 2）。数据库转储能确切地告诉你数据是否已被装载了。
- 检查配置文件和相关的区数据文件中的语法错误（问题5）。检查名字服务器的系统日志输出中有关这些错误的指示。
- 确保记录有尾部圆点，如果需要的话（问题 6）。

如果该名字服务器是该区的辅名字服务器，你首先应该检查它的主名字服务器是否有正确的数据。如果主名字服务器中的数据是正确的，而辅名字服务器却不是：

- 确信你已经增加了主名字服务器上的序列号（问题 1）。
- 检查辅名字服务器更新区数据的问题（问题 3）。

当然，如果主名字服务器中的数据不正确，那么就在主名字服务器上诊断问题。

如果有问题的服务器是一个只缓存名字服务器：

- 确信它有根线索数据（问题 7）。
- 检查你的父区到你的区的授权存在并且是正确的（问题 9 和问题 10）。记得对一个只缓存服务器，你的区看上去和任何远程区一样。即使该服务器运行在你的区内，只缓存服务器也必须能够从你的父区服务器上找到一个你的区的权威服务器。

无法查找远程名字

如果本地查询成功但无法查找本地区之外的域名，在这种情况下有一组不同的问题需要检查：

- 首先，你是不是刚刚建起你的服务器？你可能遗漏了根线索数据（问题 7）。
- 你能*ping*到远程区的名字服务器吗？你可能由于网络连接丢失而不能到达远程区的服务器（问题 8）。
- 远程区是新建的吗？或许它的授权信息还没有出现（问题9）。或者该远程区的授权信息由于疏忽而发生了错误或是过时了（问题 10）。

- 在远程区的服务器上确实有该域名吗（问题2）？在所有的远程区服务器上都有吗（问题1和问题3）？

错误的或不一致的应答

如果你查询一个本地名字时得到错误响应或不一致的应答，这取决于你查询的是哪个服务器或查询的时间，首先检查你的服务器之间的同步情况：

- 对该区，它们都拥有相同的序列号吗？在做了更改之后，你是否忘了增加主名字服务器上的序列号（问题1）？如果你增加了在主名字服务器上的序列号，那么所有的服务器可能都会有相同的序列号，但它们会从其权威数据中给出不同的应答。
- 你是否将序列号转回到1（又是问题1）？这时主名字服务器的序列号显得比辅名字服务器的序列号要小得多。
- 你是否忘了重载主名字服务器（问题2）？这时（例如，通过*nslookup*或者*dig*）主名字服务器返回的序列号与区数据文件中的序列号不同。
- 辅名字服务器在从主名字服务器上更新数据时有问题吗（问题3）？如果是，它们应在系统日志中记录了相应的错误消息。
- 名字服务器的循环反复（round robin）特性是否转到了你所查找的域名的地址吗？

如果在查询一个远程区中的名字时你得到这些结果，就应该检查那个远程区名字服务器是否已失去了同步。例如，你可以使用*nslookup*或者*dig*这样的工具来确定该远程区的管理员是否忘了增加其序列号。如果该名字服务器从它的权威数据中给出不同的响应，可是却显示相同的序列号，那么序列号可能没有被增加。如果主名字服务器的序列号要远远小于辅名字服务器的序列号，那么主名字服务器的序列号可能被重新复位了。我们通常假定一个区的主名字服务器是运行在这样的主机上，该主机列在SOA记录的MNAME（也就是第一个）字段中。

不过，你或许不能最后确定主名字服务器没有被重载。要防止远程服务器之间的更新问题也是困难的。在这样的案例中，如果你已确定是远程服务器给出的数据不正确，那么就与该区的管理员联系并（客气地）讲明你的发现。这将会帮助管理员跟踪找到在远端的问题。

如果你能确定是一个父服务器（远程区的父服务器、你的区的父服务器或甚至是你的区中的服务器）给出错误的响应，那么就检查是否是由于旧的授权信息引起的。有时这需要与远程区的管理员以及其父区的管理员联系，让他们比较授权信息和当前的、正确的权威服务器列表。

如果你未能促使管理员改正他的数据，或者你未能与那个管理员联系上，你总是可以使用 *bogus* 子语句或 *bogusns* 指令来指示你的名字服务器不去查询那台特定的服务器。

查询耗时很长

名字解析很慢，常常是由如下两个问题之一引起：

- 网络连接丢失（问题 8），对该问题你可以利用名字服务器的调试输出，以及 *ping* 这样的工具来诊断。
- 不正确的授权信息（问题 10），它指向错误的名字服务器或错误的 IP 地址。

通常，检查调试输出和发送几个 *ping* 将指出是哪个问题：要么是你不能到达那个名字服务器，要么是你能抵达那台主机但那个名字服务器不响应。

可是，有时结果不是最后的定论。例如，父名字服务器授权一组名字服务器，这些服务器不响应 *ping* 和查询，但到该远程网络的连接似乎是完全正常的（如，*traceroute* 能让你到达该远程网络的“门坎”——你和那台主机之间的最后一个路由器）。是授权信息太旧而服务器早已改变了地址吗？或者只是主机关机了？或是确实有一个远程网络问题？通常，要找出问题需要给远程区的管理员打个电话或发一条消息。（记住，*whois* 命令会给你电话号码！）

rlogin 和 rsh 到主机访问检查失败

在你刚建好你的名字服务器时，很可能会遇到这个问题。不知道已从主机表改为使用域名服务的用户将不会去更改他们的 *.rhosts* 文件。（我们在第六章中讨论了需要更新什么。）结果，*rlogin* 或 *rsh* 的访问检查将会失败并且拒绝用户的访问。

导致该问题的其他原因是缺少 *in-addr.arpa* 授权信息或 *in-addr.arpa* 授权信息不正

确（问题9和10），并且忘了给那台客户主机添加一条PTR记录（问题4）。如果你是最近才升级到BIND 4.9或更新版本，并且在一个区数据文件中有多于一个*in-addr.arpa*子域的PTR数据，你的名字服务器可能忽略了区以外的数据。任何这些情况将会导致相同的行为：

```
% rlogin wormhole
Password:
```

换句话说，即使通过*.rhosts*或*hosts.equiv*文件设置了访问不用口令，用户还是被提示要求口令。如果你去查看目的主机（在本例中，就是*wormhole.movie.edu*）上的系统日志文件，你可能会看到像这样的消息：

```
May  4 18:06:22 wormhole inetd[22514]: login/tcp: Connection
       from unknown (192.249.249.213)
```

通过使用你最喜欢的查询工具来一步一步地跟踪解析处理过程，你可以分辨是哪个问题。首先查询一个你的*in-addr.arpa*域的父名字服务器，请求你的*in-addr.arpa*区的NS记录。如果这些NS记录是正确的，查询那些被列出的名字服务器，请求相应于*rlogin*或*rsh*客户机的IP地址的PTR记录。确信那些名字服务器都有该PTR记录，而且该记录映射到正确的域名。如果不是所有的名字服务器都有该记录，那么就检查主辅名字服务器之间是否失去了同步（问题1和3）。

访问服务被拒绝

有时，并不是只有*rlogin*和*rsh*服务可用。偶尔你会在你的服务器上安装BIND服务，从而使你的无盘主机不能启动，而且主机也不能从服务器上加载磁盘。

如果发生了这种情况，确信你的名字服务器返回的名字的大小写形式与你以前的名字服务所返回的大小写形式是一致的。例如，如果你以前使用NIS并且你的NIS主机映射只包含小写名字，你应该确信你的名字服务器也返回小写名字。某些程序是区分大小写的，它不能识别在数据文件（如，*etc/bootparams*或*/etc/exports*）中同一个名字的大小写形成。

无法清除旧数据

有时，使一个名字服务器退役或更改服务器的IP地址之后，你会发现其旧地址记录

仍然到处游荡。在几周、甚至几个月之后，旧记录可能还出现在名字服务器的缓存或区数据文件中。很明显，任何缓存中的该记录到这时都应该已超时了。那么为什么它还在那里呢？这有几个原因。我们将首先讨论最简单的情况。

旧的授权信息

如果父区没有跟上它的孩子区的变化，或孩子区没有通知父区关于其区的权威名字服务器的更改，第一种（且是最简单的）情况将会出现。如果 *edu* 的管理员有对 *movie.edu* 的旧的授权信息：

```
$ORIGIN movie.edu.
@     86400     IN     NS     terminator
      86400     IN     NS     wormhole
terminator     86400     IN     A     192.249.249.3
wormhole       86400     IN     A     192.249.249.254 ; wormhole以前的IP地址
```

那么 *edu* 的名字服务器将会给出 *wormhole.movie.edu* 的旧地址。

一旦问题被限制到父区名字服务器上，就容易纠正了：只要与父区的管理员联系，并要求他更新其授权信息即可。如果你的父区是一个 gTLD（通用顶级域），你或许可以通过在你的登记员站点填写表格、修改名字服务器信息的方法来解决问题。如果任何一个子区的服务器已经缓存了坏数据，杀掉它们（清除出它们的缓存），删除任何包含该坏数据的数据文件，然后重启动。

非名字服务器的注册

只有对 gTLD 区，*com*、*net* 和 *org* 才会有这样的问题。有时，你会发现 gTLD 名字服务器给出了你某个区中主机的地址信息，而该主机上根本就没有运行名字服务器！但是为什么 gTLD 名字服务器有你的这个区的主机信息呢？

是这样的：你可以在 gTLD 区中注册根本就不是名字服务器的主机，比如你的 Web 服务器。例如，你可以通过 *com* 登记员来注册 *www.foo.com* 的地址，*com* 名字服务器就会给出这个地址。然而，你不应该这样。因为你将会失去许多有关这个地址的控制。如果你需要改变该地址，可能要花去一天或者更多的时间通过登记员来实现这个改变。如果你运行 *foo.com* 的主名字服务器，你几乎可以马上实现这种改变。

我获得了什么？

怎样确定究竟是哪个问题让你烦恼呢？注意是哪些名字服务器在发布旧数据，以及这些数据与哪些区有关：

- 该名字服务器是一个 gTLD 名字服务器吗？检查过时的、注册过的地址。
- 该名字服务器是你的父名字服务器而又不是gTLD名字服务器吗？那么检查父域看有没有旧的授权信息。

这就是我们所要讨论的全部内容。它的确不够全面，但我们希望它将会帮助你解决在使用DNS时所遇到的更多常见问题，同时给你如何去解决其他问题的思路。如果在我们自己开始学习和使用DNS的时候，有这样一本故障诊断与排除的指导书就好了！

本章内容：

- 用nslookup进行shell脚本编程
- 用解析器库例程进行C编程
- 用Net::DNS进行Perl编程

第十五章 用解析器和名字服务器的库例程编程

“我知道你在想什么，”半斤说道，
“但它不是这样的，绝不是。”“反之，”
八两继续说，“如果它是这样的，那么它就可能是；
如果它本来应该是这样的，那么它就应该是；
但是，如果它不是，那么它就不是。这就是逻辑。”

我敢打赌你会认为解析器编程很困难。恰恰相反！实际上它不是很难。DNS消息格式相当简单——你根本不必像对SNMP那样去处理ASN.1（注1）。而且你有极好的库例程，使解析DNS消息变得容易。在附录一中，我们已收录了RFC 1035的部分内容。然而你可能会发现，在我们讲述本章的过程中，有一份RFC 1035可以随时查阅是很便利的，至少当编写你自己的DNS程序时你需要一份在手边。

用 nslookup 进行 shell 脚本编程

在开始写C程序来做DNS杂事之前，你应该使用*nslookup*或者*dig*来写一个像shell脚本那样的程序。用shell脚本来开始是很有道理的：

注1： ASN.1代表Abstract Syntax Notation。ASN.1是一种对象类型编码语言，被国际标准化组织接受为国际标准。

- 写 shell 脚本要比写 C 程序快得多。
- 如果你不熟悉DNS, 可以用一种快速shell脚本原型得到你的程序逻辑的细节。当最终编写C程序时, 就不必在基本功能上再花时间, 而可以集中精力在C程序提供的额外控件上。
- 你可能会发现shell脚本程序工作得很好, 根本没必要写C程序。不仅编写shell脚本所需的时间短, 而且, 如果你要长期使用它们的话, shell脚本更容易维护。

如果你更喜欢Perl朴实的老式shell编程, 你也可以用 Perl。在本章的最后, 我们将教你如何使用由 Michael Fuhr 写的 Perl Net::DNS 模块。

典型问题

在写程序之前, 你要先找到要解决的问题。让我们来假设你想让你的网络管理系统来监视你的主和辅名字服务器。你想让它通知你几个问题：不在运行的名字服务器（它可能已经死了）、名字服务器本来应该是但却不是一个区的权威（配置文件或区数据文件可能乱了）或者区数据更新滞后的名字服务器（主名字服务器的序列号可能被意外地减小了）。

这些问题中的每一种都很容易被发现。如果一台主机上没有运行名字服务器, 该主机将会发回一个ICMP端口不可达的消息。你可以用查询工具或者解析器例程来发现这一问题。检查一个名字服务器是否是一个区的权威服务器也很容易：向它询问那个区的SOA记录。如果其响应是非权威的, 或该名字服务器没有这个SOA记录, 那么就有问题了。你一定要以非递归方式来查询SOA记录, 以防止名字服务器到另一服务器上去查该 SOA 记录。一旦你有了 SOA 记录, 就可以将序列号抽取出来。

用脚本来解决这个问题

该问题需要一个以区的域名作为参数的程序, 查找出那个区的名字服务器, 然后查询其中每个服务器请求那个区的SOA记录。其响应将会显示那个服务器是否是权威的, 并且显示那个区的序列号。如果没有响应, 该程序需要确定在那台主机上是否有名字服务器在运行。一旦写好该程序, 你要在每个想要监视的区上调用该程序。由于这一程序要查找名字服务器（通过查找那个区的NS记录）, 我们假定在区数据

的 NS 记录中你已经把所有的名字服务器都列出来了。如果不是这样，那么你将不得不修改这一程序使之从命令行上获得名字服务器的列表。

让我们写一个基本的、使用*nslookup*的shell脚本程序。首先，我们必须看看*nslookup*的输出形式，以便能够使用 Unix 的工具来解析它。我们将查找 NS 记录以找出哪些服务器被认为是这个区的权威服务器，包括当该服务器对这些 NS 记录是权威的和非权威的两种情况：

```
% nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2

>  set type=ns
```

当服务器不是这些 NS 记录的权威服务器时的响应情况：

```
>  mit.edu.
Server:  relay.hp.com
Address:  15.255.152.2

Non-authoritative answer:
mit.edu nameserver = STRAWB.MIT.EDU
mit.edu nameserver = W20NS.MIT.EDU
mit.edu nameserver = BITSY.MIT.EDU

Authoritative answers can be found from:
MIT.EDU nameserver = STRAWB.MIT.EDU
MIT.EDU nameserver = W20NS.MIT.EDU
MIT.EDU nameserver = BITSY.MIT.EDU
STRAWB.MIT.EDU  internet address = 18.71.0.151
W20NS.MIT.EDU   internet address = 18.70.0.160
BITSY.MIT.EDU   internet address = 18.72.0.3
```

然后，当服务器是这些 NS 记录的权威服务器时的响应情况为：

```
>  server strawb.mit.edu.
Default Server:  strawb.mit.edu
Address:  18.71.0.151

>  mit.edu.
Server:  strawb.mit.edu
Address:  18.71.0.151

mit.edu nameserver = BITSY.MIT.EDU
mit.edu nameserver = STRAWB.MIT.EDU
mit.edu nameserver = W20NS.MIT.EDU
```

```
BITSY.MIT.EDU   internet address = 18.72.0.3
STRAWB.MIT.EDU  internet address = 18.71.0.151
W20NS.MIT.EDU   internet address = 18.70.0.160
```

从该输出中可以看到，通过查找包含*nameserver*的行并保存行中的最后一个字段的值，我们就可以将名字服务器的域名取出来。当服务器不是这些 NS 记录的权威服务器时，它被打印了两次，因此我们不得不除去重复的行。

接下来，我们查找区的 SOA 记录，包括当服务器是包含该 SOA 记录的区的权威服务器和非权威服务器时的两种情况。我们关掉递归选项以防止名字服务器到别的权威服务器上去查找 SOA 记录：

```
% nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2

> set type=soa
> set norecurse
```

当该名字服务器不是权威并且没有 SOA 记录时，响应如下所示：

```
> mit.edu.
Server:  relay.hp.com
Address:  15.255.152.2

Authoritative answers can be found from:
MIT.EDU nameserver = STRAWB.MIT.EDU
MIT.EDU nameserver = W20NS.MIT.EDU
MIT.EDU nameserver = BITSY.MIT.EDU
STRAWB.MIT.EDU  internet address = 18.71.0.151
W20NS.MIT.EDU   internet address = 18.70.0.160
BITSY.MIT.EDU   internet address = 18.72.0.3
```

当该服务器是区的权威服务器时，响应为：

```
> server strawb.mit.edu.
Default Server:  strawb.mit.edu
Address:  18.71.0.151

> mit.edu.
Server:  strawb.mit.edu
Address:  18.71.0.151

mit.edu
        origin = BITSY.MIT.EDU
        mail addr = NETWORK-REQUEST.BITSY.MIT.EDU
        serial = 1995
```

```
        refresh = 3600 (1H)
        retry   = 900 (15M)
        expire  = 3600000 (5w6d16h)
        minimum ttl = 21600 (6H)
```

当该名字服务器不是这个区的权威服务器时，它返回到其他名字服务器的指针。如果该名字服务器以前查找过该 SOA 记录并将它缓存起来，该名字服务器会返回该 SOA 记录并说明它是非权威的。我们需要检查这两种情况。当名字服务器返回该 SOA 记录并且该记录是权威的，我们可以从含有 *serial* 的行中取出序列号。

现在，我们需要看一看当主机上没有运行名字服务器时 *nslookup* 的返回结果。我们将服务器切换到一台通常不运行名字服务器的主机上，并查找 SOA 记录：

```
% nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2

>  server galt.cs.purdue.edu.
Default Server:  galt.cs.purdue.edu
Address:  128.10.2.39

>  set type=soa
>  mit.edu.
Server:  galt.cs.purdue.edu
Address:  128.10.2.39

*** galt.cs.purdue.edu can't find mit.edu.: No response from server
```

最后，我们要看一看，如果主机没有响应，*nslookup* 会返回什么。我们通过将服务器切换到一个在 LAN 中未使用的 IP 地址来测试这种情况：

```
% nslookup
Default Server:  relay.hp.com
Address:  15.255.152.2

>  server 15.255.152.100
Default Server:  [15.255.152.100]
Address:  15.255.152.100

>  set type=soa
>  mit.edu.
Server:  [15.255.152.100]
Address:  15.255.152.100

*** Request to [15.255.152.100] timed-out
```

在最后两种情况中，错误消息被写到stderr（标准错误输出）（注2）。当写shell脚本时，我们将利用这些东西。现在我们准备好开始写shell脚本了。我们称该程序为*check_soa*：

```
#!/bin/sh
if test "$1" = ""
then
    echo usage: $0 zone
    exit 1
fi
ZONE=$1
#
# 用nslookup查找本区（$1）的名字服务器。
# 用awk从nameserver行获取名字服务器的域名。
#（域名总是位于该行的最后一个字段）用sort -u清除重复的名字；
# 实际上，我们并不关心名字拼写是否正确。
#
SERVERS=`nslookup -type=ns $ZONE |\
                awk '/nameserver/ {print $NF}' | sort -u`
if test "$SERVERS" = ""
then
    #
    # 没有发现任何服务器。仅仅是安静地退出；
    # nslookup将检查到这个错误并打印一条消息。
    # 这已经足够了。
    #
    exit 1
fi
#
# 检查每个名字服务器的SOA序列号。nslookup的输出被保存到两个临时文件：nso.$$
#（标准输出）和nse.$$（标准错误输出）。
# 每次使用nslookup时，这两个文件都将被重写。
# 关掉defname和search选项，因为我们要处理的是全称域名。
#
# 注意：这个循环相当长；不要被它转昏头。
#
for i in $SERVERS
do
  nslookup >/tmp/nso.$$ 2>/tmp/nse.$$ <<-EOF
    server $i
    set nosearch
    set nodefname
    set norecurse
    set q=soa
    $ZONE
EOF
  #
  # 这个响应表示当前的服务器（$i）是权威的吗？
```

注2： 并不是所有版本的*nslookup*都将最后一个错误消息打印成超时。一定要检查到底打印了什么。

```
    # 在两种情况下，该服务器不是权威的：一种是响应说该服务器不是权威的，
    # 另一种是响应告诉你去其他地方查找权威服务器的信息。
    #
    if egrep "Non-authoritative|Authoritative answers can be" \
                                            /tmp/nso.$$ >/dev/null
    then
      echo $i is not authoritative for $ZONE
      continue
    fi
    #
    # 我们知道该服务器是权威的；从中抽取序列号。
    #
    SERIAL=`cat /tmp/nso.$$ | grep serial | sed -e "s/.*= //"`
    if test "$SERIAL" = ""
    then
      #
      # 如果 SERIAL 为空，我们将执行下列指令。在这种情况下，应该有一个出自
      # nslookup 的错误消息；所以 cat 到“标准错误输出”文件。
      #
      cat /tmp/nse.$$
    else
      #
      # 显示服务器的域名和序列号。
      #
      echo $i has serial number $SERIAL
    fi
  done  #"for" 循环结束
  #
  # 删除临时文件。
  #
  rm -f /tmp/nso.$$ /tmp/nse.$$
```

以下是输出结果：

```
%  check_soa mit.edu
BITSY.MIT.EDU has serial number 1995
STRAWB.MIT.EDU has serial number 1995
W20NS.MIT.EDU has serial number 1995
```

如果你的时间紧迫，该简短的工具将会解决你的问题，你就可以去做其他工作了。如果你发现你要检查的区太多，而该工具速度又太慢，就会想将它转换成 C 程序。同样，如果你想要对错误消息进行更多的控制，而不是依赖 *nslookup* 来提供错误消息，那么你就不得不写 C 程序了。接下来我们就谈谈这个问题。

用解析器库例程进行 C 编程

在写代码以前，你需要熟悉 DNS 消息格式和解析器库例程。我们刚才写的 shell 脚

本中，*nslookup* 解析了 DNS 消息。不过 C 程序中，你就不得不自己做解析工作了。让我们从 DNS 的消息格式来开始本节的编程吧。

DNS 消息格式

在第十二章中你已经见过了 DNS 的消息格式，如下所示：

- 首部段
- 问题段
- 回答段
- 权威段
- 附加段

首部段的格式在 RFC 1035 中第 26-28 页有描述，收录在本书附录一中，如下所示：

```
查询标识符（2 字节）
查询响应（1 位）
操作码（4 位）
权威的响应（1 位）
截断（1 位）
期望递归（1 位）
递归可用（1 位）
保留（3 位）
响应码（4 位）
问题数（2 字节）
响应记录数（2 字节）
名字服务器记录数（2 字节）
附加记录数（2 字节）
```

你也可以在*arpa/nameser.h*文件和从消息中抽取信息的例程中找到操作码、响应码、类型和类值的定义。我们将讨论这些例程，简而言之就是名字服务器库。

问题段在 RFC 1035 的第 28-29 页中有描述，如下所示：

```
域名（长度可变）
查询类型（2 字节）
查询类（2 字节）
```

回答段、权威段和附加段在 RFC 1035 第 29-30 页中有描述。这些段由一些如下所示的资源记录组成：

域名（长度可变）
类型（2字节）
类（2字节）
TTL（2字节）
资源数据长度（2字节）
资源数据（长度可变）

首部段含有在每一段中的这些资源记录数的计数。

域名存储

就像你看到的那样，存储在DNS消息中的域名是长度可变的。与C不同，DNS不会将域名存储为以*null*结尾的字符串。域名被存为一系列的长度/值对，并以一个为零的字节结尾。域名中的每一个标号由一个长度字节和标号组成。如，名字*venera.isi.edu*被存为：

```
6 venera  3  isi  3 edu 0
```

你可以想像一个DNS消息需要多少空间来存储域名。DNS的开发者认识到了这一点，并提出了一种简单的方法来压缩域名。

域名压缩

通常，一个完整的域名或者至少域名尾部的标识与一个已经存储在消息中的名字相匹配。域名压缩通过存储一个指向前面出现过的域名的指针而不是再一次插入那个名字来消除域名的重复。这里是它的工作原理。假如一个响应消息已经包含了名字*venera.isi.edu*。如果名字*vaxa.isi.edu*被加到该响应中，标号*vaxa*被存起来，然后加入一个指向前面出现过的*isi.edu*的指针。那么，这些指针是怎样实现的呢？

长度字节中的最初两位表明接下来的是一个长度/标号对还是一个指向长度/标号对的指针。如果最初两位为零，那么跟着的是长度和标号。就像你在第二章读到的那样，一个标号的长度限制是63个字符。那是因为长度字段中只有余下的6位作为标号的长度 —— 足以表示长度0-63。如果长度字节的最初两位为1，那么接下来的就不是长度而是一个指针。指针是长度字节的最后6位和接下来的字节 —— 总共14位。指针是一个从DNS消息开头算起的偏移量。现在，当*vaxa.isi.edu*被压缩到一个只包含*venera.isi.edu*的缓冲区时，其结果是：

```
字节偏移：0 123456 7 890 1 234 5 6 7890 1    2
           -------------+--------------+--------
包内容：   6 venera 3 isi 3 edu 0 4 vaxa 0xC0 7
```

0xC0 是一个高两位为 1 并且其他位为 0 的字节。由于高两位为 1，这是一个指针而不是长度。指针值是 7 —— 第一字节的后 6 位为 0 同时第二字节是 7。在这个缓冲区的偏移量 7 处，你可以找到以 *vaxa* 开始的域名的其余部分：*isi.edu*。

在这个例子中，我们只说明了在缓冲区中压缩两个名字，不是整个 DNS 消息。一个 DNS 消息应该有一个首部同时还有其他字段。本例只是想给出域名压缩的工作原理。现在有一个好消息：只要库例程工作正常，你就不必关心名字是怎么压缩的。你所需要知道的是，如果你错了一个字节的话，解析 DNS 响应消息可能会引起怎样的混乱。例如，尝试从字节 2 而不是字节 1 开始展开名字。你将会发现，“ v ”不是一个好的长度字节或指针。

解析器库例程

解析器的库包含编写应用程序所需的例程。你可以使用这些例程来产生查询。使用接下来我们会讲到的名字服务器库例程来解析响应。

你是不是感到奇怪为什么在我们的编码中没有用 BIND 9 解析器例程吧，好吧，告诉你，这是因为还没写出来呢。BIND 9 包含的库例程能执行许多强大的 DNS 功能，但是据说它们都是面向 BIND 9 名字服务器的需要，并且用起来很复杂。开发者告诉我们，很快就会有更简单的解析器库了，而现在我们只能用 BIND 8 的解析器库。链接到 BIND 8 的库例程在 BIND 9 名字服务器上工作得也很正常。

这里是必须包括的头文件：

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
```

现在让我们来看一看这些解析器的库例程吧。

res_search

```
int res_search(const char *dname,
```

```
                int class,
                int type,
                u_char *answer,
                int anslen)
```

res_search 是“最高级”的解析器例程。它被 *gethostbyname* 调用。*res_search* 将搜索算法应用到传递给它的域名。那就是说，它接受收到的域名（dname），通过加入来自解析器搜索列表中的各种域名来“完善”该名字（如果它不是全称域名的话），再调用 *res_query* 直到收到一个成功的响应，表明已找到了一个合法的全称域名。除了实现了搜索算法，*res_search* 还查看环境变量 HOSTALIASES 所引用的文件。（变量 HOSTALIASES 在第六章中描述过。）所以它也关心你可能拥有的“私有”主机别名。*res_search* 返回响应的大小，如果有错误或响应计数为零时，它将设置 *h_errno* 并返回 −1。（*h_errno* 与 *errno* 相似，只不过 *h_errno* 是用于 DNS 查询的。）

因此，*res_search* 真正感兴趣的参数只有 *dname*；其他参数只是传递给 *res_query* 以及别的解析器例程。其余参数是：

class

你所查数据的类。这几乎总是常量 C_IN，即 Internet 类。这些类常量在 *arpa/nameser.h* 中定义。

type

你所查数据的类型。该常量也在 *arpa/nameser.h* 中定义。其典型的值是 T_NS，即查询名字服务器记录；或 T_MX，即查询 MX 记录。

answer

res_search 用于存放响应消息的缓冲区。它的大小应该至少是 PACKETSZ（在 *arpa/nameser.h* 中定义）字节。

anslen

响应缓冲区的大小（例如，PACKETSZ）。

res_search 将返回响应的大小，或出错时返回 −1。

---

res_query

```
int res_query(const char *dname,
              int class,
              int type,
```

```
                u_char *answer,
                int anslen)
```

res_query 是一个“ 中级的 ”解析器例程。它完成所有查询域名的实际工作，通过调用 *res_mkquery* 来产生查询消息，通过调用 *res_send* 来发送查询，并检查响应以确定问题是否已有答案。许多情况下，*res_query* 被 *res_search* 调用，*res_search* 只把不同的域名传递给它去查找。正如你所预料的，这两个函数有相同的参数。*res_query* 返回响应的大小，如果有错误或响应计数为零时，它将设置 *h_errno* 并返回−1。

---

res_mkquery

```
int res_mkquery(int op,
                const char *dname,
                int class,
                int type,
                const u_char *data,
                int datalen,
                const u_char *newrr,
                u_char *buf,
                int buflen)
```

res_mkquery 函数创建查询消息。它填写所有的首部字段，压缩域名到问题段中，并填写其他问题字段。

参数 *dname*、*class* 和 *type* 与 *res_search* 和 *res_query* 的参数相同。其他参数是：

op

将要执行的“ 操作 ”。通常是 QUERY，也可以是 IQUERY（反向查询）。然而，如我们在前面已解释的那样，很少使用 IQUERY。BIND 4.9.4 版及后续版本，在默认情况下甚至不再支持 IQUERY。

data

包含反向查询数据的缓冲区。当 *op* 为 QUERY 时它是 NULL。

datalen

data 缓冲区的大小。如果 *data* 为 NULL，那么 *datalen* 是零。

newrr

用于动态更新编码的缓冲区（在第十章中讲述过）。除非你正在使用该特性，否则它总是 NULL。

buf

*res_mkquery*生成查询消息的缓冲区。它的大小应该是PACKETSZ或更大，就像在*search*和*res_query*中的响应缓冲区一样。

buflen

*buf*缓冲区的大小（例如，PACKETSZ）。

*res_mkquery*返回查询消息的大小，或在出错误时返回−1。

res_send

```
int res_send(const u_char *msg,
             int msglen,
             u_char *answer,
             int anslen)
```

*res_send*实现了重试算法。它在UDP数据报中发送查询消息*msg*，但它也可以在TCP流上发送。响应消息被存在*answer*中。在所有的解析器例程中，该例程是惟一一个使用巫术（black magic）的例程（除非你精通连接的数据报套接字）。在前面其他解析器例程中，你已见过这些参数：

msg

含有DNS查询消息的缓冲区。

msglen

消息的大小。

answer

存储DNS响应消息的缓冲区。

anslen

响应消息的大小。

res_send返回响应消息的大小，或出错时返回−1。如果该例程返回−1且*errno*为ECONNREFUSED，那么目的名字服务器主机上就没有运行名字服务器。

在调用*res_search*或*res_query*之后，你可以检查*errno*是否是ECONNREFUSED。（*res_search*调用*res_query*，*res_query*调用*res_send*。）如果在调用*res_query*之后

你想检查 *errno*，那么先将 *errno* 清零。那样，你就会知道当前调用的 *res_send* 就是设置 *errno* 的函数。然而，在调用 *res_search* 之前你不必清 *errno*。*res_search* 会在调用 *res_query* 之前自己清 *errno*。

res_init

```
int res_init(void)
```

res_init 从 *resolv.conf* 文件中读入数据并初始化一个称为 *_res* 的数据结构（更多的讨论将在后面进行）。前面所讨论过的所有例程在检测到未曾调用过 *res_init* 时，都要调用 *res_init* 函数。你也可以自己来调用它，如果想在调用第一个解析器库例程之前改变某些默认值，这是很有用的。如果在 *resolv.conf* 文件中有任何 *res_init* 不理解的行时，它将忽略那些行。*res_init* 总是返回 0，即使在手册页中保留返回 −1 的权利。

herror 和 h_errno

```
extern int h_errno;
int herror(const char *s)
```

herror 是一个类似于 *perror* 的例程，但是它是基于外部变量 *h_errno* 的值而不是 *errno* 来打印一个字符串。惟一的参数是：

s 用于标明错误消息的字符串。如果提供了字符串 *s*，则首先打印该字符串，接着是“ :”，然后是基于 *h_errno* 的值的字符串。

下面是 *h_errno* 的可能值：

HOST_NOT_FOUND
域名不存在。名字服务器响应中的返回码是 NXDOMAIN。

TRY_AGAIN
或者名字服务器没在运行，或者是名字服务器返回了 SERVFAIL。

NO_RECOVERY
或是由于一个非法的域名（例如，缺少标号的名字 ——*.movie.edu*）导致域名不能被压缩，或是名字服务器返回了 FORMERR、 NOTIMP 或 REFUSED。

NO_DATA

域名存在，但没有所要求的类型的数据。

NETDB_INTERNAL

有一个与网络或名字服务无关的库错误。查 *errno* 而不是 *h_errno* 以获得该问题的描述。

_res 结构

每一个解析器例程（也就是每一个以 *res_* 开头的例程）使用一个共同的数据结构 *_res*。你可以通过修改 *_res* 来改变解析器例程的行为。如果想修改 *res_send* 的重查次数，你可以修改 *retry* 字段的值。如果想关掉解析器的搜索算法，你可以在 *options* 掩码中关掉 RES_DNSRCH 位。你将在 *resolv.h* 中找到最重要的结构 *_res*：

```
struct _  _res_state {
    int      retrans;    /* 重传时间间隔 */
    int      retry;      /* 重传次数 */
    u_long   options;    /* 选项标志 —— 参看下面 */
    int      nscount;    /* 名字服务器的个数 */
    struct sockaddr_in
             nsaddr_list[MAXNS];     /* 名字服务器的地址 */
#define nsaddr nsaddr_list[0]        /* 为了向后兼容性 */
    u_short id;                      /* 当前包的id号 */
    char     *dnsrch[MAXDNSRCH+1];   /* 被搜索域名的成员 */
    char     defdname[MAXDNAME];     /* 默认域名 */
    u_long   pfcode;                 /* RES_PRF_标志 —— 参看下面 */
    unsigned ndots:4;                /* 初始的查询门限（threshold） */
    unsigned nsort:4;                /* sort_list[]中的元素个数 */
    char     unused[3];
    struct {
           struct in_addr  addr;   /* 要排序的地址 */
           u_int32_t       mask;
    } sort_list[MAXRESOLVSORT];
};
```

options 字段是一个开启选项的简单位掩码。要开启一种特性，在选项字段中打开相应的位即可。每一选项的位掩码在 *resolv.h* 中定义，选项是：

RES_INIT

如果该位是打开的，那么 *res_init* 就已经被调用过了。

RES_DEBUG

如果解析器例程是用DEBUG选项编译的，那么该位将使解析器打印出调试消息。默认情况下该位是关闭的。

RES_AAONLY

要求响应是权威的，而不是从服务器的缓存中来的。这本是一个很有用的特性，可惜没有实现。就BIND解析器的设计而言，这个特性本应在名字服务器中实现，可惜还没有。

RES_PRIMARY

只查询主名字服务器 —— 这又还没有实现。

RES_USEVC

如果你喜欢解析器是通过虚电路（TCP）连接而不是用 UDP 数据报来进行查询，打开该位。正如你可能猜到的那样，建立和拆掉TCP连接会给性能带来损失。默认情况下该位是关闭的。

RES_STAYOPEN

如果你正在一个 TCP 连接上进行查询，打开该位会使该 TCP 连接在查询结束之后仍保持连接状态，所以你可以再使用它查询同一个远程服务器。否则，当查询响应之后，该连接将被拆掉。默认值是关。

RES_IGNTC

如果名字服务器响应中的截断位被置位，那么默认解析器的行为是用TCP重试查询。如果该位被打开，那么在响应消息中的截断位将被忽略，而且不再用 TCP 重试查询。默认值是关。

RES_RECURSE

BIND 解析器的默认行为是发送递归查询。关闭该位即关掉查询消息中的“ 期望递归 ”位。默认值是开。

RES_DEFNAMES

BIND 解析器的默认行为是将本地域名添加到所有没有圆点的域名后。关闭该位即关掉给本地域名添加域名结尾。默认值是开。

RES_DNSRCH

BIND解析器的默认行为是将在搜索列表中的每一项添加到不以圆点结尾的名字中。关闭该位即关闭搜索列表功能。默认值是开。

RES_INSECURE1

4.9.3及后续版本的BIND解析器的默认行为是忽略从未被查询过的服务器来的响应。打开该位即关掉这一安全检查。关（也就是，安全检查是开启状态）是该位的默认值。

RES_INSECURE2

4.9.3及后续BIND版本解析器的默认行为是忽略那些响应中的问题段与原始查询的问题段不匹配的响应。打开该位即关闭这一安全检查。关（也就是，安全检查是开启状态）是该位的默认值。

RES_NOALIASES

BIND解析器的默认行为是使用由用户环境变量HOSTALIASES所指定的文件中定义的别名。对4.9.3和以后版本的BIND解析器，关闭该位即关掉HOSTALIASES特性。以前的解析器不允许禁止该特性。关是默认值。

RES_USE_INET6

告诉解析器对 *gethostbyname* 函数返回IPv6地址（除了IPv4地址以外）。

RES_ROTATE

通常，发送重复查询的解析器总是先查询 *resole.conf* 中的第一个名字服务器。设置了RES_ROTATE, BIND 8.2或者后续版本的解析器会把第一个查询发送给 *resole.conf* 中的第一个名字服务器，第二个查询发送给第二个名字服务器，依次类推。详细说明请参见第六章的 *options rotate* 指令。默认情况是轮转名字服务器。

RES_NOCHECKNAME

自从BIND 4.9.4以后，解析器会检查响应中的域名保证它们和第四章中描述的命名规则一致。BIND 8.2解析器提供是否打开名字检查机制的选项。默认设置是关闭（即，名字检查打开）

RES_KEEPTSIG

该选项使得BIND 8.2或者后续版本的解析器不会从签名的DNS消息中抽出TSIG记录。这样的话，调用解析器的应用可以检查它。

RES_BLAST

同时向所有的递归服务器发送查询。尚未实现。

RES_DEFAULT

这不是一个单独的选项，而是与 RES_RECURSE、RES_DEFNAMES 和 RES_DNSRCH 结合起来使用的，所有这些选项默认都是打开的。通常，你不必特意地设置 RES_DEFAULT；调用 *res_init* 时，它会给你设置好。

名字服务器库例程

名字服务器库包含用于解析响应消息的例程。这里是必须包含的头文件：

```
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/nameser.h>
#include <resolv.h>
```

下面是名字服务器库例程：

---

ns_initparse

```
int ns_initparse(const u_char *msg,
                 int msglen,
                 ns_msg *handle)
```

ns_initparse 是在使用其他名字服务器例程之前必须调用的第一个例程。*ns_init_parse* 填写由 *handle* 所指向的数据结构，该数据结构是传递给其他例程的参数。该例程的参数是：

msg

指向响应消息缓冲区开始位置的指针。

msglen

消息缓冲区的大小。

handle

指向由 *ns_initparse* 填写的数据结构的指针。

ns_initparse 成功时返回 0，解析消息缓冲区失败时返回−1。

ns_msg_base,ns_msg_end 和 ns_msg_size

```
const u_char *ns_msg_base(ns_msg handle)
const u_char *ns_msg_end(ns_msg handle)
int ns_msg_size(ns_msg handle)
```

这些例程返回指向消息开始的指针、指向消息结束的指针和消息的大小。它们返回传入 *ns_initparse* 中的数据。惟一的参数是：

handle

一个由 *ns_initparse* 填写的数据结构。

ns_msg_id

```
u_int16_t ns_msg_id(ns_msg handle)
```

ns_msg_id 从响应消息的首部段（前面讲过）中返回标识。惟一的参数是：

handle

一个由 *ns_initparse* 填写的数据结构。

ns_msg_get_flag

```
u_int16_t ns_msg_get_flag(ns_msg handle, ns_flag flag)
```

ns_msg_get_flag 从响应消息的首部段中返回“ flag ”（标志）字段。其参数是：

handle

一个由 *ns_initparse* 填写的数据结构。

flag

可以具有下列值的枚举型数据：

```
ns_f_qr     /* 查询/响应 */
ns_f_opcode /* 操作码 */
ns_f_aa     /* 权威回答 */
ns_f_tc     /* 发生截断 */
ns_f_rd     /* 期望递归 */
ns_f_ra     /* 递归有效 */
ns_f_z      /* 必须是0 */
ns_f_ad     /* 认证数据（DNSSEC） */
ns_f_cd     /* 取消检查（DNSSEC） */
```

```
ns_f_rcode  /* 响应码 */
ns_f_max
```

ns_msg_count

```
u_int16_t ns_msg_count(ns_msg handle, ns_sect section)
```

ns_msg_count 从响应消息的首部段中返回一个计数器。其参数是：

handle

一个由 *ns_initparse* 填写的数据结构。

section

可以有下列值的枚举类型：

```
ns_s_qd  /* 查询：查询段 */
ns_s_zn  /* 更新：区段 */
ns_s_an  /* 查询：回答段 */
ns_s_pr  /* 更新：先决条件段 */
ns_s_ns  /* 查询：名字服务器段 */
ns_s_ud  /* 更新：更新段 */
ns_s_ar  /* 查询 | 更新：附加记录段 */
```

ns_parserr

```
int ns_parserr(ns_msg *handle,
               ns_sect section,
               int rrnum,
               ns_rr *rr)
```

ns_parserr 提取有关响应记录的信息并将其存储在 *rr* 中。*rr* 是一个传递给其他名字服务器库例程的参数，其参数为：

handle

一个指向由 *ns_init_parse* 填写的数据结构的指针。

section

与在 *ns_msg_count* 中所描述的参数相同。

rrnum

在本段中的资源记录的资源记录号，资源记录从 0 开始编号。*ns_msg_count* 告诉你在这一段中有多少资源记录。

rr

一个指向被初始化的数据结构的指针。

ns_parserr 成功时返回 0，当解析响应缓冲区失败时返回−1。

ns_rr 例程

```
char *ns_rr_name(ns_rr rr)
u_int16_t ns_rr_type(ns_rr rr)
u_int16_t ns_rr_class(ns_rr rr)
u_int32_t ns_rr_ttl(ns_rr rr)
u_int16_t ns_rr_rdlen(ns_rr rr)
const u_char *ns_rr_rdata(ns_rr rr)
```

这些例程从响应记录中返回单个字段的值。它们的惟一参数是：

rr 一个由 *ns_parserr* 填写的数据结构。

ns_name_compress

```
int ns_name_compress(const char *exp_dn,
                     u_char *comp_dn,
                     size_t length,
                     const u_char **dnptrs,
                     const u_char **lastdnptr)
```

ns_name_compress 压缩一个域名。正常情况下你不用调用这个例程 —— 让 *res_mkquery* 来为你调用它。然而，出于某种原因你需要压缩一个域名时，就要调用它。其参数是：

exp_dn

你提供的“扩展的”域名，也就是一个以 null 结尾的包含一个全称域名的字符串。

comp_dn

ns_name_compress 存放被压缩过的域名的地方。

length

comp_dn 缓冲区的大小。

dnptrs

一个指向已压缩的域名的指针数组。*dnptrs[0]*指向消息的开始，该链表以NULL指针结束。在初始化*dnptrs[0]*为指向消息的开头，并初始化*dnptrs[1]*为NULL之后，*dn_comp*会在每次被调用时更新该链表。

lastdnptr

一个指向*dnptrs*数组结尾的指针。*ns_name_compress*需要知道该数组的结尾在哪里，以使它不会越界。

如果你想使用该例程，看一看在BIND源程序*src/lib/resolv/res_mkquery.c*（BIND 8）或者*res/res_mkquery.c*（BIND 4）中它是怎样被使用的。通常从一个例子中去了解如何使用一个例程要比从这些书面的解释中去了解容易。*ns_name_compress*返回压缩后的名字大小，出错时返回—1。

ns_name_uncompress

```
int ns_name_uncompress(const u_char *msg,
                       const u_char *eomorig,
                       const u_char *comp_dn,
                       char *exp_dn,
                       size_t length)
```

*ns_name_uncompress*将展开一个“压缩的”域名。就像在后面的C程序*check_soa*中那样，如果你解析一个名字服务器响应消息，你就会用到该例程。参数是：

msg

一个指向响应消息开头的指针。

eomorig

一个指向该消息之后的第一个字节的指针。它被用来保证*ns_name_uncompress*不会越过该消息的结尾。

comp_dn

一个指向消息内部的已压缩域名的指针。

exp_dn

*ns_name_uncompress*将存放展开后的名字的地方。你应总是为该展开的名字分配一个有MAXDNAME个字符的数组。

length

exp_dn 缓冲区的大小。

ns_name_uncompress 返回压缩名字的大小，或出错时返回－1。你或许想知道为什么 *ns_name_uncompress* 返回压缩名字的大小，而不是展开后的名字大小。这么做的原因是因为当你调用 *ns_name_uncompress* 时，你正在解析 DNS 消息，需要知道压缩名字在消息中所占用的空间大小以便可以跳过它。

ns_name_skip

```
int ns_name_skip(const u_char **ptrptr, const u_char *eom)
```

ns_name_skip 与 *ns_name_uncompress* 相似，不同的是它只是跳过名字而不是将其解压。参数是：

ptrptr

一个指向将要跳过的名字的指针的指针。原来的指针前移越过那个名字。

eom

一个指向该消息之后第一个字节的指针。它被用来保证 *ns_name_skip* 不会越过该消息的结尾。

如果成功 *ns_name_skip* 将返回 0，当解压名字失败时返回－1。

ns_get16 和 ns_put16

```
u_int ns_get16(const u_char *cp)
void  ns_put16(u_int s, u_char *cp)
```

DNS 消息中有一些字段是无符号短整数（如，类型、类和数据长度等）。*ns_get16* 返回一个由 *cp* 指向的 16 位（二进制位）整数。*ns_put16* 将 *s* 的 16 位二进制值赋给由 *cp* 所指向的位置。

ns_get32 和 ns_put32

```
u_long ns_get32(const u_char *cp)
void   ns_put32(u_long l, u_char *cp)
```

除了处理一个32位整数而不是16位整数外，这些例程与对应的16位例程相似。资源记录中的TTL（生存期）是一个32位整数。

解析DNS响应

学习如何解析DNS消息的最简单的方法就是看已有的程序。假如你有BIND的源代码，最好去看*src/lib/resolv/res_debug.c*（BIND 8）或*res/res_debug.c*（BIND 4）。（如果你真的决定使用BIND 9，或许不得不读接近三千行的*lib/dns/message.c*。）*res_debug.c*包括*fp_query*（在BIND 8.2及后续版本中是*res_pquery*），这个函数会在名字服务器的调试输出中打印出DNS消息。我们的范例程序就源于该文件。

你不会总是想手工地解析DNS响应。一种解析响应的“折中”方法是调用*p_query*，*p_query*调用*fp_query*，来打印出DNS消息。然后使用Perl或*awk*来提取你所需要的内容。

范例程序：check_soa

对于那个我们曾经用shell脚本来解决的问题，现在让我们来看看如何用C程序来解决同样的问题。

这里是所需的头文件、外部变量声明和函数声明。注意，我们同时使用了*h_errno*（用于解析器例程）和*errno*。我们将该程序要检查的名字服务器数限制为二十。你很少会看到有多于十个名字服务器的区，所以上限二十应该足够了：

```
/*************************************************************
 * check_soa - 获取指定区的每一个名字服务器的SOA记录,                *
 *        且打印序列号。                                        *
 *                                                           *
 * 用法：check_soa zone                                       *
 *                                                           *
 * 下列错误将被报告:                                          *
 *     o There is no address for a server.                   *
 *     o There is no server running on this host.            *
 *     o There was no response from a server.                *
 *     o The server is not authoritative for the zone.       *
 *     o The response had an error response code.            *
 *     o The response had more than one answer.              *
 *     o The response answer did not contain an SOA record.  *
 *     o The expansion of a compressed domain name failed.   *
*************************************************************/
```

```
/* 头文件 */
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <errno.h>
#include <arpa/nameser.h>
#include <resolv.h>
/* 错误变量 */
extern int h_errno;  /* 用于解析器的错误 */
extern int errno;    /* 通常的系统错误 */

/* 我们自己的例程；本章的稍后部分将介绍 */
void nsError();             /* 报告解析器错误 */
void findNameServers();     /* 查找一个区的名字服务器 */
void addNameServers();      /* 添加名字服务器到我们的列表 */
void queryNameServers();    /* 从名字服务器获取 SOA 记录 */
void returnCodeError();     /* 报告响应消息错误 */

/* 我们将要检查名字服务器个数的最大值 */
#define MAX_NS 20
```

该程序的主体很小。我们用一个字符串指针数组*nsList*来存储该区名字服务器的名字。我们调用解析器函数*res_init*来初始化*_res*结构。该程序不必显式地调用*res_init*函数，因为使用*_res*结构的第一个解析器例程将会调用它。如果想在调用第一个解析器例程之前修改*_res*中的任一字段的值，我们会在刚刚调用*res_init*之后完成这些修改。接着，该程序调用*findNameServers*来找出由在*argv[1]*参数中指明的区的所有名字服务器，把它们存储到*nsList*中。最后，该程序调用*queryNameServers*来查询在*nsList*中的每一个名字服务器以找出该区的SOA记录：

```
main(argc, argv)
int argc;
char *argv[];
{
    char *nsList[MAX_NS];  /* 名字服务器列表 */
    int  nsNum = 0;        /* 列表中的名字服务器个数 */

    /* 安全性检查：一个且仅有一个参数？ */
    if(argc != 2){
        (void) fprintf(stderr, "usage: %s zone\n", argv[0]);
        exit(1);
    }
    (void) res_init();
    /*
     * 查找该区的名字服务器。名字服务器被写到 nsList 中。
     */
    findNameServers(argv[1], nsList, &nsNum);
```

```
    /*
     * 为获得区的 SOA 记录而查询每一个名字服务器。
     * 名字服务器读自 nsList。
     */
    queryNameServers(argv[1], nsList, nsNum);

    exit(0)
}
```

以下为 *findNameServers* 例程。该例程查询本地的名字服务器以找出该区的 NS 记录。然后它调用 *addNameServers* 来解析响应消息，并且保存它所找到的全部名字服务器。在头文件 *arpa/nameser.h* 和 *resolv.h* 中包含我们所使用的函数声明：

```
/************************************************************
 * findNameServers——为指定的区查找所有的名字服务器,                *
 * 且存储它们的名字到 nsList。nNum 是 nsList 数组中的服务器个数。    *
 ************************************************************/
void
findNameServers(domain, nsList, nsNum)
char *domain;
char *nsList[];
int  *nsNum;
{
    union {
        HEADER hdr;              /* 定义在 resolv.h 中 */
        u_char buf[NS_PACKETSZ]; /* 定义在 arpa/nameser.h 中 */
    } response;                  /* 响应缓冲区 */
    int responseLen;             /* 缓冲区的长度 */

    ns_msg handle;  /* 对应响应消息的句柄 */
    /*
     * 查找指定域名的 NS 记录。我们要求域名是全称域名,
     * 所以我们使用 res_query()。如果我们想使用解析器的搜索算法,
     * 则使用 res_search()。
     */
    if((responseLen =
           res_query(domain,      /* 我们所关心的区            */
                     ns_c_in,     /* Internet 类的记录          */
                     ns_t_ns,     /* 查询名字服务器记录          */
                     (u_char *)&response,      /* 响应缓冲区    */
                     sizeof(response)))        /* 缓冲区的大小  */
                                       < 0){  /* 如果是负数    */
        nsError(h_errno, domain); /* 报错                       */
        exit(1);                  /* 并退出                     */
    }
    /*
     * 为该响应初始化一个句柄。这个句柄将被稍后用来从响应中抽取信息
     */
    if (ns_initparse(response.buf, responseLen, &handle) < 0) {
        fprintf(stderr, "ns_initparse: %s\n", strerror(errno));
        return;
```

```
    }

    /*
     * 根据响应生成一个名字服务器列表。依赖DNS的实现不同，NS记录可能位于回答
     * 段，和/或在权威段。名字服务器的地址也许在附加记录段，但是我们将忽略它们，
     * 因为过后调用gethostbyname()比在此解析并存储地址要容易得多。
     */

    /*
     * 从回答段添加名字服务器。
     */
    addNameServers(nsList, nsNum, handle, ns_s_an);

    /*
     * 从权威段添加名字服务器。
     */
    addNameServers(nsList, nsNum, handle, ns_s_ns);
}

/*************************************************************
 * addNameServers - 在一个段中检查资源记录。保存所有名字服务器的名字     *
 *************************************************************/
void
addNameServers(nsList, nsNum, handle, section)
char *nsList[];
int  *nsNum;
ns_msg handle;
ns_sect section;
{
    int rrnum;  /*资源记录号*/
    ns_rr rr;   /*扩展资源记录*/

    int i, dup; /*其他变量*/

    /*
     * 在本段中检查所有的资源记录
     */
    for(rrnum = 0; rrnum < ns_msg_count(handle, section); rrnum++)
    {
        /*
         * 扩展资源记录号rrnum到rr。
         */
        if (ns_parserr(&handle, section, rrnum, &rr)) {
            fprintf(stderr, "ns_parserr: %s\n", strerror(errno));
        }
        /*
         * 如果记录类型是NS，保存名字服务器的名字。
         */
        if (ns_rr_type(rr) == ns_t_ns) {

            /*
             * 分配存储名字字符串的空间。像任何一个优秀的程序员那样，
```

```
                 * 我们测试 malloc 的返回值，且一旦错误就退出。
                 */
                nsList[*nsNum] = (char *) malloc (MAXDNAME);
                if(nsList[*nsNum] == NULL){
                    (void) fprintf(stderr, "malloc failed\n");
                    exit(1);
                }

                /* 扩展名字服务器的域名 */
                if (ns_name_uncompress(
                            ns_msg_base(handle),      /* 消息的开始          */
                            ns_msg_end(handle),       /* 消息的结束          */
                            ns_rr_rdata(rr),          /* 在消息中的位置      */
                            nsList[*nsNum],           /* 结果                */
                            MAXDNAME)                 /*nsList 缓冲区的大小 */
                                    < 0) {            /* 负数：错误          */
                    (void) fprintf(stderr, "ns_name_uncompress failed\n");
                    exit(1);
                }
                /*
                 * 检查我们刚解开的域名，如果不重复就把它添加到名字服务器表中。
                 * 如果是重复的，我们就忽略它。
                 */
                for(i = 0, dup=0; (i < *nsNum) && !dup; i++)
                    dup = !strcasecmp(nsList[i], nsList[*nsNum]);
                if(dup)
                    free(nsList[*nsNum]);
                else
                    (*nsNum)++;
            }
        }
    }
```

注意，我们没有显式地检查零名字服务器记录的情况。我们不需要检查是因为 *res_query* 将标记那种情况为一个错误，它会返回 −1 并设置 *herrno* 为 NO_DATA。如果 *res_query* 返回 −1，我们将调用自己的例程 *nsError*，而不用 *herror*，来打印出 *h_errno* 的错误字符串。*herror* 例程不是很适合我们的程序，因为它的消息假定你正在查询地址数据（也就是，如果 *h_errno* 是 *NO_DATA*，错误消息是“没有与名字关联的地址”）。

下一个例程会查询已找到的每一个名字服务器的 SOA 记录。在这个例程中，我们修改了 *_res* 结构中几个字段的值。通过修改 *nsaddr_list* 字段，我们改变了 *res_send* 要查询的服务器。我们关掉 *options* 字段中的位以关闭搜索列表 —— 本程序处理的名字均为全称域名：

```
/*****************************************************************
*queryNameServers —— 向 nsList 中的每一个名字服务器查询指定区的 SOA 记录。*
```

```
 * 报告碰到的错误（例如，名字服务器没有运行或响应不是一个权威响应），              *
 * 如果没有错误，打印区的序列号。                                              *
 ****************************************************************/
void
queryNameServers(domain, nsList, nsNum)
char *domain;
char *nsList[];
int nsNum;
{
    union {
        HEADER hdr;               /* 定义在 resolv.h 中 */
        u_char buf[NS_PACKETSZ];  /* 定义在 arpa/nameser.h 中 */
    } query, response;            /* 查询和响应缓冲区 */
    int responseLen, queryLen;    /* 缓冲区的长度 */

    u_char     *cp;       /* 解析 DNS 消息的字符指针 */

    struct in_addr saveNsAddr[MAXNS];  /* 根据 _res 保存的地址 */
    int nsCount;          /* 根据 _res 保存的地址数 */
    struct hostent *host; /* 用于查询 ns 地址的结构 */
    int i;                /* 计数器变量 */

    ns_msg handle;  /* 对应响应消息的 handle */
    ns_rr rr;       /*  扩展的资源记录 */

    /*
     * 保存 _res 名字服务器列表，因为我们稍后需要恢复它。
     */
    nsCount = _res.nscount;
    for(i = 0; i < nsCount; i++)
      saveNsAddr[i] = _res.nsaddr_list[i].sin_addr;

    /*
     * 关闭搜索算法，且关闭在我们调用 gethostbyname()前附加本地域名的操作；
     * 名字服务器的域名将是全称域名。
     */
    _res.options &= ~(RES_DNSRCH | RES_DEFNAMES);

    /*
     * 向每个名字服务器查询该区的 SOA 记录
     */
    for(nsNum-- ; nsNum >= 0; nsNum--){

        /*
         * 首先，我们必须获得每个名字服务器的 IP 地址。到目前为止，我们所知道的都
         * 只是域名。我们用 gethostbyname()获得地址信息。但首先，我们必须恢复 _res
         * 中的某些值，因为 _res 将影响 gethostbyname()。（通过循环，我们在前次的
         * 迭代查询中已经改变了 _res 的值）。
         * 我们不能简单地再次调用 res_init()以恢复值，因为 _res 的一些字段在变量
         * 被声明时已经被初始化了，而不是当调用 res_init（）时初始化。
         */
        _res.options |= RES_RECURSE;  /* 递归选项打开(默认的) */
```

```
_res.retry = 4;                 /* 重试四次(默认的)     */
_res.nscount = nsCount;         /* 原来的名字服务器     */
for(i = 0; i < nsCount; i++)
    _res.nsaddr_list[i].sin_addr = saveNsAddr[i];

/* 查找名字服务器的地址 */
host = gethostbyname(nsList[nsNum]);
if (host == NULL) {
    (void) fprintf(stderr,"There is no address for %s\n",
                                        nsList[nsNum]);
    continue; /* nsNum 次 for 循环 */
}
/*
 * 现在准备工作都做好了，下面的工作非常有趣。
 * 主机包含我们正在试验的名字服务器的 IP 地址。
 * 将主机的第一个地址放在 _res 结构中。
 * 然后我们将查找 SOA 记录……
 */
(void) memcpy((void *)&_res.nsaddr_list[0].sin_addr,
   (void *)host->h_addr_list[0], (size_t)host->h_length);
_res.nscount = 1;

/*
 * 关闭递归选项。我们不希望名字服务器向其他名字服务器查询
 * SOA 记录；这个名字服务器就应该是这个数据的权威。
 */
_res.options &= ~RES_RECURSE;

/*
 * 减小重试的次数。我们可能要检查好几个名字服务器，
 * 所以不希望对一个服务器等太长时间。
 * 只查询一个地址，重试两次，我们最多等 15 秒钟。
 */
_res.retry = 2;

/*
 * 我们希望在下一个响应中看到响应码，所以我们自己写查询消息，
 * 并且自己发送，而不用 res_query()来做。
 * 如果 res_query()返回 -1，那么可能没有响应要检查。
 *
 * 没有必要检查 res_mkquery()是否返回 -1。如果压缩失败，
 * 我们早先用这个域名调用 res_query()就会失败。
 */
queryLen = res_mkquery(
             ns_o_query,            /* 常规的查询       */
             domain,                /* 要查找的区       */
             ns_c_in,               /* Internet 类型    */
             ns_t_soa,              /* 查找一个 SOA 记录 */
             (u_char *)NULL,        /* 总是为 NULL      */
             0,                     /* NULL 的长度      */
             (u_char *)NULL,        /* 总是为 NULL      */
             (u_char *)&query,      /* 查询的缓冲区     */
             sizeof(query));        /* 缓冲区的大小     */
```

```
/*
 * 发送查询消息。如果目标主机上没有运行名字服务器，res_send()返回 -1，
 * 并且 errno 为 ECONNREFUSED。首先将 errno 清零。
 */
errno = 0;
if((responseLen = res_send((u_char *)&query,/* 查询   */
                           queryLen,        /* 真正的长度 */
                           (u_char *)&response,/* 缓冲区    */
                           sizeof(response)))  /* 缓冲区大小 */
                                < 0){          /* 错误     */
    if(errno == ECONNREFUSED) { /* 主机上没有服务器          */
        (void) fprintf(stderr,
            "There is no name server running on %s\n",
            nsList[nsNum]);
    } else {                    /* 其他：没有响应 */
        (void) fprintf(stderr,
            "There was no response from %s\n",
            nsList[nsNum]);
    }
    continue; /* nsNum 次 for 循环 */
}

/*
 * 为该响应初始化一个句柄，这个句柄将稍后用来从响应中获取信息。
 */
if (ns_initparse(response.buf, responseLen, &handle) < 0) {
    fprintf(stderr, "ns_initparse: %s\n", strerror(errno));
    return;
}

/*
 * 如果响应报告一个错误，就发出一个消息并继续处理列表中的下一个服务器。
 */
if(ns_msg_getflag(handle, ns_f_rcode) != ns_r_noerror){
    returnCodeError(ns_msg_getflag(handle, ns_f_rcode),
                                          nsList[nsNum]);
    continue; /* nsNum 次 for 循环 */
}

/*
 * 我们收到一个权威响应吗？检查权威回答位。如果名字服务器不是权威的，
 * 报告这个错误并到下一个服务器。
 */
if(!ns_msg_getflag(handle, ns_f_aa)){
    (void) fprintf(stderr,
        "%s is not authoritative for %s\n",
        nsList[nsNum], domain);
    continue; /* nsNum 次 for 循环 */
}

/*
 * 响应应该只包含一个回答；
```

```
             * 如果有多于一个，报告错误并继续处理下一个服务器。
             */
            if(ns_msg_count(handle, ns_s_an) != 1){
                (void) fprintf(stderr,
                    "%s: expected 1 answer, got %d\n",
                    nsList[nsNum], ns_msg_count(handle, ns_s_an));
                continue; /* nsNum 次 for 循环 */
            }

            /*
             * 扩展回答段记录号 0 到 rr。
             */
            if (ns_parserr(&handle, ns_s_an, 0, &rr)) {
                    if (errno != ENODEV){
                            fprintf(stderr, "ns_parserr: %s\n",
                                    strerror(errno));
                    }
            }

            /*
             * 我们询问一个 SOA 记录；如果我们获得其他什么东西，
             * 报告错误并继续处理下一个服务器。
             */
            if (ns_rr_type(rr) != ns_t_soa) {
                (void) fprintf(stderr,
                    "%s: expected answer type %d, got %d\n",
                    nsList[nsNum], ns_t_soa, ns_rr_type(rr));
                continue; /* nsNum 次 for 循环 */
            }

            /*
             * 使 cp 指向 SOA 记录。
             */
            cp = (u_char *)ns_rr_rdata(rr);

            /*
             * 忽略 SOA origin 和邮件地址，这是我们不关心的，
             * 它们都是标准的“压缩名字”
             */
            ns_name_skip(&cp, ns_msg_end(handle));
            ns_name_skip(&cp, ns_msg_end(handle));

            /* cp 现在指向序列号；打印它 */
            (void) printf("%s has serial number %d\n",
                nsList[nsNum], ns_get32(cp));
        } /* nsNum 次 for 循环结束 */
    }
```

注意，当调用 *gethostbyname* 时我们使用的是递归查询，而在查询 SOA 记录时却使用的是非递归查询。*gethostbyname* 可能需要查询别的服务器以查出主机的地址。但

是当查询SOA记录时，我们不希望名字服务器去查询别的服务器——毕竟它应该是该区的权威。允许名字服务器向其他服务器查询SOA记录将会使错误检查失败。

下面的两个例程打印出错误消息：

```
/************************************************************************
 *nsError －根据h_error的值打印查找NS记录失败的错误消息。                    *
 *   Res_query()把DNS消息的返回代码转换成一组较小的错误列表并将错误值          *
 *   放在h_errno中。有一个根据h_errno的值打印字符串的例程，称为herror()，     *
 *   就像perror()函数那样。不幸的是，herror()的消息假设你搜索的是主机的地址记录。*
 *   在本程序中，我们查找的是区的NS记录，所以我们需要自己的错误字符串列表。     *
 ************************************************************************/
void
nsError(error, domain)
int error;
char *domain;
{
    switch(error){
        case HOST_NOT_FOUND:
          (void) fprintf(stderr, "Unknown zone: %s\n", domain);
          break;
        case NO_DATA:
          (void) fprintf(stderr, "No NS records for %s\n", domain);
          break;
        case TRY_AGAIN:
          (void) fprintf(stderr, "No response for NS query\n");
          break;
        default:
          (void) fprintf(stderr, "Unexpected error\n");
          break;
    }
}

/*************************************************************
 * returnCodeError - 根据DNS响应的返回代码打印一条错误消息          *
 *************************************************************/
void
returnCodeError(rcode, nameserver)
ns_rcode rcode;
char *nameserver;
{
    (void) fprintf(stderr, "%s: ", nameserver);
    switch(rcode){
        case ns_r_formerr:
          (void) fprintf(stderr, "FORMERR response\n");
          break;
        case ns_r_servfail:
          (void) fprintf(stderr, "SERVFAIL response\n");
          break;
        case ns_r_nxdomain:
```

```
            (void) fprintf(stderr, "NXDOMAIN response\n");
            break;
          case ns_r_notimpl:
            (void) fprintf(stderr, "NOTIMP response\n");
            break;
          case ns_r_refused:
            (void) fprintf(stderr, "REFUSED response\n");
            break;
          default:
            (void) fprintf(stderr, "unexpected return code\n");
            break;
      }
  }
```

用 *libc* 中的解析器和名字服务器例程来编译本程序：

```
%  cc -o check_soa check_soa.c
```

或者，如果你最近已经按我们在附录三中所描述的那样编译了 BIND 代码，并想使用最新的头文件和解析器库，那么用下面的命令：

```
%   cc -o check_soa -I/usr/local/src/bind/src/include \
check_soa.c /usr/local/src/bind/src/lib/libbind.a
```

以下是输出结果：

```
%  check_soa mit.edu
BITSY.MIT.EDU has serial number 1995
W20NS.MIT.EDU has serial number 1995
STRAWB.MIT.EDU has serial number 1995
```

如果你回过去看那个 shell 脚本的输出，除了 shell 脚本的输出结果是按名字服务器的名字排序的之外，其结果看上去是一样的。你看不出来的是该 C 程序的运行速度要快得多。

用 Net::DNS 进行 Perl 编程

如果用 shell 脚本来解析 *nslookup* 的输出似乎太笨拙，而写 C 程序似乎太复杂，那么考虑使用由 Michael Fuhr 写的 Net::DNS 模块来写 Perl 程序。你可以在 *http://www.perl.com/CPAN-local/modules/by-module/Net/Net-DNS-0.12.tar.gz* 找到该软件包。

Net::DNS 将解析器、DNS 消息、DNS 消息的段和各个资源记录视为对象，并提供设置或查询每个对象属性的方法。我们将首先给出每个对象类型，然后给出一个 Perl 版本的 *check_soa* 程序。

解析器对象

在进行任何查询之前，你必须首先创建一个解析器对象：

```
$res = new Net::DNS::Resolver;
```

解析器对象从*resolv.conf*文件进行初始化，但你可以通过调用该对象的方法来改变默认设置。许多在Net::DNS::Resolver手册页中描述的方法与前面讲的*_res*结构中的字段和选项相对应。例如，如果你想设置解析器在超时之前重试每一查询的次数，可以调用 *$res->retry* 方法：

```
$res->retry(2);
```

要进行一次查询，调用下列方法之一：

```
$res->search
$res->query
$res->send
```

尽管参数要少一些，这些方法的行为与在 C 编程章节中所描述的 *res_search*、*res_query*和*res_send*函数很相似。你必须提供一个名字，可以有选择地提供记录类型和类（默认行为是查询IN类中的A记录）。这些方法返回Net::DNS::Packet对象，待会儿我们会谈到。以下是几个例子：

```
$packet = $res->search("terminator");
$packet = $res->query("movie.edu", "MX");
$packet = $res->send("version.bind", "TXT", "CH");
```

包对象

解析器查询返回Net::DNS::Packet对象，你可以使用这些对象的方法来访问DNS消息的首部、问题、响应、权威和附加段：

```
$header     = $packet->header;
@question   = $packet->question;
```

```
@answer     = $packet->answer;
@authority  = $packet->authority;
@additional = $packet->additional;
```

首部对象

DNS 消息首部作为 Net::DNS::Header 对象被返回。在 Net::DNS::Header 手册页中描述的方法与在RFC 1035中描述的以及C程序中使用的HEADER结构中的首部字段相对应。例如，如果你想找出这是否是一个权威响应，你可以调用 *$header->aa* 方法：

```
if ($header->aa) {
    print "answer is authoritative\n";
} else {
    print "answer is not authoritative\n";
}
```

问题对象

DNS消息的问题段作为Net::DNS::Question对象的一个链表返回。你可以用下列方法来找到一个问题对象的名字、类型和类：

```
$question->qname
$question->qtype
$question->qclass
```

资源记录对象

DNS 消息的响应、权威和附加段作为 Net::DNS::RR 对象的链表返回。你可以用下列方法来找到一个资源记录对象的名字、类型、类和 TTL：

```
$rr->name
$rr->type
$rr->class
$rr->ttl
```

每种记录类型是Net::DNS::RR类的一个子类，并且有它自己特定类型的方法。这里是一个例子，该例显示如何从一个 MX 记录中获得优先级和邮件交换器（mail exchanger）：

```
$preference = $rr->preference;
```

```
$exchanger  = $rr->exchange;
```

Perl 版本的 check_soa

既然我们已经描述了Net::DNS使用的对象, 让我们来看一看在一个完整的程序中如何使用它们。我们用 Perl 重写了 *check_soa*：

```
#!/usr/local/bin/perl -w

use Net::DNS;

#------------------------------------------------------------------------
# 从命令行获取区。
#------------------------------------------------------------------------

die "Usage:  check_soa zone\n" unless @ARGV == 1;
$domain = $ARGV[0];

#------------------------------------------------------------------------
# 查找该区的所有名字服务器
#------------------------------------------------------------------------

$res = new Net::DNS::Resolver;

$res->defnames(0);
$res->retry(2);

$ns_req = $res->query($domain, "NS");
die "No name servers found for $domain: ", $res->errorstring, "\n"
    unless defined($ns_req) and ($ns_req->header->ancount > 0);

@nameservers = grep { $_->type eq "NS" } $ns_req->answer;

#------------------------------------------------------------------------
# 检查每一个名字服务器上的SOA 记录。
#------------------------------------------------------------------------

$| = 1;
$res->recurse(0);
foreach $nsrr (@nameservers) {

  #--------------------------------------------------------------------
  # 将解析器设置为查询该名字服务器。
  #--------------------------------------------------------------------

  $ns = $nsrr->nsdname;
  print "$ns ";

  unless ($res->nameservers($ns)) {
```

```
        warn ": can't find address: ", $res->errorstring, "\n";
        next;
    }

    #------------------------------------------------------------------
    # 获得 SOA 记录。
    #------------------------------------------------------------------

    $soa_req = $res->send($domain, "SOA");
    unless (defined($soa_req)) {
        warn ": ", $res->errorstring, "\n";
        next;
    }

    #------------------------------------------------------------------
    # 该名字服务器是这个区的权威吗？
    #------------------------------------------------------------------

    unless ($soa_req->header->aa) {
        warn "is not authoritative for $domain\n";
        next;
    }

    #------------------------------------------------------------------
    # 我们应该仅获得一个回答。
    #------------------------------------------------------------------

    unless ($soa_req->header->ancount == 1) {
        warn ": expected 1 answer, got ",
              $soa_req->header->ancount, "\n";
        next;
    }

    #------------------------------------------------------------------
    # 我们收到了一个 SOA 记录吗？
    #------------------------------------------------------------------

     unless (($soa_req->answer)[0]->type eq "SOA") {
         warn ": expected SOA, got ",
              ($soa_req->answer)[0]->type, "\n";
         next;
    }

    #------------------------------------------------------------------
    # 打印序列号。
    #------------------------------------------------------------------

    print "has serial number ", ($soa_req->answer)[0]->serial, "\n";
}
```

既然你已经了解了如何使用 shell 脚本、Perl 脚本和 C 代码来编写 DNS 程序，你就应该能使用最适合你的语言来写你自己的程序了。

本章内容：

- 使用 CNAME 记录
- 通配符
- MX 记录的限制
- 拨号连接
- 网络名字和序号
- 其他资源记录
- DNS 和 WINS
- DNS 和 Windows 2000

第十六章

其他问题

海象说：“到了谈论很多事情的时候了：
鞋子、海船和封蜡，卷心菜和国王，
海洋为什么热气腾腾，以及猪是否有双翅。”

该总结一下散乱的头绪了。我们已经讲述了DNS和BIND的主要部分，但是有一些有趣的问题我们还没有探讨。在这些内容中，有一些对你也许是非常有用的，比如，如何在 Windows 2000 上使用 BIND；也许还有其他你感兴趣的东西。出于良好的职业道德，在未完成你的教育之前，我们不能让你去闯荡世界。

使用 CNAME 记录

我们在第四章中讨论了CNAME资源记录。然而，我们没有告诉你关于CNAME记录的全部内容，这些内容我们留在本章讲述。当你建立你的第一台名字服务器时，你并没有注意到神奇的CNAME记录的细微差别。也许你没有认识到还有比我们阐述的更多的内容，也许你并不关心。这些琐碎的内容中，有一些很有趣，有一些很神奇。我们将为你揭开它们神秘的面纱。

附加到内部节点的 CNAME

如果你曾经因为公司的重组而给你的区改名，你也许考虑过生成一个从该区的旧域

名指向新域名的CNAME记录。例如，如果区*fx.movie.edu*更名为*magic.movie.edu*，我们试图生成一个CNAME记录将所有旧域名映射为新域名：

```
fx.movie.edu.  IN  CNAME  magic.movie.edu.
```

通过这个记录，你希望所有对*empire.fx.movie.edu*的查询都转化为对*empire.magic.movie.edu*的查询。不幸的是，这项工作无法完成——如果*fx.movie.edu*是一个拥有其他记录的内部节点，你无法将一个CNAME记录附加给它。请注意，*fx.movie.edu*有一个SOA记录和一个NS记录，因此附加一个CNAME记录给它违反了如下规则：一个域名或者是一个别名或者是一个正规的名字，但不能两者都是。

然而，如果你运行的是BIND 9，你可以使用新的DNAME记录（在第十章中介绍过）创建一个从区的老域名到新域名的别名：

```
fx.movie.edu. IN DNAME magic.movie.edu.
```

DNAME记录可以与*fx.movie.edu*中的其他记录类型共存（就好比SOA记录和NS记录毫无疑问地存在那里一样），但是你不能有以*fx.movie.edu*结尾的任何其他域名。当在*fx.movie.edu*中的名字被查询的时候，它就合成从*fx.movie.edu*中域名到*magic.movie.edu*中域名的CNAME记录。

如果你没有BIND 9，你就不得不用老式的方法为区中的每一个域名创建一个CNAME记录：

```
empire.fx.movie.edu,  IN CNAME empire.magic.movie.edu.
bladerunner.fx.movie.edu. IN CNAME bladerunner.magic.movie.edu.
```

如果子域没有被授权，且相应地没有附加的SOA记录和NS记录，你可以为*fx.movie.edu*生成一个别名，不过它仅仅适用于域名*fx.movie.edu*，而不能适用于*fx.movie.edu*区中的其他域名。

庆幸的是，你用来管理你的区数据文件的工具能够帮助你产生CNAME记录。（第四章为你介绍的*h2n*可以完成这项工作。）

从CNAME指向CNAME

你也许想知道使一个别名（CNAME记录）指向另一个别名是否可能。这在一个别

名从你的区之外的域名指向你的区内的一个域名时是有用的。你也许对你的区之外的别名没有任何控制。如果你想改变它所指向的域名，应该做些什么？你可以简单地添加另一条 CNAME 记录吗？

回答是肯定的：你可以将 CNAME 记录链接在一起。BIND 实现支持该功能，而且 RFC 没有明确地禁止它。不过链接 CNAME 记录不是一个明智之举。RFC 推荐不采用它，因为它可能会产生 CNAME 循环和降低解析速度。紧急关头，你可以这样做，如果有什么地方出现问题，你在网上不会得到太多的同情。而且，如果一台不是基于 BIND 的名字服务器实现出现，所有这一切都将结束（注 1）。

资源记录数据中的 CNAME

除了 CNAME 记录的任何其他记录都必须在资源记录数据中采用正规名字，否则应用程序和名字服务器将不能正常工作。正如我们在第五章中叙述的那样，*sendmail* 只能识别 MX 记录右边的本地主机的正规名。如果 *sendmail* 不能识别本地主机名，当减少 MX 列表时，它将不能准确地分离出 MX 记录，并且可能将邮件发送给自己或者非首选主机，从而引起邮件循环。

当它们在一个记录的右边遇到别名的时候，BIND 8 将会显示如下消息：

```
Sep 27 07:43:48 terminator named[22139]: "digidesign.com IN NS" points to a CNAME
(ns1.digidesign.com)
Sep 27 07:43:49 terminator named[22139]: "moreland.k12.ca.us IN MX" points to a
CNAME (mail.moreland.k12.ca.us)
```

多个 CNAME 记录

一个有缺陷的配置是使多个 CNAME 记录附加到同一域名。某些管理员利用该选项依据循环法在 RRset 之间轮转。例如，下面的记录：

```
fullmonty  IN  CNAME  fullmonty1
fullmonty  IN  CNAME  fullmonty2
fullmonty  IN  CNAME  fullmonty3
```

注 1： 有一种情况就是这样（微软 DNS 服务器，它封装于 Windows NT 和 Windows 2000），不过它允许从 CNAME 指向 CNAME。

可以用来返回名字服务器上所有附加到 *fullmonty1* 的地址，接着返回所有附加到 *fullmonty2* 的地址，然后返回所有附加到 *fullmonty3* 的地址。这种名字服务器不能识别这种错误的配置。（至少，它违反了规则“ CNAME 和其他数据 ”。）

BIND 4 由于识别不了，所以不认为这是一个错误的配置。BIND 8 和 9.1.0 以及后续版本可以识别。如果你需要这种配置，在 BIND 8 中必须使用下列选项：

```
options{
             mutiple-cnames yes;
}
```

在 BIND 9 中没有任何选项允许这种配置。很自然，默认值为不允许。

查询 CNAME

有时候你也许要查询一个 CNAME 记录本身，而不是关于正规名的数据。利用 *nslookup* 或者 *dig* 这很容易做到。你既可以将查询类型设置为 *cname*，也可以将查询类型设置为 *any*，然后查询域名：

```
% nslookup
Default Server:  wormhole
Address:  0.0.0.0

> set query=cname
> bigt
Server:  wormhole
Address:  0.0.0.0

bigt.movie.edu  canonical name = terminator.movie.edu
> set query=any
> bigt
Server:  wormhole
Address:  0.0.0.0

bigt.movie.edu  canonical name = terminator.movie.edu
> exit

% dig bigt.movie.edu cname
; <<>> DiG 8.3 <<>> bigt.movie.edu cname
;; res options: init recurs defnam dnsrch
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 4
;; QUERY SECTION:
```

```
;;      bigt.movie.edu, type = CNAME, class = IN

;; ANSWER SECTION:
bigt.movie.edu.         1D IN CNAME     terminator.movie.edu.
```

找出主机的别名

利用DNS找出主机的别名是件很容易的事。依据主机表可以很容易地找到主机的正规名和任何别名。无论你查找哪一个别名，它都会和正规名一起出现在同一行内：

```
%   grep terminator /etc/hosts
192.249.249.3  terminator.movie.edu terminator bigt
```

然而，如果利用DNS查找正规名，得到的将是正规名。无论是对于应用程序还是名字服务器，都没有好办法得知该正规名是否存在别名：

```
%  nslookup
Default Server:  wormhole
Address:  0.0.0.0

>   terminator
Server:  wormhole
Address:  0.0.0.0

Name:    terminator.movie.edu
Address:  192.249.249.3
```

如果利用 *nslookup* 或者 *dig* 查找别名，你会看到别名和正规名。*nslookup* 和 *dig* 将在消息中报告别名和正规名。但是你看不到指向该正规名的任何其他别名：

```
%  nslookup
Default Server:  wormhole
Address:  0.0.0.0

>  bigt
Server:  wormhole
Address:  0.0.0.0

Name:    terminator.movie.edu
Address:  192.249.249.3
Aliases:  bigt.movie.edu

>  exit
%   dig bigt.movie.edu
; <<>> DiG 8.3 <<>> bigt.movie.edu
;; res options: init recurs defnam dnsrch
```

```
;; got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 4
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL: 4
;; QUERY SECTION:
;;      bigt.movie.edu, type = A, class = IN

;; ANSWER SECTION:
bigt.movie.edu.         1D IN CNAME     terminator.movie.edu.
terminator.movie.edu.   1D IN A         192.249.249.3
```

找出一台主机的全部CNAME的惟一方法就是传送整个区，然后从中挑选出主机是正规名的所有 CNAME 记录：

```
%  nslookup
Default Server:  wormhole
Address:  0.0.0.0

>   ls -t cname movie.edu
 [wormhole.movie.edu]
$ORIGIN movie.edu.
bigt                    1D IN CNAME     terminator
wh                      1D IN CNAME     wormhole
dh                      1D IN CNAME     diehard
>
```

即使这种方法只能显示区内的别名，但还可能有其他区的别名指向该区的正规名。

通配符

还有一项我们没有详细论述的内容就是DNS通配符。有时候，当你想用一个资源记录涵盖任何可能的域名，从而代替无数的除了它们申请的域名以外都相同的资源记录。DNS 保留了一个特殊的字符 —— 星号(*)，作为通配符用在区数据文件中。它可以匹配一个名字中任意数目的标号，只要名字服务器数据库中没有该名字的精确匹配项。

通常，你使用通配符将邮件转发到非 Internet 连通的网络。假设你的站点没有连接到Internet 上，但是你有个主机要在你的网络和Internet 之间转发邮件，这时你可以在*movie.edu* 区中添加一条带有通配符的MX 记录供Internet 使用，从而将你的所有邮件指向转发器，这里举一个例子：

```
*.movie.edu.  IN  MX  10 movie-relay.nea.gov.
```

因为通配符匹配一个或多个标号，所以该资源记录可以作用于诸如 *terminator.movie.edu*、*empire.fx.movie.edu* 或 *casablanca.bogart.classics.movie.edu* 等这类名字。使用通配符的危险性在于与搜索列表的冲突。通配符也匹配 *cujo.movie.edu.movie.edu*，使得通配符在使用内部区数据时变得危险。请注意，在查找MX 记录的时候，有些版本的 *sendmail* 将使用搜索列表：

```
% nslookup
Default Server:  wormhole
Address:  0.0.0.0

> set type=mx                                          —查找 cujo 的 MX 记录
>  cujo.movie.edu
Server:  wormhole
Address:  0.0.0.0
cujo.movie.edu.movie.edu     —这不是一个真实的主机名！
        preference = 10, mail exchanger = movie-relay.nea.gov
```

应用通配符有什么限制呢？通配符不能匹配那些已经有数据的名字。假设你在域的数据中运用通配符，比如在有关 *db.movie* 的部分内容中：

```
*     IN  MX  10 mail-hub.movie.edu.
et    IN  MX  10 et.movie.edu.
jaws  IN  A   192.253.253.113
fx    IN  NS  bladerunner.fx.movie.edu.
fx    IN  NS  outland.fx.movie.edu.
```

发往*terminator.movie.edu*的邮件会被发往*mail-hub.movie.edu*，而发往*et.movie.edu*的邮件会被直接发往*et.movie.edu*。对*jaws.movie.edu*的MX查找会得到“ 没有该名字的MX 数据 ”的结果。通配符不会起作用，因为一个记录已经存在了。通配符也不能应用于 *fx.movie.edu* 域内的域名，因为它们不能透过授权起作用。通配符也不能作用于域名*movie.edu*，因为通配符表示跟随一个圆点的零个或多个标号，或者是跟随 *movie.edu*。

MX 记录的限制

在MX记录这个话题中，让我们讨论一下MX 记录如何导致邮件使用比实际所需更长的路径。当邮件目的地的域名被查找时，MX 记录是返回的数据列表。该列表不是按照离邮件收发器最近的交换器（exchanger）排序的。这里给出有关该问题的一个例子。在你的非Internet连通的网络中有两台可以将Internet上的邮件转发到该网

的主机，其中一台在美国，另一台在法国。你的网络在希腊。你的绝大部分邮件来自美国，因此你安排某个人维护你的域并设置两条通配符 MX 记录 —— 高优先级的指向美国的转发器，低优先级的指向法国的转发器。由于美国的转发器优先级高，所有邮件都通过该转发器（只要它是可到达的）。如果某个在法国的人给你发了一封信，它将穿越大西洋抵达美国，然后再返回，因为在 MX 列表中没有任何信息指明法国的转发器距离发信者比较近。

拨号连接

近来，网络方面的另一个发展对 DNS 提出了挑战，这就是 Internet 拨号连接。在 Internet 早期，DNS 刚出现的时候，没有拨号连接这种服务。随着 Internet 的迅猛发展和流行，以及提供到 Internet 拨号连接的网络服务提供商的大量出现，有关域名服务的一个全新问题被引入了。

设置DNS支持拨号的基本目的是使你网络中的每台主机都能解析需要访问的每台主机的域名。（当然，如果你的 Internet 连接已经断了，那么你的主机根本没必要解析 Internet 域名。）如果你正使用按需拨号，有一个附加的目的 —— 最小化不必要的拨号连接：如果你正查找本地网络上的域名，就不应该请求路由器建立到 Internet 的连接。

我们把拨号连接分为两类：一类是手工拨号，意思是到 Internet 的连接必须由用户手工完成；另一类是按需拨号，这意味着使用一个设备（通常是路由器，但有时就是一台运行 Linux 或其他服务器操作系统的主机）自动连接到 Internet，当主机需要访问 Internet 时。每类拨号方式又分成两种情况：一种是仅有一台需要拨号连接到 Internet 的主机，另一种是包括若干个主机的小网络要拨号连接到 Internet。在讨论这几种情况之前，首先讨论一下引起拨号的原因和如何避免拨号。

拨号的原因

许多ISDN比较流行的地方的用户，尤其在欧洲，通常使用按需拨号连接到Internet。就算不能完全避免，这些用户也希望能将到 Internet 的不必要连接最小化。建立连接通常比多保持几分钟的连接更昂贵，并且总是很浪费时间。

不幸的是，BIND 名字服务器不能很好地适应按需拨号的需求。它们定期地发送查询获取当前根名字服务器的列表，甚至当名字服务器不能解析域名的时候。例如，假设当前你的本地域名是 *tinyoffice.megacorp.com*，并且你有一台本地名字服务器是这个区的权威。在一些解析器上，你的默认搜索列表可能包括：

```
tinyoffice.megacorp.com
megacorp.com
```

现在假设你想 FTP 到本地系统中的 *deadbeef.tinyoffice.megacorp.com*，但是你误拼成 *deadbeer*：

```
% ftp deadbeer
```

由于你的搜索列表、你的解析器将会先查找 *deadbeer.tinyoffice.megacorp.com*。由于是 *tinyoffice.megacorp.com* 区的权威，你的本地名字服务器将告知域名不存在。但这时，你的解析器将附加你的搜索列表中的第二个域名，并查找 *deadbeer.megacorp.com*。为了查找域名是否存在，你的名字服务器将会查询 *megacorp.com* 服务器，而这需要建立拨号连接。

避免拨号

这里有一些技术帮助你最小化不必要的拨号连接。首先，可能也是最简单的，就是运行支持否定缓存的BIND版本（也就是比BIND 4.9.5更新的版本，我们推荐BIND 8 和 9）。这样的话，如果你误将 *deadbeer* 放入配置文件，你的名字服务器将查找 *deadbeer.megacorp.com* 一次，然后缓存该域名不存在这个事实，缓存时间为 *megacorp.com* 否定缓存的 TTL。

另一个技术是使用最小化的搜索列表。如果你的本地域名是 *tinyoffice.megacorp.com*，那么你可以使用只包含 *tinyoffice.megacorp.com* 的搜索列表。这样的话，错误的输入就不会引起拨号。

使用一个现代解析器也是非常重要的。BIND 4.9以后的解析器的默认搜索列表只包括本地域名，在本书中这被认为是最小的搜索列表。并且，即使一个域名没有以点结尾，现代解析器也会试图当它存在一样进行查找。

最后，你可以使用其他的名字服务（比如，*/etc/hosts*）作为本地解析，并且配置你

的解析器使其仅当无法在 */etc/hosts* 中找到一个名字的时候才使用 DNS。只要你的本地主机的名字都在 */etc/hosts* 中，你就不必担心不必要的 Internet 拨号连接。

现在，我们就开始使用这些技术！

单个主机的手工拨号

处理简单拨号情况最容易的方法是配置你的主机解析器使用你的 ISP 的名字服务器。多数ISP为它们的订户（subscriber）运行名字服务器。如果你不能确信你的ISP是否为你的Internet使用提供名字服务器，或者如果你不知道他们的IP地址是什么，请检查他们的 Web 站点，给它们发送 Email，或给他们打电话。

有些操作系统，比如 Windos 95、98 和 NT，允许你为一个特殊的拨号连接定义一组名字服务器。这样，当你拨号到UUNet时可以配置解析器使用名字服务器的一个，当拨号到办公室时使用另一个名字服务器。如果你能拨到多个ISP，这将非常有用。

通常情况下，这些配置对于大多数临时拨号用户已经足够了。如果拨号连接没有建立，域名解析将会失败，但是这也许不是问题，因为没有与 Internet 连通之前，Internet 域名服务是没有用的。

然而，一些人想在拨号连接建立起来的时候运行名字服务器。例如，你可以缓存你经常查找的域名来提高你的机器的性能。对于类似于Unix的操作系统（如，Linux），这是非常容易建立的：通常，你可以使用类似于*ifup*的脚本来建立拨号连接，*ifdown* 来断开连接。这种情况下，*ifup* 和 *ifdown* 在做完大部分工作后也可能分别调用脚本 *ifup-post*和*ifdown-post*。你可以单独启动*named*，或者在*ifup-post*脚本中用*ndc start* 启动它，而且你可以在 *ifdown-post* 脚本中用 *ndc down* 或者 *rndc down* 来关闭它。另外你需要做的惟一一件事情是，在 *resolv.conf* 中设置你的本地域名。查询本地主机上的名字服务器，默认解析器操作都应该成功，不管名字服务器是否在运行。

多个主机的手工拨号

处理多个主机/手工拨号的最简单的方法类似于“只解析器”（resolver-only）配置。你能配置你的解析器使用你的ISP的名字服务器，也可以配置解析器在查询一个名

字服务器前检查 */etc/hosts*（或者 NIS，如果你主张那种情况）。然后确信包含本地网络上所有主机名字的 */etc/hosts* 文件是完好的。

如果你宁愿在本地运行名字服务器，只需稍微改动如下配置：配置你的解析器使用你的本地名字服务器而不是ISP的名字服务器。这样你可以从本地缓存中获益匪浅，但是即使你的Internet连接已经断开，名字解析还是工作的（通过*/etc/hosts*）。如前所述，你也可以从脚本 *ifup-post* 和 *ifdown-post* 中启动和停止本地名字服务器。

如果你真正想使用 DNS 类解析所有的名字，可以放弃 */etc/hosts* 文件，并且在你的名字服务器上为你的主机创建正向地址映射和反向地址映射区。而且你应该最小化你的解析器的搜索列表来避免你的名字服务器查询一些古怪的远程域名。

单个主机的按需拨号

如果你有一个按需拨号到 Internet 的单个主机，最简单的方法是使用只解析器（resolver-only）配置。配置你的解析器使用 ISP 的名字服务器，并且当解析器需要查询一个域名的时候，它会查询其中一个名字服务器并建立连接。如果你的主机查询的一些域名被认为是“家喻户晓”的，如 *localhost* 或者 *1.0.0.127.in-addr.arpa*，那么你可以把它们加入到*/etc/hosts*，并且配置你的解析器在查询名字服务器之前先检查 */etc/hosts*。

如果你宁愿在本地运行名字服务器，一定要保证它能够把 *localhost* 和 *1.0.0127.in-addr.arpa* 分别映射到 127.0.0.1 和 *localhost*，并且要把你的搜索列表修整到最小。

如果你的名字服务器建立了一些你并不希望的连接，你可以打开查询日志（在BIND 4.9 名字服务器上用 *option query-log*，在 BIND 4.9 或者 8 名字服务器上使用 *ndc querylog*，或者在 BIND 9.1.0 名字服务器上使用 *rndc querylog*）并查找建立连接的域名。如果它们大部分是在一个区内的，你或许可以考虑把你的名字服务器配置为那个区的辅名字服务器。至少这样的话，为解析区中的域名你最多在每次刷新间隔建立一次连接。

多个主机的按需拨号

这种情况下最简单的方法完全和我们在“多个主机的手工拨号”一节描述的第一种

办法一样：解析器配置为查询名字服务器之前先检查 */etc/hosts* 的只解析器（resolver-only）配置。对所有的按需拨号配置你都要修整搜索列表。

另外，你可以试着选择下面两种方法中的一个：运行一个本地名字服务器并把它作为 */etc/hosts* 的备份或者在本地名字服务器上创建本地主机的正向和反向映射区。

在按需拨号的情况下运行权威名字服务器

对某些人来讲这或许是个很愚蠢的题目 —— 在按需拨号建立连接的情况下谁会运行权威名字服务器呢？ —— 但是在世界的某些地方，带宽和Internet连接不是很容易得到的情况下，这是很必要的。并且，信不信由你，BIND 提供了一种机制适应这种服务器。

如果你在按需拨号的情况下运行一个权威名字服务器，你要把区维护的活动尽量集中到一个小窗口下。如果你的名字服务器是一百个区的权威，你宁愿没有若干分钟的区刷新时间间隔，因为它会导致 SOA 查询使按需拨号链路一次又一次建立。

在 BIND 8.2 和更新的名字服务器或者 BIND 9.1.0 及其后续版本的名字服务器中，你可以配置 *heartbeat interval*。heartbeat interval 是你设置你的名字服务器建立按需拨号连接的频度，以分钟计算：

```
options {
     heartbeat-interval 180;      // 3小时
};
```

默认值为 60 分钟，你也可以把间隔设为 0 来禁止区维护。

如果你把一个或者多个区作为拨号区，名字服务器将试图把对这些区的维护集中到短的时间间隔内，并试图在 heartbeat interval 内完成维护。对于一个辅区，这意味着抑制正常的刷新间隔（如果刷新间隔比heartbeat interval小，甚至忽略它！），并且只在heartbeat interval 到期后才向主区查询区的SOA 记录。对于主区，这意味着发送 NOTIFY 消息，即会建立按需拨号连接并触发辅名字服务器上的刷新。

你可以使用 *option* 语句中的 *dialup* 子语句来把所有名字服务器的区标记为拨号区：

```
options {
     heartbeat-interval 60;
```

```
        dialup yes;
};
```

如果要标记单个区为拨号区，使用*zone*语句中的*dailup*子语句：

```
zone "movie.edu" {
    type master;
    file "db.movie.edu";
    dialup yes;
};
```

拨号区在另一个地方也是有用的，或许这并不是我们意料之中的：一个名字服务器可以作为成千上万个区的辅名字服务器。一些ISP提供辅名字服务器服务，特点是大规模，但对那些设置他们的区的刷新间隔太小的异己者没什么好处。他们的名字服务器将会在发出这些区的SOA记录查询后被淹没。通过配置所有的区为拨号区并合理地设置heartbeat interval，ISP可以阻止这样的情况发生。

网络名字和序号

最初的DNS规范没有提供根据网络号查找网络名字的能力 —— 最初的*HOSTS.TXT*文件提供的一个特性。从那时起，RFC 1101定义了存储网络名字的系统；该系统也可以用子网或者子网掩码，所以它比*HOSTS.TXT*更有效。而且它根本不需要改变名字服务器软件，它完全基于对PTR和A记录的灵活运用。

记住，为了映射一个IP地址到DNS内的域名，需要反转IP地址，附加*in-addr.arpa*，接着查找PTR记录。相同的技术可以用在将网络号映射到网络名；例如，映射网络15/8到“HP Internet.”。查找网络号，包括网络位和用以组成四个字节的尾部填充的0，并查找PTR数据就像你查找主机的IP地址。例如，查找旧的ARPA网的网络名，网络10/8，查找*0.0.0.10.in-addr.arpa*的PTR数据，你将得到类似*ARPAnet.ARPA*这样的回答。

如果ARPA网划分有子网，你还要在*0.0.0.10.in-addr.arpa*找到一个地址记录。该地址是子网掩码，例如255.255.0.0。如果你对子网名而不是网络名感兴趣，你需要把掩码作用到IP地址上，并查找子网号。

这个技术允许你将网络号映射到网络名。为了提供一个完全的解决方案，必须有一

种方法把网络名映射到网络号，这是通过PTR记录完成的。网络名字有指向网络号的PTR数据（反转网络号并附加 *in-addr.arpa*）。

让我们看一下在HP的区数据文件（HP Internet有网络号15/8）中的数据，并一步一步地将网络号映射成网络名。

db.hp.com 文件的部分内容如下：

```
;
; 将HP的网络名映射到15.0.0.0。
;
hp-net.hp.com.            IN  PTR 0.0.0.15.in-addr.arpa.
```

db.corp.hp.com 文件的部分内容如下：

```
;
; 将corp的子网名映射到15.1.0.0。
;
corp-subnet.corp.hp.com.  IN  PTR 0.0.1.15.in-addr.arpa.
```

db.15 文件的部分内容如下：

```
;
; 将15.0.0.0映射到hp-net.hp.com。
; HP的子网掩码是255.255.248.0。
;
0.0.0.15.in-addr.arpa.    IN  PTR hp-net.hp.com.
                          IN  A   255.255.248.0
```

db.15.1 文件的部分内容如下：

```
;
; 将15.1.0.0映射回它自己的子网名。
;
0.0.1.15.in-addr.arpa.    IN  PTR corp-subnet.corp.hp.com.
```

下面是查找IP地址15.1.0.1对应的子网名的过程：

1. 根据地址类别应用相应的默认网络掩码。15.1.0.1是A类地址，因此掩码是255.0.0.0。对地址15.1.0.1应用该掩码得到网络号为15。
2. 发送一个对 *0.0.0.15.in-addr.arpa* 的查询（*type=A* 或 *type=ANY*）。
3. 查询响应包含地址数据。因为有关于 *0.0.0.15.in-addr.arpa* (子网掩码：255.255.248.0)的地址数据，对该地址应用子网掩码，得到15.1.0.0。

4. 发送一个对 *0.0.1.15.in-addr.arpa* 的查询（*type=A* 或 *type=ANY*）。

5. 查询响应不包含地址数据，因此 15.1.0.0 没有进一步划分子网。

6. 发送一个对 *0.0.1.15.in-addr.arpa* 的 PTR 查询。

7. 查询响应包含对应 15.1.0.1 的网络名：*corp-subnet.corp.hp.com*。

除了在网络名和网络号之间进行映射，你还可以用PTR记录列出关于你的区的所有网络：

```
movie.edu.  IN  PTR  0.249.249.192.in-addr.arpa.
            IN  PTR  0.253.253.192.in-addr.arpa.
```

坏消息：尽管 RFC 1101 包括了建立该映射要了解的全部所需要的内容，可是我们还不知道有哪个软件真正地使用这类网络名字编码，而且很少有管理员因为加入这些信息而惹麻烦。在有软件真正使用DNS编码的网络名字之前，建立这类映射的惟一理由就是炫耀。不过对很多人来说，这个理由已经足够了。

其他资源记录

在本书中，还有几个资源记录我们没有探讨。第一个从一开始就见过面了，就是HINFO，但是没有广泛应用。其他记录在RFC 1183和相继的几个RFC中有定义。大多数是试验性的，但有些已处在标准化过程中，而且正在被广泛使用。我们将讲述它们，从而为它们的使用开个头。

主机信息

HINFO表示主机信息（Host INFOrmation）。记录专用数据是一对识别硬件类型和操作系统的字符串。这些字符串应该来自列在“Assigned Numbers”RFC (通常是RFC 1700)中的MACHINE NAMES 和 OPERATING SYSTEM NAMES，但这个要求不是必须满足的，你可以用自己的缩写。该RFC并不全面，因此你很有可能在列表中找不到你的系统。最初，主机信息记录被设计用来让服务（如，FTP）决定如何与远程系统交互。例如，这使自动协商的数据类型转换成为可能。不幸的是，这没有成功，很少有站点为他们的系统提供精确的HINFO值。一些网络管理员用HINFO记录帮助他们跟踪机器类型，以取代将机器类型记入数据库或笔记本。这里

是两个HINFO记录的例子，注意，如果硬件类型或操作系统中包含空格，它们必须用引号包围起来：

```
;
; 这些机器名和系统名不来自于RFC 1700
;
wormhole  IN  HINFO  ACME-HW  ACME-GW
cujo      IN  HINFO  "Watch Dog Hardware"  "Rabid OS"
```

在把它们加入你的区之前（尤其是那些Internet可见的区），你应该知道HINFO记录可能引起安全隐患。通过提供容易访问的关于系统的信息，给黑客闯入系统提供了方便。

AFSDB

AFSDB的语法类似于MX记录，语义有点类似于NS记录。一个AFSDB记录或者给出一个AFS单元（cell）数据库服务器的位置，或者给出一个DEC单元的认证名字服务器的位置。该记录所指向的服务器类型、运行服务器的主机名都包含在该记录的专用数据部分里。

那么，什么是AFS单元数据库服务器？或者说什么是AFS？AFS最初代表安德鲁文件系统（Andrew File System），在卡内基-梅隆大学（Carnegie-Mellon University）作为安德鲁项目（Andrew Project）的一部分，由一组优秀的人员设计完成。（现在它是IBM的产品。）与NFS一样，AFS是一个网络文件系统，但是AFS在处理广域网延迟方面比NFS做得更好，而且它提供文件的本地缓冲，从而增强性能。一个AFS单元数据库服务器运行一个进程，它负责跟踪一个单元（一组逻辑主机）中的各种AFS文件服务器上的文件集（一组文件）的位置。因此可以看出，AFS单元数据库服务器是找到一个单元内任何文件的关键。

那么，什么是认证名字服务器呢？它拥有一个DCE单元内的各种可利用的服务的位置信息。什么是DCE单元呢？它是一组逻辑上的主机，共享开放组织（Open Group）的分布式计算环境（DCE, Distributed Computing Environment）提供的服务。

现在，让我们言归正传。通过网络访问另一个单元的AFS或DCE服务，你首先要找到被访问单元的单元数据库服务器或认证名字服务器在哪里。因而有了新的记录类型。附加到记录的域名给出了服务器所知的单元名字。单元通常共享DNS域的名字，因此这看起来不会太陌生。

正如我们所说，AFSDB记录的语法和MX记录的语法相似。在记录中的优先值的位置，你可以为AFS单元数据库服务器指定数字1或为DCE认证名字服务器指定数字2。

在记录中邮件交换器主机的位置，你可以指定运行服务器的主机的名字。简单吧！

比如说，一个*fx.movie.edu*的系统管理员建立了一个DCE单元（该单元包括AFS服务），因为她想做实验，用分布式处理加速图形表现。它在*bladerunner.fx.movie.edu*上既运行AFS单元数据库服务器又运行DCE名字服务器，在*empire.fx.movie.edu*上运行另一个单元数据库服务器，在*aliens.fx.movie.edu*上运行另一个DCE名字服务器。她应该像下面这样建立AFSDB记录：

```
; 我们的DCE单元称为fx.movie.edu，与区的域名相同
fx.movie.edu.   IN  AFSDB  1 bladerunner.fx.movie.edu.
                IN  AFSDB  2 bladerunner.fx.movie.edu.
                IN  AFSDB  1 empire.fx.movie.edu.
                IN  AFSDB  2 aliens.fx.movie.edu.
```

X25, ISDN 和 RT

这三类记录是为了支持下一代Internet的研究而专门创建的。其中两类记录：X25和ISDN是特别对应于X.25和ISDN网络的地址记录。这两种记录都有适合网络类型的记录专用数据。X25记录类型使用X.121地址（X.121是ITU-T建议，用来指定使用在X.25网络中的地址格式）。ISDN记录类型使用ISDN地址。

ISDN表示综合业务数字网（Integrated Services Digital Network）。世界范围内的电话公司使用ISDN协议来允许电话网络同时传送话音和数据，形成一个综合网络。尽管ISDN的可用性在整个美国好坏不一，它还是在一些国际市场上被广泛采用。因为ISDN使用电话公司的网络，所以一个ISDN地址就是一个电话号码，实际上它由一个国家号后面紧跟一个地区号或城市号然后再跟一个本地号码构成的。有时，在电话号码的尾部有几位看不见的额外数字，叫子地址。子地址被指定在记录专用数据的一个分离字段中。

几个X25和ISDN记录类型的例子如下所示：

```
relay.pink.com.  IN  X25  31105060845
```

```
delay.hp.com.    IN  ISDN  141555514539488
hep.hp.com.      IN  ISDN  141555514539488 004
```

这些记录是用来与RT（Route Through)记录类型联合使用的。RT在语法和语义上都与MX记录类型相似：它指定一个路由包（而不是邮件）到目的主机的中间主机。因此，现在不仅仅可以路由邮件到不能直接连接到Internet上的主机，你还可以用另一个作为转发器的主机将任何一种IP数据包路由到该主机。这些数据包可以是Telnet或FTP会话的一部分，甚至可以是一个DNS查询。

与MX一样，RT包括一个优先值，它指示到一个特定主机的递送的优先程度。例如，下面的记录：

```
housesitter.movie.edu.  IN  RT  10 relay.pink.com.
                        IN  RT  20 delay.hp.com.
```

指示主机经由*relay.pink.com*（第一选择）或*delay.hp.com*（第二选择）路由到*housesitter.movie.edu*的包。

RT同X25和ISDN（甚至A）记录一起工作的方法如下：

1. Internet主机A要发送一个数据包到没有连接到Internet的主机B。
2. 主机A查找主机B的RT记录。这次查找同时返回每一个中间主机的地址记录（(A、X25和ISDN)。
3. 主机A将中间主机的列表分类，并寻找它自己的域名。如果主机A发现了自己的域名，它就删除该域名和所有高优先值的中间主机。这和*sendmail*对邮件交换器列表的“缩减”操作类似。
4. 主机A检查剩下的优先值最高的中间主机的地址记录。如果主机A所连接的网络对应于地址记录所指示的网络类型，它就使用那个网络发送数据包到这个中间主机。例如，如果主机A试图经由*relay.pink.com*发送数据包，则它需要一个到X.25网络的连接。
5. 如果主机A缺乏适当的连通性，它就尝试下一个由RT记录指定的中间主机，例如，如果主机A缺乏到X.25的连通性，它会退而经由ISDN发送数据包到*delay.hp.com*。

这个过程一直继续，直到数据包被路由到优先值最高的中间主机。然后该优先值最高的中间主机可以直接将数据包递送到目的主机地址（可能是A、X25或者ISDN）。

位置

RFC 1876定义了一个实验型记录类型LOC，它允许域管理员对计算机、子网和网络的位置进行编码。在这种情况下，位置是指纬度、经度和海拔高度。将来的应用程序可以用这些信息产生网络地图、评估路由效率，等等。

在LOC记录的基本形式中，它将纬度、经度和海拔高度（按这种顺序）作为它的记录专用数据。纬度和经度用下面的格式表示：

```
<degrees> [minutes [seconds.<fractional seconds>]] (N|S|E|W)
```

海拔高度用米来表示。

如果你想知道到底如何得到那些数据，请访问*http://www.ckdhr.com/dns-loc/*上的“RFC 1876 Resources”。这个站点是RFC 1876的作者之一Christopher Davis创建的，它搜集了许多不可缺少的信息、有用的链接和人们创建LOC记录的工具。

如果你没有自己的全球定位系统（GPS, Global Positioning System）接收机，有两个站点是很方便的，一个是Etak的Eagle Geocoder，它在*http://www.geocode.com/eagle.html-ssi*上，你可以利用它找到美国多数地址的纬度和经度；另一个是AirNav的Airport Information，它在*http://www.airnav.com/airports/*上，利用它你可以找到距离你最近的机场的海拔高度。如果附近没有较大的机场，请不要担心：数据库中甚至包括我附近医院的直升机停机坪！

这里是我们的一台主机的LOC记录：

```
huskymo.boulder.acmebw.com.  IN  LOC  40 2 0.373 N 105 17 23.528 W 1638m
```

在记录专用数据中的可选字段允许你描述实体的大小，以米为单位（LOC记录可以描述网络，毕竟网络的规模可以很大），就像水平和垂直精度一样。默认值是1米，这对于单个主机是理想的。水平精度的默认值是10,000米，垂直精度是10米。这些默认值表示典型的邮政分区或邮政编码的大小，意思是你可以正确、容易地找到邮政分区号的纬度和经度。

你也可以将LOC记录附加到子网名和网络名。如果你已经花时间以RFC 1101(在本章前面有论述)描述的格式输入你的网络名和地址信息，则你能将LOC记录附加到该网络名字：

```
;
; 将HP的网络名映射到15.0.0.0。
;
hp-net.hp.com.    IN    PTR 0.0.0.15.in-addr.arpa.
                  IN    LOC 37 24 55.393 N 122 8 37 W 26m
```

SRV

如果你事先不知道一个服务或服务器运行在哪台主机上，对它进行定位是很困难的。一些区管理员企图通过使用区内服务专用的别名来解决这个问题。例如，在电影大学，我们生成别名*ftp.movie.edu*，并将它指向运行我们FTP档案库（archive）的主机的域名：

```
ftp.movie.edu.       IN    CNAME           plan9.fx.movie.edu.
```

这使人们很容易地获得域名从而访问我们的FTP档案库，并把人们用来访问档案库的域名与档案库所运行的主机的域名分开。如果移动档案库到不同的主机，我们只要改变CNAME记录就可以了。

在RFC 2052中介绍的实验型SRV记录是一个用于服务定位的通用机制。SRV也提供有力的特性，允许区管理员分散负荷并提供备份服务，这与MX记录相似。

SRV记录的一个独特方面是它被附加的域名的格式。类似服务专用别名，SRV记录所附加的域名给出了待寻找的服务名称，服务所使用的协议，后面是域名。为了与主机域名区分，表示服务名字和协议的标号以下划线开始，例如：

```
_ftp._tcp.movie.edu
```

描绘ftp到*movie.edu*需要获得的SRV记录，以便找到*movie.edu*的FTP服务器，同时：

```
_http._tcp.www.movie.edu
```

描绘访问到URL *http://www.movie.edu/*需要查找的SRV记录，以便找到*www.movie.edu* Web服务器。

服务和协议的名字应该出现在最新 Assigned Number RFC 中（本书写作时的最新 Assigned Number RFC 是 RFC 1700），或仅仅为本地使用而采用惟一名字。不要使用端口或协议号，只使用名字。

SRV 记录有四个资源记录的专用字段：*priority*、*weight*、*port* 和 *target*。*priority*、*weight* 和 *port* 是无符号 16 位整数（0 到 65535）。target 是一个域名。

priority 的作用类似 MX 记录中的优先级：priority 域的数字越小，相关联的目标被使用的可能性越大。当搜索提供给定服务的主机时，客户端应先尝试相同优先级的目标，然后再尝试优先值更高（即优先级更低）的目标。

weight 允许区管理员将负荷分配到多个目标上。客户端将按照权值的比例查询相同优先级的目标。例如，如果一个目标有优先级 0 和权值 1，而另一个目标有优先级 0 和权值 2，则第二个目标接收的负荷（查询、连接，等等）是第一个的两倍。服务的客户分配负载：典型的，这些客户使用系统调用来选择一个随机数。如果这个数字在上三分之一范围内，它们将尝试第一个目标；而如果这个数字在下三分之二范围内，它们将尝试第二个目标。

port 指定被请求的服务运行的端口。这允许区管理员在非标准端口上运行服务器。例如，一个区管理员可以用 SRV 记录指引一个 Web 服务器上的 Web 浏览器在 8000 端口上运行，以取代标准 HTTP 端口（80）。

最后，target 指定运行服务（在 port 字段指定的端口上）的主机的域名。target 必须是主机（不是别名）的规范名字，附加到一个地址记录。

那么，对于 *movie.edu* FTP 服务器，我们添加以下记录到 *db.movie.edu*：

```
_ftp._tcp.movie.edu.  IN  SRV  1  0  21  plan9.fx.movie.edu.
                      IN  SRV  2  0  21  thing.fx.movie.edu.
```

这表示有 SRV 能力的（SRV-capable）FTP 客户端访问 *movie.edu* 的 FTP 服务的时候，首先尝试在 *plan9.fx.movie.edu* 端口 21 上的 FTP 服务器，如果 plan9 的 FTP 服务器不可用，则尝试在 *thing.fx.movie.edu* 端口 21 上的 FTP 服务器。

记录：

```
_http._tcp.www.movie.edu.  IN  SRV  0  2  80   www.movie.edu.
```

```
                                IN  SRV  0  1  80   www2.movie.edu.
                                IN  SRV  1  1  8000 postmanrings2x.movie.edu.
```

指示对 *www.movie.edu* 的 Web 查询先到 *www.movie.edu* 的 80 端口和 *www2.movie.edu* 的 80 端口，并且到 *www.movie.edu* 的查询是到 *www2.movie.edu* 的查询的两倍。如果两个都不可用，则查询到 *postmanrings2x.movie.edu* 的 8000 端口。

为了告知一个特定服务是不可用的，可以在 target 字段使用点“ . ”：

```
_gopher._tcp.movie.edu.  IN  SRV  0  0  0  .
```

不幸的是，在客户中对 SRV 记录的支持是非常少的 —— Windows 2000 是个例外。（本章后面将给出更详细的介绍）。这真是太糟糕了，尽管 SRV 是那么有用。因为 SRV 没有被广泛支持，所以不要用 SRV 记录代替地址记录。至少包括一个针对 SRC 记录所附加的基本域名的地址记录是很明智的，如果你想在不同地址之间分散负荷，还需更多的地址记录。如果在 SRV 记录中列出一个主机仅仅是作为备份，请不要包括它的 IP 地址。同样，如果一个主机在非标准端口上运行服务，请不要包括它的地址记录，因为用 A 记录没有办法使客户端重定向（redirect）到非标准端口。

那么，对于 *www.movie.edu*，需要包括下面所有记录：

```
_http._tcp.www.movie.edu.  IN  SRV  0  2  80   www.movie.edu.
                           IN  SRV  0  1  80   www2.movie.edu.
                           IN  SRV  1  1  8000 postmanrings2x.movie.edu.
www.movie.edu.             IN  A    200.1.4.3 ; www.movie.edu 的地址和
                           IN  A    200.1.4.4 ; www2.movie.edu 的地址
                                              ; 这是给不理解 SRV 的客户端使用的
```

能够处理 SRV 记录（无论它们何时出现）的浏览器发往 *www.movie.edu* 的请求数是发往 *www2.movie.edu* 的两倍，而且当两个主要的 Web 服务器都不可用时，该浏览器将使用 *postmanrings2x.movie.edu*。不能使用 SRV 记录的浏览器将会使请求轮转发往 *www.movie.edu* 和 *www2.movie.edu*。

DNS 和 WINS

在我们的第一版，提到 NetBIOS 名字和 DNS 域名的密切合作，但是要注意，没办法使 DNS 执行 NetBIOS 名字服务器的功能。基本地，要执行 NetBIOS 名字服务器的功能，DNS 名字服务器需要支持动态更新。

当然，BIND 8 和 9 都支持动态更新。不幸的是，Windows NT 4.0 中的 DHCP 服务器不向名字服务器发送动态更新。它仅仅与微软的 WINS 服务器对话。WINS 服务器处理它们自己特别的、专有的动态刷新，不过仅仅针对 NetBIOS 客户端。换句话说，WINS 名字服务器不和 DNS 对话。

不过，微软在 Windows NT 4.0 中提供了名字服务器，它能与 WINS 服务器对话。正如你期望的那样，微软的 DNS 服务器有一个优秀的图形管理工具，并为进入 WINS 提供了方便的接口：你可以配置服务器，当在 DNS 区中没有找到数据时，可以向 WINS 服务器查询该数据。

通过在区中添加一个新的 WINS 记录可以实现上述操作。与 SOA 记录一样，WINS 记录被附加到区的域名。它作为一个标志（flag），告诉微软 DNS 服务器如果没找到要查找的域名地址的话就查询 WINS 服务器。下面这个记录：

```
@       0       IN      WINS            192.249.249.39 192.253.253.39
```

告诉微软 DNS 服务器向运行在 192.249.249.39 和 192.253.253.39（按照这个顺序）上的 WINS 服务器查询名字。TTL 等于 0 是为了防止该记录被查找和存储。

同样，有一个 WINS-R 记录允许微软 DNS 服务器使用 NetBIOS NBSTAT 请求反向映射 IP 地址。如果 *in-addr.arpa* 区包含一个 WINS-R 记录，如下所示：

```
@       0       IN      WINS-R          movie.edu
```

并且在区内没有出现待查找的 IP 地址，名字服务器将尝试向被反向映射的 IP 地址发送一个 NetBIOS NBSTAT 请求。这相当于拨通一个电话号码问接电话的人“你叫什么名字？”结果是一个点和域名附加到记录专用的数据后面，本例中是“.movie.edu”。

这些记录在两个域名空间之间提供了有价值的结合。不幸的是，这样的结合并不完美。麻烦出在细节上。

正如我们所看到的，主要问题是只有微软 DNS 服务器支持 WINS 和 WINS-R（注 2）。

注 2： 少数商业产品（比如，MetaInfo 的 Meta IP/DNS）是另外加上 WINS 功能的一个 BIND 8 端口。无论如何，普通的 BIND 不能与 WINS 服务器对话。

因此，如果你想使 *fx.movie.edu* 区中的查询转发到特技系的 WINS 服务器，则所有 *fx.movie.edu* 名字服务器必须是微软 DNS 服务器。为什么？设想 *fx.movie.edu* 的 DNS 服务器是混合的，一些是微软 DNS 服务器，一些是 BIND。如果一个远程服务器需要查找 *fx.movie.edu* 内的一个 NetBIOS 名字，它将根据往返时间选择使用哪个 *fx.movie.edu* DNS 服务器进行查询。如果碰巧他选择的是微软 DNS 服务器，则它能够将域名解析为动态分配的地址。但是，如果它碰巧选择了 BIND 服务器，它将不能解析该域名。

迄今为止，我们听说过的最好的 DNS-WINS 配置是将所有的 WINS- 映射数据都放置在它自己的区中，比如说 *wins.movie.edu*。所有 *wins.movie.edu* 名字服务器都是微软 DNS 服务器，*wins.movie.edu* 区包含的仅仅只是一个 SOA 记录、NS 记录和一个指向 *wins.movie.edu* 的 WINS 服务器的 WINS 记录。采用这种配置，该区的权威服务器之间不会产生不一致的回答。

当然，反向映射数据并不能很容易地为 BIND 和微软名字服务器划分各自的区来维护。所以，如果你想使传统的基于 PTR 记录的反向映射和增强的 WINS-R 反向映射共存的话，你需要单独在微软 DNS 服务器上运行你的反向映射区。

另一个问题是 WINS 和 WINS-R 是专有的。BIND 名字服务器不理解它们，事实上，从微软主 DNS 服务器传输一个 WINS 记录的 BIND 辅名字服务器将不能加载区，因为 WINS 是未知的类型。（在第十四章中，我们非常详细地讨论过这个问题，以及如何解决这个问题。）

这个问题的答案是在 BIND 8 中引入的 DNS 标准动态更新功能（在第十章描述），且在 Windows 2000 中支持它。动态更新允许授权添加和删除 BIND 名字服务器上的记录，这反过来又为微软的用户提供了将 DNS 用于 NetBIOS 域名服务所需的功能。所以没有进一步的纷乱 ...

DNS 和 Windows 2000

Windows 2000 能使用标准动态更新来注册 DNS 中的主机。对于一个 Windows 2000 客户来讲，注册意味着增加该客户的名字到地址和地址到名字的映射信息，也就是以前注册到 WINS 服务器的信息。对于 Windows 2000 服务器来讲，注册表示添加

记录到一个区以告诉客户他正在运行什么服务以及在哪里（在哪个主机的哪个端口）。例如，Windows 2000 的域名控制器用动态更新增加一个 SRV 记录来告诉 Windows 2000 的客户 Windows 2000 域名的 Kerberos 服务在哪里运行。

Windows 2000 如何使用动态更新

那么，当一个客户注册时，添加了什么内容呢？让我们重新启动特效实验室的 Windows 2000 客户来看一下。

我们的客户叫做 *mummy.fx.movie.edu*。它有固定的 IP 地址 192.253.254.13(而不是从DHCP服务器得到IP)。启动的时候，客户端上的动态更新例程将会按照以下步骤执行：

1. 在本地名字服务器上查找 *mummy.fx.movie.edu* 的 SOA 记录。虽然没有该域名的SOA 记录，但是响应的权威部分含有包含 *mummy.fx.movie.edu* 区的SOA 记录，即响应中包含 *fx.movie.edu* 区的 SOA 记录。
2. 在 SOA 记录的 MNAME 字段查找名字服务器的地址 *bladerunner.fx.movie.edu*。
3. 在下面两个先决条件满足的情况下，向 *bladerunnner.fx.movie.edu* 发送动态更新：第一，*mummy.fx.movie.edu* 不是别名（比如，它不含有 CNAME 记录）；第二，没有指向 192.253.254.13 的地址记录。动态更新包含非更新的部分，它仅仅用来探测那儿有什么。
4. 如果 *mummy.fx.movie.edu* 已经指向它的地址，停止。否则，在 *mummy.fx.movie.edu* 不是别名并且没有地址记录的情况下向 *bladerunner.fx.movie.edu* 发送另一个动态更新。如果条件满足，更新会添加一个从 *mummy.fx.movie.edu* 指向 192.253.264.13 的地址记录。如果 *mummy.fx.movie.edu* 已经有一个地址记录，那么客户端会发送更新去删除该记录并添加它自己的地址记录。
5. 查找 *254.253.192.in-addr.arpa* 的 SOA 记录。
6. 查找在该 SOA 记录的 MNAME 字段中的名字服务器的地址（然而，由于 MNAME 字段包含了我们刚刚查找过的 *bladerunner.fx.movie.edu*，并且 Windows 2000 有一个缓存解析器，所以不需要再次进行查询）。

7. 如果*13.254.253.192.in-addr.arpa*不是别名，就向*bladerunner.fx.movie.edu*发送动态更新。如果先决条件满足，更新将会添加一个从192.253.254.13映射向*mummy.fx.movie.edu*的PTR记录。如果*13.254.253.in-addr.arpa*是别名，停止。

如果我们使用Windows 2000上的微软DHCP服务器，那么默认地，DHCP服务器将会添加一个PTR记录。在DHCP服务器的基于MMC的管理接口上也有一个选项，该选项允许管理员设定DHCP服务器同时添加PTR记录和A记录。然而，如果DHCP服务器已经添加了A记录，它就不会设置先决条件。

服务器，特别是Windows 2000域名控制器，用动态更新在DNS中注册许多信息，不管是第一次启动还是后来定期更新。（例如，*netlogon*服务每小时注册一次它的SRV记录！）这允许客户定位他们正运行在任何主机和端口上的服务。既然我们刚刚设置了一个称为*fx.movie.edu*的Windows 2000域，就看一下我们的域名控制器（*matrix.fx.movie.edu*）所添加的记录：

```
$ORIGIN fx.movie.edu.
@                       600       A       192.253.254.14
_kerberos._tcp.dc._msdcs  600       SRV     0 100 88   matrix.fx.movie.edu.
_ldap._tcp.dc._msdcs      600       SRV     0 100 389  matrix.fx.movie.edu.
_ldap._tcp.e437709a-1862-11d3-8eda-00400536c213.domains._msdcs 600  SRV  0 100 389
matrix.fx.movie.edu.
e4377099-1862-11d3-8eda-00400536c213._msdcs  600  CNAME  matrix.fx.movie.edu.
gc._msdcs                 600       A       192.253.253.14
_ldap._tcp.gc._msdcs      600       SRV     0 100 3268 matrix.fx.movie.edu.
_ldap._tcp.pdc._msdcs     600       SRV     0 100 389  matrix.fx.movie.edu.
_gc._tcp                  600       SRV     0 100 3268 matrix.fx.movie.edu.
_kerberos._tcp            600       SRV     0 100 88   matrix.fx.movie.edu.
_kpasswd._tcp             600       SRV     0 100 464  matrix.fx.movie.edu.
_ldap._tcp                600       SRV     0 100 389  matrix.fx.movie.edu.
_kerberos._udp            600       SRV     0 100 88   matrix.fx.movie.edu.
_kpasswd._udp             600       SRV     0 100 464  matrix.fx.movie.edu.
```

哇！这么多记录啊！

这些记录告诉Windows 2000客户域名控制器提供的服务在哪里运行，包括Kerberos和LDAP（注3）。你可以从SRV记录中看到它们都运行在*matrix.fx.movie.edu*上，我们惟一的域名控制器。如果我们有两个域名控制器，你将会看到几乎两倍多的SRV记录。

注3： 为解释每个记录的功能，请参看微软知识库中的文章Q178169。

所有 SRV 记录的拥有者以 *fx.movie.edu* 结尾，这是 Windows 2000 域的名字。如果我们把我们的 Windows 2000 域称为 *effects.movie.edu*，那么动态更新例程将会更新包含域名 *effects.movie.edu* 和 *movie.edu* 的区。当然，这会使 *movie.edu* 凌乱，因为它有其他运行 Windows 2000 的授权子域。因此，应该确保以我们的区命名我们的 Windows 2000 域名。

Windows 2000 和 BIND 的问题

微软用 DNS 替代 WINS 的决定是伟大的，但同时，微软的实现给那些运行 BIND 名字服务器的人们增加了一些问题。首先 Windows 2000 客户端和 DHCP 服务有一个删除地址记录的坏习惯，这些地址记录属于同一域名的客户或者服务器。例如，如果你让特效实验室的用户自己配置他们的计算机并设置计算机的名字，且碰巧一个用户设置了已经被使用的名字，如被我们的一个 rendering 服务器使用，那么他的计算机可能会试图删除有冲突的地址记录（这些记录属于 rendering 服务器），并增加自己的地址记录。这不太友好。

幸运的是，这个行为可在客户端被纠正。实际上，客户端将检查它正使用的名字是否已有一个地址记录，这个地址记录由上文所述的第四步的先决条件所设置。（如果存在，它将默认删掉它。）但是你可以参照微软知识库中的文章 Q246804，来告诉客户端不要删除这些冲突的记录。代价是什么呢？客户端不能区分相同域名不同主机使用的地址和原来属于它的地址，所以，如果客户端改变了地址，它就不能自动地更新区。

如果你选择你的 DHCP 服务器处理所有的注册，那将没有地址冲突。DHCP 服务器不会使用先决条件来检测冲突，它只是不拘小节地删除冲突地址记录。

既然 DHCP 服务器处理所有注册有缺陷，为什么还要考虑它呢？因为如果你允许所有的客户端注册它们自己，你就只能使用基本的、基于 IP 地址的访问列表来授权动态更新，那么为动态更新区你将允许任何客户端的地址。这些客户端中聪明的用户就可以很容易地制造一些自定义动态更新来改变你的区的 MX 记录或者你的 WEB 服务器的地址。

安全动态更新

确切地说，微软并不仅仅有这些错误，是不是？不是，至少微软DNS服务器就不是。微软 DNS 服务器支持 GSS-TISG，TSIG 的一个变种（参见第十一章）。使用 GSS-TSIG 的客户端从Kerberos名字服务器获取一个TSIG密钥，并用该密钥对动态更新签名。用 GSS（通用安全服务）去获取密钥意味着管理员不需要为每个客户端都硬编码一个密钥。

由于客户端用来对更新签名的 TSIG 密钥名字就是客户的域名，因此名字服务器简单地通过跟踪来增加给定记录的 TSIG 的域名，能确信只有增加地址的客户以后才能删除他所增加的地址。只有具有相同 TSIG 密钥名的更新者被允许删除对应的记录。

如果 Windows 2000 客户端的未签名动态更新被拒绝，他们将尝试使用 GSS-TSIG 签名的动态更新。你也可以根据知识库中的文章Q246804来配置，使它们首先发送带签名的更新。

BIND 和 GSS-TSIG

不幸的是，BIND 名字服务器并不支持 GSS-TSIG，所以你不能在 BIND 上使用 Windows 2000 的安全动态更新。然而，将来版本的 BIND 9 打算支持 GSS-TSIG。一旦 BIND 支持 GSS-TSIG，你就可以使用第十章所描述的“控制哪个密钥更新哪个记录”的所有更新规则了。下面是一个简单的规则集合：

```
zone "fx.movie.edu" {
    type master;
    file "db.fx.movie.edu";
    update-policy {
        grant *.fx.movie.edu. self *.fx.movie.edu. A;
        grant matrix.fx.movie.edu. self matrix.fx.movie.edu. ANY;
        grant matrix.fx.movie.edu. subdomain fx.movie.edu. SRV;
    };
};
```

这或许已经足够使Windows 2000的客户和服务器注册他们需要的存在于你区中的信息。

怎么做呢？

同时，你怎么处理你网络上的Windows 2000的扩散（proliferation）呢？好，微软建议你把所有的名字服务器升级到微软DNS服务器的Windows 2000版本。但是如果你更喜欢BIND（我们当然喜欢），你或许喜欢一些其他的选项。

处理Windows 2000客户

处理你的windows 2000客户的首选（可能是最普遍的）就是为所有客户创建一个授权子域。在我们的例子中称为*win.fx.movie.edu*。在*win.fx.movie.edu*中，任何事情都能发生：客户能够占用其他客户的地址，并且有人可能发送一堆手工生成的动态更新以在区中添加伪记录。创建沙箱（sandbox）的目的（如果你喜欢，可以是监狱）就是防止客户的侵入和破坏。如果你有哄骗的经历，那么你对这个概念会有直观的理解。

默认地，一个Windows 2000客户将试图用相同于Windows 2000域的域名把他自己注册到正向映射区。因此我们必须做一些额外配置，告诉我们的客户在*win.fx.movie.edu*而不在*fx.movie.edu*中注册。特别地，我们必须按顺序*My Computer* → *Properties* → *Network Identification* → *Properties* → *More*打开一个窗口，不选中*Change primary DNS suffix when domain membership changes*，并且在标签为*Primary DNS suffix of this computer*的字段里输入*win.fx.movie.edu*。让我们所有的客户重复上面的操作。

另一个可能性就是将你的客户留在你的主产生区（对我们实验室来说，该区就是*fx.movie.edu*），但是只允许来自DHCP服务器地址的动态更新。而后你配置你的DHCP服务器，设想其负责维护A记录和PTR记录。（你能够为不使用DHCP的主机手工添加A和PTR记录。）

在这个情况下，某些小淘气向你的名字服务器发送他们定制的动态更新就更困难了，这是因为牵涉到DHCP服务器地址的冒用。但还是有可能，某人可以创建一个客户，该客户的域名与区中现存域名冲突。

处理Windows2000服务器

你需要适应的主服务器是DC（Domain Control，域控制器，或控制器，如果你有多

个主服务器的话）。DC打算添加一批我们前面显示的SRV记录。如果DC在启动时不能添加那些记录，DC用主文件格式以超级用户的名义将记录写到文件*System32\Config\netlogon.dns*中。

首先，你需要确定你要更新的区。那就是找到一个包含Windows 2000域名的区。当然，如果你的Windows 2000域与现存区有相同的名字，那么该现存区就是要更新的区。否则，一直删除你的Windows 2000域的开头标号，直到你得到一个区的域名。

一旦得到你需要更新的区，你就要决定如何进行更新。如果你不介意让你的域控制器动态更新你的区，只需要往*zone*语句中添加一条合适的*allow-update*子语句就行了。如果你不允许你的DC完全控制区，你能够取消动态更新，并允许DC创建*netlogon.dns*文件。而后使用一个$INCLUDE控制语句读取文件内容到你的区数据文件中：

```
$INCLUDE netlogon.dns
```

如果你不喜欢这两个选择，这是因为你想要DC能够改变它的SRV记录，但不想要DC破坏你的区，那么你仍旧需要一点小诀窍。你能够利用SRV记录中所有者名字的有趣格式，并创建称为*_udp.fx.movie.edu,_tcp.fx.movie.edu,_site.fx.movie.edu*以及*_msdcd. fx.movie.edu*的授权子域（在我们的例子里）。我们将关闭针对*_msdcd. fx.movie.edu*的名字检查，这是因为除了SRV记录之外，域名控制器打算向区里添加一个地址记录。而后允许DC动态更新这些区，而不包括你的主区：

```
acl dc { 192.253.254.13; };

zone "_udp.fx.movie.edu" {
    type master;
    file "db._udp.fx.movie.edu";
    allow-update { dc; };
};

zone "_tcp.fx.movie.edu" {
    type master;
    file "db._tcp.fx.movie.edu";
    allow-update { dc; };
};

zone "_sites.fx.movie.edu" {
    type master;
    file "db._sites.fx.movie.edu";
```

```
        allow-update { dc; };
    };

    zone "_msdcs.fx.movie.edu" {
        type master;
        file "db._msdcs.fx.movie.edu";
        allow-update { dc; };
        check-names ignore;
    };
```

现在你已经一举两得，既有安全产生区，又可以动态注册服务。

附录一

DNS 消息格式和资源记录

本附录概述了DNS 消息格式，并列举所有资源记录类型。资源记录在区数据文件中是文本格式，而在DNS 消息中是二进制格式。在这里，你会发现有少数资源记录本书没有讲述，因为它们是实验性的或者已经废弃不用了。

这里包括了 Paul Mockapetris 写的 RFC 1035 的一部分，它涉及主文件（本书中称之为区数据文件）的文本格式和DNS消息格式（解析DNS数据包的时候需要它们）。

主文件格式

（摘自 RFC 1035 的 33-35 页）

这些文件的格式是一系列的条目。尽管圆括号可以使条目的列表跨行，文本的文字中也能包含回车换行符（CRLF），但这些条目主要还是面向行的。任何制表符和空格的组合在独立的项之间担当分隔符的角色，这些项构成了条目。主文件中的任一行可以结束于注释。注释以一个分号（“ ;”）开始。

定义了下面这样一些条目：

```
blank[comment]

$ORIGIN domain-name [comment]
```

```
$INCLUDE file-name [domain-name] [comment]

domain-namerr [comment]

blankrr [comment]
```

有或没有注释的空行允许出现在文件的任何地方。

定义了两个控制条目：$ORIGIN 和 $INCLUDE。$ORIGIN 后面跟一个域名，并重新设置相对域名的当前起点为指定的名字。$INCLUDE将指名的文件插入到当前文件中，可选地，还可以指定一个域名用来为所包含的文件设置相对域名起点。$INCLUDE 也可以有注释。注意，$INCLUDE 条目从不改变父文件的相对起点，不管在所包含的文件里是否改变了相对起点。

最后两个条目表示 RR。如果一个 RR 条目以空格开头，则假定该 RR 为最后声明的所有者拥有。如果一个 RR 条目以域名开头，则所有者的名字被重新设置。

RR 的内容是下面表格中的一种：

```
[TTL] [class] type RDATA
[class] [TTL] type RDATA
```

RR 以可选的 TTL 和 class 字段开始，后面跟着是类型和对应于类型和类的 RDATA 字段。类和类型使用标准助记符；TTL 是一个十进制整数。如果省略类和 TTL，那么默认地为最后一次明确声明的值。因为类型和类助记符是不相交的，所以解析是惟一的。

domain-names（域名）使得主文件中的数据得到很大的共享。域名中的标识被表示为以圆点分隔开的字符串。引用惯例允许域名中存储任意字符。以圆点结尾的域名称为绝对域名，被看做是完整的。不是以圆点结尾的域名称为相对域名；实际域名是相对部分和起点的串联，该起点在 $ORIGIN 或 $INCLUDE 中指定，或作为主文件加载例程的参数。若没有起点可用，相对域名就是一个错误。

有两种方法表示 character-string（字符串）：表示为连续的内部没有间隔的一组字符，或者表示为一个以“ 开始和结束的串 ”。一个以“ 括起来的字符串 ”中可以出现任何字符，除了“ 本身之外，如果要写 ”，则必须用反斜杠（\）引用。

因为这些文件是文本文件, 所以为了允许任意数据都能被装入, 需要一些特殊编码。特别是：

. 根。

@ 一个独立的 @ 被用来表示当前起点。

\X

这里 X 是除了阿拉伯数字（0-9）以外的任意字符, 它用来表示该字符本身, 从而使它的特殊含义不起作用。例如, \. 能用来在标识中加入圆点（注 1）。

\DDD

这里每个 D 都是一个阿拉伯数字, 它是一个 8 位字节对应于由 DDD 描述的十进制数。这个 8 位字节被认为是文本, 不会检查它是否有特殊含义（注 2）。

( ) 圆括号用于跨行的数据。实际上是圆括号内的行终止不被承认（注 3）。

; 分号用来开始一个注释, 该行剩余的部分被忽略。

字符的大小写

（摘自 RFC 1035 的第 9 页）

对于 DNS 中所有正式协议的部分, 所有字符串（例如, 标识、域名等）之间的比较都采用不区分大小写的方式。目前, 这一规则适用于整个域系统, 无一例外。不过, 将来在当前应用基础上增加的应用可能要在名字中用完整个二进制8位字节的容量, 因此, 应该避免用 7 位 ASCII 码存储域名, 或用特殊字节结束标识。

类型

这里是资源记录类型的完整列表, 主文件中使用文本表示法。DNS 查询和响应中使用二进制表示法。这些资源记录在 RFC 1035 的 13-21 页中有描述。

注 1： 没有被 BIND 4.8.3 实现。

注 2： 没有被 BIND 4.8.3 实现。

注 3： BIND 4.8.3 只允许在 SOA 和 WKS 资源记录中使用圆括号。

地址

（摘自 RFC 1035 的第 20 页）

文本表示法：

```
owner ttl class A address
```

例子：

```
localhost.movie.edu.   IN A 127.0.0.1
```

二进制表示法：

```
 地址类型编码：1
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ADDRESS                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
其中：
ADDRESS         32 位 Internet 地址。
```

CNAME 规范名

（摘自 RFC 1035 的第 14 页）

文本表示法：

```
owner ttl class CNAME canonical-dname
```

例子：

```
wh.movie.edu.  IN  CNAME  wormhole.movie.edu.
```

二进制表示法：

```
CNAME 类型编码：5
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                     CNAME                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
其中：
CNAME           一个域名，它指定拥有者的规范名或主名（primary name）。拥有者名字是一
                个别名。
```

HINFO 主机信息（HINFO host information）

（摘自 RFC 1035 的第 14 页）

文本表示法：

```
owner ttl class HINFO cpu os
```

例子：

```
grizzly.movie.edu.  IN  HINFO  VAX-11/780 UNIX
```

二进制表示法：

```
HINFO 类型编码：13
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                      CPU                      /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                       OS                      /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
其中：
CPU             一个指定 CPU 类型的字符串。
OS              一个指定操作系统类型的字符串。
```

MB 邮箱域名（实验性的）

（摘自 RFC 1035 的第 14 页）

文本表示法：

```
owner ttl class MB mbox-dname
```

例子：

```
al.movie.edu.  IN  MB  robocop.movie.edu.
```

二进制表示法：

```
MB 类型编码：7
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                   MADNAME                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

其中：
MADNAME　　一个域名，它指定一台拥有指定的邮箱的主机。

MD 邮件目的地（已废弃）

MD 已经被 MX 代替。

MF 邮件转发器（已废弃）

MF 已经被 MX 代替。

MG 邮件组成员（实验性的）

（摘自 RFC 1035 的第 16 页）

文本表示法：

```
owner ttl class MG mgroup-dname
```

例子：

```
admin.movie.edu.  IN  MG  al.movie.edu.
                  IN  MG  ed.movie.edu.
                  IN  MG  jc.movie.edu.
```

二进制表示法：

```
MG 类型编码：8
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    MGMNAME                    /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
其中：
MGMNAME　　一个域名，它指定一个邮箱，该邮箱是域名指定的邮件组的成员。
```

MINFO 邮箱或邮件列表信息（实验性的）

（摘自 RFC 1035 的第 16 页）

文本表示法：

owner ttl class MINFO *resp-mbox error-mbox*

例子：

```
admin.movie.edu.  IN  MINFO  al.movie.edu. al.movie.edu.
```

二进制表示法：

```
MINFO 类型编码：14
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    RMAILBX                    /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    EMAILBX                    /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

其中：

RMAILBX　一个域名，它指定一个对邮件列表或邮箱负责的邮箱。如果这个域名命名了根，则 MINFO RR 的所有者对自己负责。注意，许多现有的邮件列表用邮箱 X-request 表示邮件列表的 RMAILBX 字段，例如，用 Msgroup-request 表示 Msgroup。这个字段提供了更通用的机制。

EMAILBX　一个域名，它指定一个邮箱，该邮箱用来接收与 MINFO RR 的所有者指定的邮件列表或邮箱有关的错误消息（类似于 ERRORS-TO：该字段已经被提议）。如果这个域名命名根，错误要返回给消息的发送者。

MR 邮件重命名（实验性的）

（摘自 RFC 1035 的第 17 页）

文本表示法：

owner ttl class MR *new-mbox*

例子：

```
eddie.movie.edu.  IN  MR  eddie.bornagain.edu.
```

二进制表示法：

```
MR 类型编码：9
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    NEWNAME                    /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

其中：

NEWNAME　一个域名，它指定一个邮箱，该邮箱是给定邮箱的合适的新名字。

MX 邮件交换器

（摘自 RFC 1035 的第 17 页）

文本表示法：

```
owner ttl class MX preference exchange-dname
```

例子：

```
ora.com.  IN  MX  0  ora.ora.com.
          IN  MX  10 ruby.ora.com.
          IN  MX  10 opal.ora.com.
```

二进制表示法：

```
MX 类型编码：15
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                  PREFERENCE                   |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                   EXCHANGE                    /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
其中：
PREFERENCE      一个 16 位整数，它指定这个 RR 在其他属于同一个所有者的 RR 中的优先级。
                值低的优先。
EXCHANGE        一个域名，它指定一个主机作为所有者名字的邮件交换器。
```

NS 名字服务器

（摘自 RFC 1035 的第 18 页）

文本表示法：

```
owner ttl class NS name-server-dname
```

例子：

```
movie.edu.  IN   NS  terminator.movie.edu
```

二进制表示法：

```
NS 类型编码：2
```

```
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    NSDNAME                    /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

其中：

NSDNAME　　一个域名，它指定一个主机，该主机应该是指定类和域的权威。

NULL 空（实验性的）

（摘自 RFC 1035 的第 17 页）

二进制表示法：

```
NULL 类型编码：10
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    anything                   /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

RDATA 字段中可以是任何内容，只要其长度小于等于 65535 字节。

NULL 没有被 BIND 实现。

PTR 指针

（摘自 RFC 1035 的第 18 页）

文本表示法：

```
owner ttl class PTR dname
```

例子：

```
1.249.249.192.in-addr.arpa.  IN PTR wormhole.movie.edu.
```

二进制表示法：

```
PTR 类型编码：12
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    PTRDNAME                   /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

其中：

PTRDNAME　　一个域名，它指向域名空间的某个位置。

SOA 权威的开始

（摘自 RFC 1035 的第 19-20 页）

文本表示法：

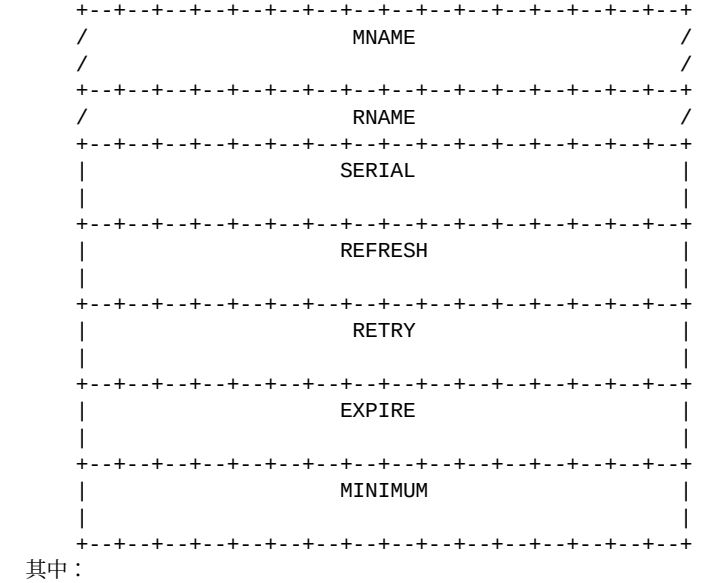
```
owner ttl class SOA source-dname mbox (serial refresh retry expire minimum)
```

例子：

```
movie.edu. IN SOA terminator.movie.edu. al.robocop.movie.edu. (
                      1        ; 序列号
                      10800    ; 3 小时后刷新
                      3600     ; 1 小时后重新
                      604800   ; 1 周后期满
                      86400 )  ; TTL 最小值为 1 天
```

二进制表示法：

SOA 类型编码：6



其中：

MNAME	名字服务器的域名，该服务器是这个区初始的或主数据来源。
RNAME	一个域名，它指定该区负责人的邮箱。
SERIAL	该区初始拷贝的 32 位无符号版本号。区传送保存这个值。这个值在到达最大值后会返回 0，重新开始，要使用序列空间算法进行比较。
REFRESH	一个 32 位区刷新时间间隔。
RETRY	一个 32 位时间间隔，等这个时间间隔过去才可以再次尝试一个失败过的刷新。
EXPIRE	一个 32 位时间值，它指定在该区不再是权威之前可以等待的时间间隔的上限。
MINIMUM	无符号 32 位最小 TTL 字段，它将和来自这个区的任意资源记录一起被输出。

TXT 文本（TXT text）

（摘自 RFC 1035 的第 20 页）

文本表示法：

```
owner ttl class TXT txt-strings
```

例子：

```
cujo.movie.edu.  IN  TXT  "Location: machine room dog house"
```

二进制表示法：

```
TXT 类型编码：16
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    TXT-DATA                   /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
其中：
TXT-DATA        一个或多个字符串。
```

WKS 众所周知的服务

（摘自 RFC 1035 的第 21 页）

文本表示法：

```
owner ttl class WKS address protocol service-list
```

例子：

```
terminator.movie.edu.  IN  WKS 192.249.249.3  TCP ( telnet smtp
                                                ftp shell domain )
```

二进制表示法：

```
WKS 类型编码：11
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |                    ADDRESS                    |
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    |       PROTOCOL        |                       |
    +--+--+--+--+--+--+--+--+                       |
    |                                               |
    /                   BIT MAP                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

其中：

ADDRESS	一个 32 位 Internet 地址
PROTOCOL	一个 8 位 IP 协议号
BIT MAP	一个可变长度的位映射，该位映射的长度必须是 8 的倍数。

摘自 RFC 1183 的新类型

AFSDB 安德鲁文件系统数据库（实验性的）

文本表示法：

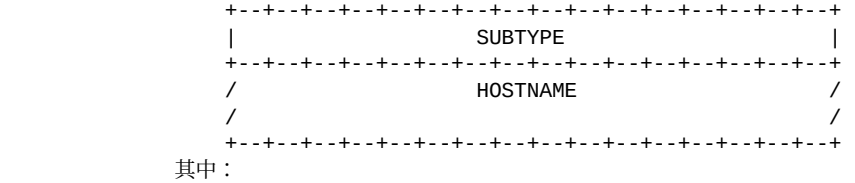
```
owner ttl class AFSDB subtype hostname
```

例子：

```
fx.movie.edu.  IN  AFSDB  1 bladerunner.fx.movie.edu.
               IN  AFSDB  2 bladerunner.fx.movie.edu.
               IN  AFSDB  1 empire.fx.movie.edu.
               IN  AFSDB  2 aliens.fx.movie.edu.
```

二进制表示法：

AFSDB 类型编码：18



其中：

SUBTYPE	SUBTYPE 为 1 表示是一个 AFS 单元数据库服务器。SUBTYPE 为 2 表示是一个 DCE 认证了的名字服务器。
HOSTNAME	一个域名，它指定一台主机，该主机有一个单元服务器，该单元是由 RR 的所有者命名的。

ISDN 综合业务数字网地址（实验性的）

文本表示法：

```
owner ttl class ISDN ISDN-address sa
```

例子：

```
delay.hp.com.    IN  ISDN  141555514539488
```

```
hep.hp.com.      IN  ISDN  141555514539488 004
```

二进制表示法：

```
ISDN 类型编码：20
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                  ISDN ADDRESS                 /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                   SUBADDRESS                  /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
其中：
ISDN ADDRESS     一个字符串，该字符串用来表示所有者的ISDN号和DDI(Direct Dial
                 In)，  如果有DDI的话。
SUBADDRESS       一个可选的字符串，该字符串指定了子地址。
```

RP 负责人（实验性的）

文本表示法：

```
owner ttl class RP mbox-dname txt-dname
```

例子：

```
; 当前起点是fx.movie.edu
@            IN  RP   ajs.fx.movie.edu.   ajs.fx.movie.edu.
bladerunner  IN  RP   root.fx.movie.edu.  hotline.fx.movie.edu.
             IN  RP   richard.fx.movie.edu.  rb.fx.movie.edu.
ajs          IN  TXT  "Arty Segue, (415) 555-3610"
hotline      IN  TXT  "Movie U. Network Hotline, (415) 555-4111"
rb           IN  TXT  "Richard Boisclair, (415) 555-9612"
```

二进制表示法：

```
RP 类型编码：17
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    MAILBOX                    /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
    /                    TXTDNAME                   /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
其中：
MAILBOX          一个域名，它指定负责人的邮箱。
TXTDNAME         一个TXT RR所在域的域名。接下来执行的查询可以获得txt-dname中相
                 关的TXT资源记录。
```

RT 路由通过（实验性的）

文本表示法：

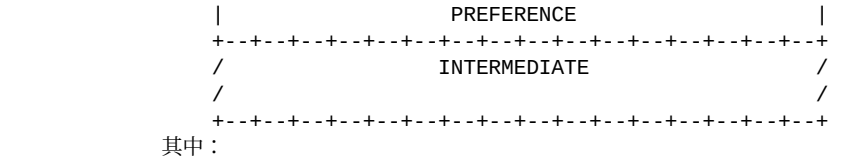
```
owner ttl class RT preference intermediate-host
```

例子：

```
sh.prime.com.  IN  RT  2   Relay.Prime.COM.
               IN  RT  10  NET.Prime.COM.
```

二进制表示法：

RT 类型编码：21



其中：

PREFERENCE　一个 16 位整数，它指定这个 RR 在其他属于同一个所有者的 RR 中的优先级。值低的优先。

EXCHANGE　一个域名，它指定一个主机，该主机作为到达所有者指定的主机的媒介。

X25 X.25 地址（实验性的）

文本表示法：

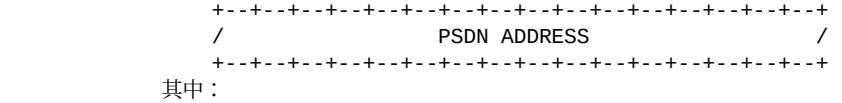
```
owner ttl class X25 PSDN-address
```

例子：

```
relay.pink.com.  IN  X25   31105060845
```

二进制表示法：

X25 类型编码：19



其中：

PSDN ADDRESS　一个字符串，它标识与所有者相关联的 X.121 编号计划中的 PSDN(Public Switched Data Network，公共交换数字网)地址。

摘自 RFC 1664 的新类型

PX 指向 X.400/RFC 822 映射信息的指针

文本表示法：

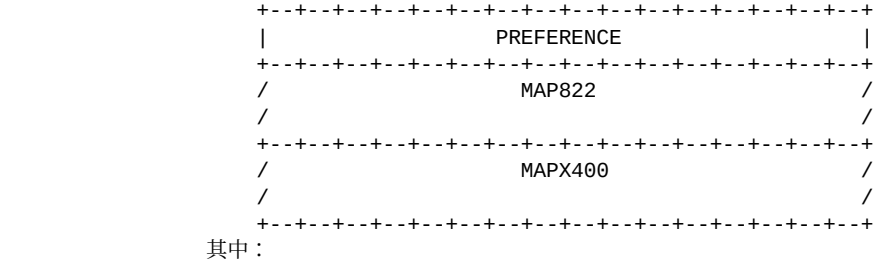
```
owner ttl class PX preference RFC822 address X.400 address
```

例子：

```
ab.net2.it.  IN  PX  10   ab.net2.it.  O-ab.PRMD-net2.ADMDb.C-it.
```

二进制表示法：

PX 类型编码：26



其中：

PREFERENCE　一个 16 位整数，它指定这个 RR 在其他属于同一个所有者的 RR 中的优先级。值低的优先。

MAP822　一个包含 rfc822 域的域名单元，rfc822 域是 RFC 1327 映射信息的 RFC 822 部分。

MAPX400　一个包含以域语法表示的 x400 值的域名单元，以域语法表示的 x400 值源于 RFC 1327 映射信息的 X.400 部分。

类

（摘自 RFC 1035 的第 13 页）

CLASS 字段出现在资源记录中。定义有下面一些 CLASS 助记符和值：

IN

1：Internet

CS

2：CSNET 类（已被废弃 —— 仅用于一些已被废弃的 RFC 的示例中）

CH

3：CHAOS 类

HS

4：Hesiod 类

DNS 消息

为了编写解析 DNS 消息的程序，你需要了解 DNS 的消息格式。DNS 查询和响应常常包含于 UDP 数据报中。每一条消息都完全包含于一个 UDP 数据报中。如果查询和响应是通过TCP发送的，则要在它们前面加一个双字节值指示该查询或响应的长度（不包括双字节值的长度）。DNS 消息的格式和内容如下所示。

消息格式

（摘自 RFC 1035 的第 25 页）

所有域协议内的通信都可以通过一种单一的格式传送，这种格式叫做消息（message）。消息的顶级格式分为五段（某些情况下，一些段是空的），如下所示：

```
+---------------------+
|        Header       |
+---------------------+
|       Question      | 查询名字服务器的问题
+---------------------+
|        Answer       | 回答问题的资源记录
+---------------------+
|      Authority      | 指向权威的资源记录
+---------------------+
|      Additional     | 包括附加信息的资源记录
+---------------------+
```

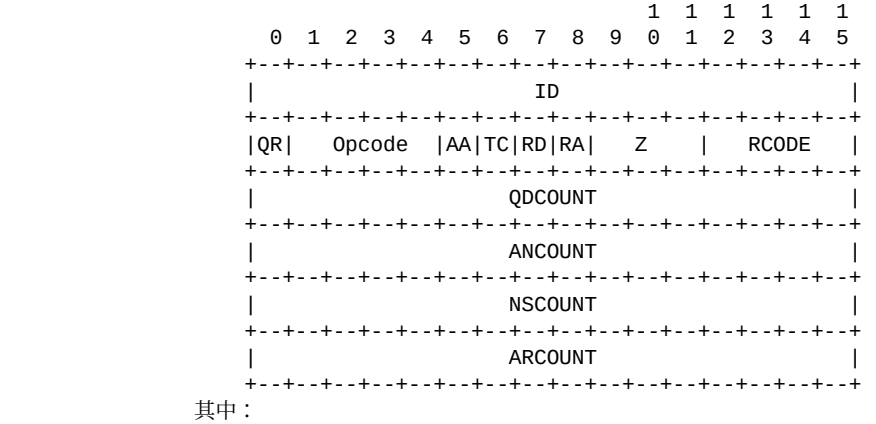
首部（header）段总是有的。首部包括一些字段，这些字段说明剩余的段中哪些将出现，以及消息是查询还是响应，是标准查询还是其他操作码，等等。

首部之后的段的名字取自它们在标准查询中所起的作用。问题（question）段中的字段是描述发给名字服务器的问题的。这些字段是查询类型(QTYPE)、查询类(QCLASS)和查询域名(QNAME)。后面的三个段有相同的格式：一个可能为空的连

续资源记录列表。回答（answer）段包括回答问题的资源记录；权威（authority）段包括指向权威名字服务器的资源记录；附加记录（additional record）段包括的资源记录与查询相关，但不是问题严格的回答。

首部段的格式

（摘自 RFC 1035 的第 26-28 页）



其中：

ID	一个 16 位的标识符，它由产生任意一种查询的程序分配。这个标识符被拷贝到相应的回答，并被请求者用来匹配发出的查询。
QR	一个 1 位字段，它指定该消息是查询(0)，还是响应(1)。
OPCODE	一个 4 位字段，它指定该消息的查询种类。这个值由查询的初始发出者设置，并被复制到响应。这些值是： 0 标准查询 (QUERY) 1 反向查询 (IQUERY) 2 服务器状态请求(STATUS) 3-15 保留将来使用
AA	权威回答 —— 该位在响应中有效，它说明响应的服务器是问题段中的域名的权威。注意，由于别名的缘故，响应段的内容中可能包含多个所有者名字。AA 位对应于查询名字或响应段中第一个所有者的名字。
TC	截断 —— 指定由于长度超过了传输通道允许的范围，这个消息被截断了。
RD	期望递归 —— 这一位可能在查询中设置并拷贝到回答中。如果 RD 被设置，它指示名字服务器进行递归查询。递归查询支持是可选的。
RA	递归可用 —— 这一位在响应中被设置或清除，指示在名字服务器中递归查询支持是否可用。
Z	保留将来使用。在所有的查询和响应中都必须被设置为 0。
RCODE	响应码 —— 这个 4 位字段作为响应的一部分被设置。取值的意义解释如下： 0 没有错误的情况。 1 格式错误 —— 名字服务器不能解释该查询。 2 服务器故障 —— 由于名字服务器自身的问题，不能处理该查询。 3 名字错误 —— 仅对来自权威服务器的响应有意义，该编码表示查询中引用的域名不存在。

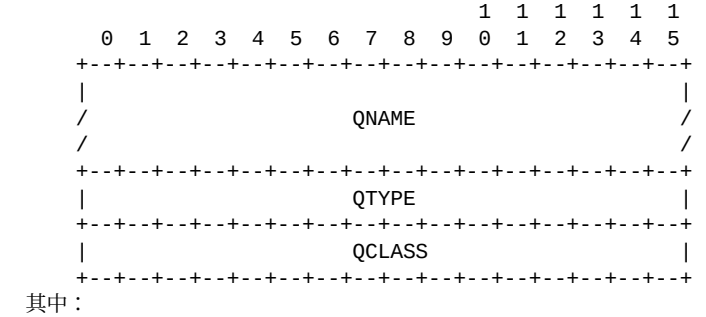
4	没有实现 —— 名字服务器不支持查询请求的种类。
5	拒绝 —— 因为策略原因，名字服务器拒绝执行指定的操作。例如，一个名字服务器可能不希望提供信息给特定的请求者，或者一个名字服务器可能不希望对特定的数据执行特定的操作（例如，区传送）。
6-15	保留将来使用。

QDCOUNT	一个无符号 16 位整数，它指定问题段中的条目数。
ANCOUNT	一个无符号 16 位整数，它指定响应段中的资源记录数。
NSCOUNT	一个无符号 16 位整数，它指定权威记录段中的名字服务器资源记录数。
ARCOUNT	一个无符号 16 位整数，它指定附加记录段中的资源记录数。

问题段格式

（摘自 RFC 1035 的第 28-29 页）

在很多查询中，问题段用来携带“ 问题 ”，也就是定义需要问什么的参数。该段包括 QDCOUNT（通常是 1）个条目，每个条目的格式如下:



其中：

QNAME	一个域名，它被表示为一个标识序列。每个标识由一个长度字节后面加上对应数量的字节构成。域名以根的空标识（一个值为 0 的长度字节）结束。注意，该域可能是奇数字节；它不用填充。
QTYPE	一个双字节码，它指定查询的类型。该字段的值包括所有有效的 TYPE 字段的编码和可匹配多于一个 RR 类型的更一般的编码。
QCLASS	一个双字节码，它指定查询的类。例如，对于 Internet，QCLASS 字段是 IN。

QCLASS 的值

（摘自 RFC 1035 的第 13 页）

QCLASS字段出现在查询的问题段中。QCLASS值是CLASS值的超集；每个CLASS都是有效的 QCLASS。除了 CLASS 值之外，还定义了以下的 QCLASS：

* 255 Any 类

QTYPE 的值

（摘自 RFC 1035 的第 12-13 页）

QTYPE 字段出现在查询的问题段中。QTYPES 是 TYPE 的超集，因此所有的 TYPE 都是有效的 QTYPE。另外，还定义了以下的 QTYPE：

AXFR
: 252 请求传送整个区

MAILB
: 253 请求与邮箱有关的记录（MB、MG 或 MR）

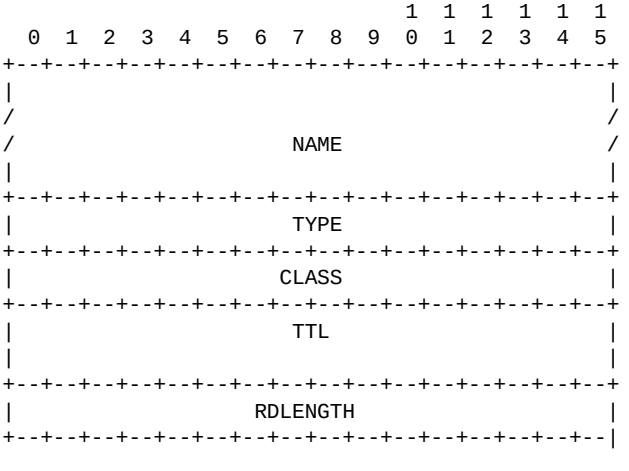
MAILA
: 254 请求邮件代理资源记录（已被废弃 —— 见 MX）

* 255 请求所有的记录

回答、权威和附加段的格式

（摘自 RFC 1035 的第 29-30 页）

回答、权威和附加段具有相同的格式：一组数量可变的资源记录，这个记录的数量在首部中相应的计数字段中指定。每个资源记录的格式如下：



```
    /                     RDATA                     /
    /                                               /
    +--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

其中：

NAME	这个资源记录所属的域名。
TYPE	包含 RR 类型码的双字节。这个字段指定了 RDATA 字段中数据的含义。
CLASS	指定 RDATA 字段中数据种类的双字节。
TTL	一个 32 位的无符号整数，它指定资源记录在被丢弃前可以放在缓存中的时间间隔（以秒为单位）。0 值表示该 RR 只能用于正在进行的事务处理，而不能被放在缓存中。
RDLENGTH	一个 16 位整数，它指定以字节为单位的 RDATA 字段的长度。
RDATA	一个描述资源的可变长度的字节串。该信息的格式根据资源记录的 TYPE 和 CLASS 的不同而变化。例如，如果 TYPE 是 A ，CLASS 是 IN，则 RDATA 字段是一个 4 字节的 ARPA Internet 地址。

数据传输顺序

（摘自 RFC 1035 的第 8-9 页）

本文档描述的首部和数据的传输顺序是在字节水平上决定的。如果用图表表示一组字节，则传输该组字节的顺序就是通常阅读英文的顺序。例如，在下面的图表中，字节按照它们被编号的顺序传输。

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       1       |       2       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       3       |       4       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       5       |       6       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

一个字节表示一个数量，图表中最左边的位是最高位。也就是说，标号为 0 的位是最高的位。例如，下面的图表表示数值 170（十进制）。

```
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|1 0 1 0 1 0 1 0|
+-+-+-+-+-+-+-+-+
```

同样，要是多字节字段表示一个数字量，则整个字段中最左边的位是最高的位。当多字节数被传输时，最高的字节最先传输。

资源记录数据

数据格式

除了双字节和四字节整数值之外，资源记录数据还能包括域名或字符串。

域名

（摘自 RFC 1035 的第 10 页）

消息中的域名以一系列标识来表示。每个标识由一个字节的长度字段后面加上对应数量的字节构成。因为每个域名以根的空标识结束，所以一个域名以一个值为 0 的字节结束。每个长度字节的高两位必须是0，长度字段剩余的6个位限制标识长度小于等于 63 字节。

消息压缩

（摘自 RFC 1035 的第 30 页）

为了缩小消息的尺寸，域系统利用压缩方案消除消息中域名的重复。在该方案中，一个完整的域名或在域名末端的标识列表被指向前面出现过的同一名字的指针取代。

该指针由双字节序列构成：

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| 1  1|                OFFSET                   |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

头两个位都是“ 1 ”。这使得指针与标识区分开来，标识必须以两个“ 0 ”开头，因为标识被限制在63个字节以内（小于等于63字节）。（10和01两种组合被保留为将来使用。） OFFSET字段指定相对于消息开始处（就是域首部中ID字段的第一个字节）的偏移量。0 偏移量指的是 ID 字段的第一个字节，等等。

字符串

（摘自 RFC 1035 的第 13 页）

字符串由一个长度字节后面加上对应数量的字符构成。字符串被作为二进制信息对待，最长可以达到 256 个字符（包括长度字节）。

附录二

BIND 兼容性真值表

表 B-1 表明各个版本的 BIND 分别支持哪些特性。

表 B-1　BIND 兼容性真值表

特性	BIND 版本 4.9.7	8.1.2	8.2.3	9.1.0
支持多处理器				X
动态更新		X	X	X
带 TSIG 签名的动态更新			X	X
基于 TSIG 的更新策略				X
NOTIFY		X	X	X
增量区传送			X	X
转发	X	X	X	X
转发区			X	X
转发器使用 RTT			X	
视图（Views）				X
循环反复（round robin）	X	X	X	X
可配置 RRset 顺序			X	
可配置排序列表	X		X	X
关闭递归	X	X	X	X

表 B-1 BIND 兼容性真值表（续）

特性	BIND 版本 4.9.7	8.1.2	8.2.3	9.1.0
递归访问列表			X	X
IPv6 特性				
AAAA 记录	X	X	X	X
A6 记录				X
DNAME 记录				X
位串标号				X
遵循 DNAME 和 A6 链				X
查询访问列表	X[a]	X	X	X
区传送访问列表	X[b]	X	X	X
DNSSEC 特性				
装载 SIG、KEY 和 NXT 资源记录			X	X
验证“委托链”				X
动态更新保护区				X

a： 通过 secure_zone TXT 资源记录

b： 通过 xfrnets

附录三

在 Linux 上编译和安装 BIND

大多数最近发布的 Linux 中包括的 BIND 的版本都是比较新的 —— 通常是 BIND 8.2.2。BIND 8.2.3 仍然是最新的 BIND 版本，而 ISC 建议你升级到 BIND 9。如果你不想等着 Linux 更新到 BIND 8.2.3 或者 BIND 9.1.0，那么本附录将介绍如何自己升级你的 Linux 下的 BIND 版本。

BIND 8.2.3 安装说明

编译和安装 BIND 8.2.3 非常容易。下面是详细的说明。

获得源代码

首先，要得到源代码，通过匿名 FTP，从 *ftp.isc.org* 就能得到：

```
%  cd /tmp
%   ftp ftp.isc.org.
Connected to isrv4.pa.vix.com.
220 ProFTPD 1.2.0 Server (ISC FTP Server) [ftp.isc.org]
Name (ftp.isc.org.:user): ftp
331 Anonymous login ok, send your complete e-mail address as password.
Password:
230 Anonymous access granted, restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

现在你需要找到正确的文件：

```
ftp  >  cd /isc/bind/src/cur/bind-8
250 CWD command successful.
ftp  > binary
200 Type set to I.
ftp  >  get bind-src.tar.gz
local: bind-src.tar.gz remote: bind-src.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for bind-src.tar.gz (1309147 bytes).
226 Transfer complete.
1309147 bytes received in 23 seconds (56 Kbytes/s)
ftp  > quit
221 Goodbye.
```

解压缩源代码

现在你已经得到被压缩的包含 BIND 源代码的 *tar* 文件。用 *tar* 解压缩：

```
%  tar -zxvf bind-src.tar.gz
```

（这里假设你的 *tar* 版本能处理被 gzip 压缩的文件；如果你没有，可以通过匿名 FTP 从 *ftp.gnu.org/gnu/tar/tar-13.tar.* 得到一个新版本的 *tar*。）这样会产生一个 *src* 目录，它包含几个子目录：*bin*、*include*、*lib* 和 *port*。这些子目录包括的内容如下所示：

bin

全部 BIND 二进制的源代码，包括 *named*。

include

BIND 代码引用的包含文件的拷贝。你需要用它们来建立你的名字服务器，以取代系统原有的，因为它们已经被更新。

lib

BIND 使用的库的源代码。

port

BIND 用来为不同的操作系统定制编译设置和编译选项的信息。

使用适当的编译器设置

编译之前，你需要一个 C 编译器。几乎所有的 Linux 版本都自带 *gcc*，GNU C 编译

器，该编译器一直非常好用。如果你要获得*gcc*，可以从*//www.fsf.org/software/gcc/gcc.html* 上得到相关信息。

默认情况下，BIND假设你使用GNU C编译器和各种其他GNU工具，如*flex*和*byacc*等。对于大部分的 Linux 开发环境来讲这是一个标准的部分。如果你的 Linux 使用其他的程序，你需要修改*/port/linux/Makefile.set*。该文件使得 BIND 知道该使用哪个程序。

编译

接着，从顶级目录开始编译所有代码。首先，运行：

```
%  make stdlinks
```

然后运行：

```
%  make clean
%  make depend
```

这将删除任何以前编译产生的目标文件并更新*Makefile*的从属文件。然后通过运行：

```
% make all
```

编译源代码。源代码的编译应该没有任何错误。然后安装新的*named*和*named-xfer*程序到*/usr/sbin* 目录下。为了完成这些任务，你必须以 root 身份登录。使用命令：

```
#  make install
```

BIND 9.1.0 安装说明

下面是如何在 Linux 主机上编译和安装 BIND 9.1.0。

获得源代码

和 BIND 8.2.3 一样，我们首先应该得到源代码。这次你还是需要FTP到*ftp.isc.org*：

```
% cd /tmp
% ftp ftp.isc.org.
Connected to isrv4.pa.vix.com.
220 ProFTPD 1.2.1 Server (ISC FTP Server) [ftp.isc.org]
Name (ftp.isc.org.:user): ftp
331 Anonymous login ok, send your complete email address as your password.
Password:
230 Anonymous access granted, restrictions apply.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

切换到正确的目录，然后找到你所需要的文件：

```
ftp> cd /isc/bind9/9.1.0/
250 CWD command successful.
ftp> get bind-9.1.0.tar.gz
local: bind-9.1.0.tar.gz remote: bind-9.1.0.tar.gz
200 PORT command successful.
150 Opening BINARY mode data connection for bind-9.1.0.tar.gz (3299471 bytes).
226 Transfer complete.
3299471 bytes received in 92.4 secs (35 Kbytes/sec)
ftp> quit
221 Goodbye.
```

解压缩源代码

用 *tar* 命令解压缩 *tar* 文件：

```
% tar zxvf bind-9.1.0.tar.gz
```

与 BIND 8.2.3 不同的是，该命令会在工作目录下为所有的 BIND 源代码创建一个 *bind-9.1.0* 子目录。（BIND 8 总是把文件解压缩到当前工作目录下。）*bind-9.1.0* 子目录将包含以下子目录：

bin

包含所有 BIND 的二进制源代码，包括 *named* 在内。

contrib

一些工具。

doc

BIND 的文档，包括非常有价值的管理员资源手册。

lib

BIND 使用的库的源代码。

make

makefile 文件。

运行配置和编译

这部分也不同于 BIND 8，BIND 9 使用了让人不可思议的 *configure* 脚本来设置正确的 *include* 文件和编译器设置。你可以阅读 README 文件，然后决定是否需要进行特殊的设置。*configure* 支持命令行选项，这允许你不使用线程、使用不同的安装目录或者更多的不同设置来进行编译。运行 *configure*：

```
%   ./configure
```

或者，例如你不想使用线程，可以使用如下命令：

```
%   ./configure disable-threads
```

编译 BIND，键入：

```
%  make all
```

源代码的编译应该没有问题。安装 BIND，以 root 身份键入：

```
#   make install
```

到此为止，所有工作全部完成！

附录四

顶级域

本表列出了所有双字符国家（地区）代码和所有非国家顶级域。本书完成时，并非所有的国家（地区）都在 Internet 的域名空间进行了注册，但是缺的并不多。

域	国家（地区）或组织
AC	阿森松岛（Ascension）
AD	安道尔共和国（Andorra）
AE	阿联酋（United Arab Emirates）
AF	阿富汗（Afghanistan）
AG	安提瓜和巴布达（Antigua and Barbuda）
AI	安圭拉（Anguilla）
AL	阿尔巴尼亚（Albania）
AM	亚美尼亚（Armenia）
AN	荷属安的列斯群岛（Netherlands Antilles）
AO	安哥拉（Angola）
AQ	南极洲（Antarctica）
AR	阿根廷（Argentina）
ARPA	（美国国防部）高级研究计划署网络（ARPA Internet）
AS	美属萨摩亚群岛（American Samoa）
AT	奥地利（Austria）

域	国家（地区）或组织
AU	澳大利亚（Australia）
AW	阿鲁巴（Aruba）
AZ	阿塞拜疆（Azerbaijan）
BA	波黑（Bosnia and Herzegovina）
BB	巴巴多斯（Barbados）
BD	孟加拉（Bangladesh）
BE	比利时（Belgium）
BF	布基纳法索（Burkina Faso）
BG	保加利亚（Bulgaria）
BH	巴林（Bahrain）
BI	布隆迪（Burundi）
BJ	贝宁（Benin）
BM	百慕大（Bermuda）
BN	文莱（Brunei Darussalam）
BO	玻利维亚（Bolivia）
BR	巴西（Brazil）
BS	巴哈马群岛（Bahamas）
BT	不丹（Bhutan）
BV	布维岛（Bouvet Island）
BW	博茨瓦纳（Botswana）
BY	白俄罗斯（Belarus）
BZ	伯利兹（Belize）
CA	加拿大（Canada）
CC	科科斯群岛（Cocos Keeling Islands）
CD	刚果民主共和国（Congo, Democratic Republic of the）
CF	中非共和国（Central African Republic）
CG	刚果（Congo）
CH	瑞士（Switzerland）
CI	科特迪瓦（Cote D'Ivoire）
CK	库克群岛（Cook Islands）
CL	智利（Chile）
CM	喀麦隆（Cameroon）

域	国家（地区）或组织
CN	中国（China）
CO	哥伦比亚（Colombia）
COM	通用（以前的商业）
CR	哥斯达黎加（Costa Rica）
CU	古巴（Cuba）
CV	佛得角（Cape Verde）
CX	圣诞岛（Christmas Island）
CY	塞浦路斯（Cyprus）
CZ	捷克共和国（Czech Republic）
DE	德国（Germany）
DJ	吉布提（Djibouti）
DK	丹麦（Denmark）
DM	多米尼加联邦（Dominica）
DO	多米尼加共和国（Dominican Republic）
DZ	阿尔及利亚（Algeria）
EC	厄瓜多尔（Ecuador）
EDU	教育（education）
EE	爱沙尼亚（Estonia）
EG	埃及（Egypt）
EH	西撒哈拉（Western Sahara）
ER	厄立特里亚（Eritrea）
ES	西班牙（Spain）
ET	埃塞俄比亚（Ethiopia）
FI	芬兰（Finland）
FJ	斐济（Fiji）
FK	福克兰群岛(即马尔维纳斯群岛)[Falkland Islands (Malvinas)]
FM	密克罗尼西亚联邦（Micronesia, Federated States of）
FO	法罗群岛（Faroe Islands）
FR	法国（France）
FX	法国大都会（France, Metropolitan）
GA	加蓬（Gabon）
GB	英国（United Kingdom）注

域	国家（地区）或组织
GD	格林纳达（Grenada）
GE	格鲁吉亚（Georgia）
GF	法属圭亚那地区（French Guiana）
GG	格恩西岛、奥尔德尼岛、萨克岛（英吉利海峡群岛）
GH	加纳（Ghana）
GI	直布罗陀（Gibraltar）
GL	格陵兰（Greenland）
GM	冈比亚（Gambia）
GN	几内亚（Guinea）
GOV	政府（government）
GP	瓜德罗普（Guadeloupe）
GQ	赤道几内亚（Equatorial Guinea）
GR	希腊（Greece）
GS	南乔治亚和南三明治群岛（South Georgia and the South Sandwich Islands）
GT	危地马拉（Guatemala）
GU	关岛（Guam）
GW	几内亚比绍（Guinea-Bissau）
GY	圭亚那（Guyana）
HK	中国香港特别行政区（Hong Kong）
HM	赫德和麦克唐纳岛群（Heard and McDonald Islands）
HN	洪都拉斯（Honduras）
HR	克罗地亚（Croatia）
HT	海地（Haiti）
HU	匈牙利（Hungary）
ID	印度尼西亚（Indonesia）
IE	爱尔兰（Ireland）
IL	以色列（Israel）
IN	印度（India）
INT	国际组织（international entities）
IO	英属印度洋领地（British Indian Ocean Territory）
IQ	伊拉克（Iraq）
IR	伊朗（Iran）

域	国家（地区）或组织
IS	冰岛（Iceland）
IT	意大利（Italy）
JE	泽西岛（Jersey）
JM	牙买加（Jamaica）
JO	约旦（Jordan）
JP	日本（Japan）
KE	肯尼亚（Kenya）
KG	吉尔吉斯斯坦（Kyrgyzstan）
KH	柬埔寨（Cambodia）
KI	基里巴斯（Kiribati）
KM	科摩罗（Comoros）
KN	圣基茨和尼维斯（Saint Kitts and Nevis）
KP	朝鲜人民民主主义共和国（Korea, Democratic People's Republic of）
KR	韩国（Korea, Republic of）
KW	科威特（Kuwait）
KY	开曼群岛（Cayman Islands）
KZ	哈萨克斯坦（Kazakhstan）
LA	老挝人民民主主义共和国（Lao People's Democratic Republic）
LB	黎巴嫩（Lebanon）
LC	圣卢西亚（Saint Lucia）
LI	列支敦士登（Liechtenstein）
LK	斯里兰卡（Sri Lanka）
LR	利比里亚（Liberia）
LS	莱索托（Lesotho）
LT	立陶宛（Lithuania）
LU	卢森堡（Luxembourg）
LV	拉脱维亚（Latvia）
LY	利比亚（Libyan）
MA	摩洛哥（Morocco）
MC	摩纳哥（Monaco）
MD	摩尔多瓦共和国（Moldova, Republic of）

域	国家（地区）或组织
MG	马达加斯加（Madagascar）
MH	马绍尔群岛（Marshall Islands）
MIL	军事（military）
MK	前南斯拉夫马其顿共和国（Macedonia, the Former Yugoslav Republic of）
ML	马里（Mali）
MM	缅甸（Myanmar）
MN	蒙古（Mongolia）
MO	中国澳门特别行政区（Macau）
MP	北马里亚纳群岛（Northern Mariana Islands）
MQ	马提尼克岛（Martinique）
MR	毛里塔尼亚（Mauritania）
MS	蒙特塞拉特岛（Montserrat）
MT	马耳他（Malta）
MU	毛里求斯（Mauritius）
MV	马尔代夫（Maldives）
MW	马拉维（Malawi）
MX	墨西哥（Mexico）
MY	马来西亚（Malaysia）
MZ	莫桑比克（Mozambique）
NA	纳米比亚（Namibia）
NATO	北大西洋公约组织（North Atlantic Treaty Organization）
NC	新喀里多尼亚（New Caledonia）
NE	尼日尔（Niger）
NET	网络组织（networking organizations）
NF	诺福克岛（Norfolk Island）
NG	尼日利亚（Nigeria）
NI	尼加拉瓜（Nicaragua）
NL	荷兰（Netherlands）
NO	挪威（Norway）
NP	尼泊尔（Nepal）
NR	瑙鲁（Nauru）

域	国家（地区）或组织
NU	纽埃（Niue）
NZ	新西兰（New Zealand）
OM	阿曼（Oman）
ORG	组织（organizations）
PA	巴拿马（Panama）
PE	秘鲁（Peru）
PF	法属玻利尼西亚（French Polynesia）
PG	巴布亚新几内亚（Papua New Guinea）
PH	菲律宾共和国（Philippines）
PK	巴基斯坦（Pakistan）
PL	波兰（Poland）
PM	圣皮埃尔和密克隆群岛（St. Pierre and Miquelon）
PN	皮特克恩岛（Pitcairn）
PR	波多黎各（Puerto Rico）
PS	巴勒斯坦（Palestinian Authority）
PT	葡萄牙（Portugal）
PW	帕劳群岛（Palau）
PY	巴拉圭（Paraguay）
QA	卡塔尔（Qatar）
RE	留尼汪（Reunion）
RO	罗马尼亚（Romania）
RU	俄罗斯联邦（Russian Federation）
RW	卢旺达（Rwanda）
SA	沙特阿拉伯（Saudi Arabia）
SB	所罗门群岛（Solomon Islands）
SC	塞舌尔（Seychelles）
SD	苏丹（Sudan）
SE	瑞典（Sweden）
SG	新加坡（Singapore）
SH	圣海伦娜（St. Helena）
SI	斯洛文尼亚（Slovenia）
SJ	斯瓦尔巴特和扬马延群岛（Svalbard and Jan Mayen Islands）

域	国家（地区）或组织
SK	斯洛伐克（Slovakia）
SL	塞拉利昂（Sierra Leone）
SM	圣马力诺（San Marino）
SN	塞内加尔（Senegal）
SO	索马里（Somalia）
SR	苏里南（Suriname）
ST	圣多群美和普林西比（Sao Tome and Principe）
SU	前苏联（Union of Soviet Socialist Republics）
SV	萨尔瓦多（El Salvador）
SY	叙利亚阿拉伯共和国（Syrian Arab Republic）
SZ	斯威士兰（Swaziland）
TC	特克斯和凯科斯群岛（Turks and Caicos Islands）
TD	乍得（Chad）
TF	法国南部（French Southern Territories）
TG	多哥（Togo）
TH	泰国（Thailand）
TJ	塔吉克斯坦（Tajikistan）
TK	托克劳（Tokelau）
TM	土库曼斯坦（Turkmenistan）
TN	突尼斯（Tunisia）
TO	汤加（Tonga）
TP	东帝汶（East Timor）
TR	土耳其（Turkey）
TT	特立尼达和多巴哥（Trinidad and Tobago）
TV	图瓦卢（Tuvalu）
TW	中国台湾省（Taiwan, Province of China）
TZ	坦桑尼亚联合共和国（Tanzania, United Republic of）
UA	乌克兰（Ukraine）
UG	乌干达（Uganda）
UK	联合王国（United Kingdom）
UM	美国外岛（United States Minor Outlying Islands）
US	美国（United States）

域	国家（地区）或组织
UY	乌拉圭（Uruguay）
UZ	乌兹别克斯坦（Uzbekistan）
VA	梵蒂冈（Vatican City State）
VC	圣文森特和格林纳丁斯（Saint Vincent and The Grenadines）
VE	委内瑞拉（Venezuela）
VG	英属维尔京群岛［Virgin Islands (British)］
VI	美属维尔京群岛［Virgin Islands (U.S.)］
VN	越南（Vietnam）
VU	瓦努阿图（Vanuatu）
WF	瓦利斯和富图纳群岛 Wallis and Futuna Islands
WS	萨摩亚群岛（Samoa）
YE	也门（Yemen）
YT	马约特（Mayotte）
YU	南斯拉夫（Yugoslavia）
ZA	南非（South Africa）
ZM	赞比亚（Zambia）
ZR	扎伊尔（Republic of Zaire）
ZW	津巴布韦（Zimbabwe）

注： 实际上，英国（United Kingdom）用“ UK ”作为它的顶级域。

附录五

BIND 名字服务器和解析器配置

BIND 名字服务器引导文件指令和配置文件语句

这里是 BIND 名字服务器引导文件指令和配置文件语句的一个简明列表，也包括 BIND 解析器的配置指令。有些指令和语句只在较新的版本中才存在，因此你的服务器可能还不支持它们。对新的指令或语句都给出了引入它们的具体的版本号（比如，8.2+）。如果语句已经存在很长时间，则不进行标注。

BIND 4 引导文件指令

directory

功能：

改变当前工作目录

语法：

`directory` *`new-directory`*

例子：

```
directory /var/named
```

参见：

8.X.X 和 9.X..X *options* 语句，*directory* 子语句

讲述章节：第四章

primary

功能：

将一个名字服务器配置为区的主名字服务器

语法：

```
primary domain-name-of-zone file
```

例子：

```
primary  movie.edu   db.movie.edu
```

参见：

8.X.X 和9.X.X *zone* 语句，类型*master*

讲述章节：第四章

secondary

功能：

将一个名字服务器配置为区的辅名字服务器

语法：

```
secondary domain-name-of-zone ip-address-list [backup-file]
```

例子：

```
secondary  movie.edu   192.249.249.3 bak.movie.edu
```

参见：

8.X.X 和9.X.X *zone* 语句，类型*slave*

讲述章节：第四章

cache

功能：

定义加载根线索（根名字服务器的名字和地址）的文件的名称

语法：

```
cache    .    file
```

例子：

```
cache    .    db.cache
```

参见：

8.X.X 和 9.X.X *zone* 语句，类型 *hint*

讲述章节：第四章

forwarders

功能：

配置未解析的查询所发往的名字服务器

语法：

```
forwarders ip-address-list
```

例子：

```
forwarders 192.249.249.1 192.249.249.3
```

参见：

8.X.X 和 9.X.X *options* 语句，*forwarders* 子语句

讲述章节：第十章

sortlist

功能：

指定优先的网络

语法：

```
sortlist network-list
```

例子：

```
sortlist 10.0.0.0
```

参见：

8.2+ 和 9.1.0+ *options* 语句，*sortlist* 子语句

讲述章节：第十章

slave

该语句与 4.9.X 的指令 *options forward-only* 以及 8.X.X 和 9.X.X 的 *options* 子语句 *forward* 相同。

include (4.9+)

功能：

在 *named.boot* 中包括其他文件的内容

语法：

include *file*

例子：

```
include  bootfile.primary
```

参见：

8.X.X 和 9.X.X *include* 语句

讲述章节：第七章

stub (4.9+)

功能：

指定你的名字服务器应该定时获得授权信息的孩子区

语法：

stub *domain-name-of-zone ip-address-list* [*backup-file*]

例子：

```
stub movie.edu 192.249.249.3 stub.movie.edu
```

参见：

8.X.X 和 9.X.X *zone* 语句，类型 *stub*

讲述章节：第九章

options (4.9+)

options forward-only

功能：

防止你的名字服务器不使用转发器而自己解析域名

参见：

8.X.X 和 9.X.X *option* 语句，*forward* 子语句

讲述章节：第十章

options no-recursion

功能：

防止你的名字服务器执行域名的递归解析

参见：

8.X.X 和 9.X.X *options* 语句，*recursion* 子语句

讲述章节：第十章和第十一章

options no-fetch-glue

功能：

防止你的名字服务器在构造响应时加入 glue 记录

参见：

8.X.X *options* 语句，*fetch-glue* 子语句

讲述章节：第十章和第十一章

options query-log

功能：

记录你的服务器收到的所有查询

参见：

8.X.X 和9.1.0+ *logging* 语句，目录 *queries*

讲述章节：第七章和第十四章

options fake-iquery

功能：

告诉你的名字服务器用假的回答（替代错误）响应老式的反向查询

参见：

8.X.X *options* 语句，*fake-iquery*子语句

讲述章节：第十二章

limit (4.9+)

limit transfers-in

功能：

限制你的名字服务器同时进行的区传送的总数

参见：

8.X.X 和9.X.X *options* 语句，*transfers-in* 子语句

limit transfers-per-ns

功能：

限制你的名字服务器从任意一个服务器同时请求的区传送的总数

参见：

8.X.X 和9.X.X *options* 语句，*transfers-per-ns* 子语句

limit datasize

功能：

增加 *named* 使用的数据段的大小（只在某些操作系统上有效）

参见：

8.X.X和9.1.0+ *options* 语句， *datasize*子语句

讲述上述所有内容的章节：第十章

xfrnets (4.9+)

功能：

限制从你的名字服务器到IP地址列表或网络的区传送总数

语法：

```
xfrnets ip-address-or-network-list
```

例子：

```
xfrnets 15.0.0.0 128.32.0.0
```

参见：

8.X.X 和9.X.X *options*和*zone*语句， *allow-transfer*子语句

讲述章节：第十一章

bogusns (4.9+)

功能：

告诉你的名字服务器不要查询给出错误回答的服务器列表

语法：

```
bogusns ip-address-list
```

例子：

```
bogusns 15.255.152.4
```

参见：

8.X.X和9.1.0+ *server*语句， *bogus*子语句

讲述章节：第十章

check-names (4.9.4+)

功能：

配置名字检查机制

语法：

```
check-names primary|secondary|response fail|warn|ignore
```

例子：

```
check-names primary ignore
```

参见：

8.X.X *options* 和 *zone* 语句，*check-names* 子语句

讲述章节：第四章

BIND 8 配置文件语句

acl

功能：

生成命名的地址匹配列表

语法：

```
acl name {
   address_match_list;
};
```

讲述章节：第十章和第十一章

controls(8.2+)

功能：

配置 *ndc* 用来控制名字服务器的通道

语法：

```
controls {
   [ inet ( ip_addr | * ) port ip_port allow address_match_list; ]
   [ unix path_name perm number owner number group number; ]
};
```

讲述章节：第七章

include

功能：

将指定的文件插入遇到 *include* 语句的地方

语法：

```
include path_name;
```

讲述章节：第七章

key (8.2+)

功能：

定义密钥 ID（key ID），它能用于 *server* 语句或者将 TSIG 密钥同某个名字服务器联系起来的地址匹配列表。

语法：

```
key key_id {
  algorithm algorithm_id;
  secret secret_string;
};
```

讲述章节：第十章和第十一章

logging

功能：

配置名字服务器的日志行为

语法：

```
logging {
```

```
  [ channel channel_name {
    ( file path_name
       [ versions ( number | unlimited ) ]
       [ size size_spec ]
     | syslog ( kern | user | mail | daemon | auth | syslog | lpr |
                news | uucp | cron | authpriv | ftp |
                local0 | local1 | local2 | local3 |
                local4 | local5 | local6 | local7 )
     | null );

    [ severity ( critical | error | warning | notice |
                 info  | debug [ level ] | dynamic ); ]
    [ print-category yes_or_no; ]
    [ print-severity yes_or_no; ]
    [ print-time yes_or_no; ]
  }; ]

  [ category category_name {
    channel_name; [ channel_name; ... ]
  }; ]
  ...
};
```

讲述章节：第七章

options

功能：

配置全局选项

语法：

```
options {
  [ allow-query { address_match_list }; ]
  [ allow-recursion { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ also-notify { ip_addr; [ ip_addr; ... ] }; ]
  [ auth-nxdomain yes_or_no; ]
  [ blackhole { address_match_list }; ]
  [ check-names ( master | slave | response ) ( warn | fail | ignore ); ]
  [ cleaning-interval number; ]
  [ coresize size_spec; ]
  [ datasize size_spec; ]
  [ deallocate-on-exit yes_or_no; ]
  [ dialup yes_or_no; ]
  [ directory path_name; ]
  [ dump-file path_name; ]
  [ fake-iquery yes_or_no; ]
  [ fetch-glue yes_or_no; ]
  [ files size_spec; ]
```

```
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr ; [ ip_addr ; ... ] ] }; ]
  [ has-old-clients yes_or_no; ]
  [ heartbeat-interval number; ]
  [ host-statistics yes_or_no; ]
  [ interface-interval number; ]
  [ lame-ttl number; ]
  [ listen-on [ port ip_port ] { address_match_list }; ]
  [ maintain-ixfr-base yes_or_no; ]
  [ max-ixfr-log-size number; ]
  [ max-ncache-ttl number; ]
  [ max-transfer-time-in number; ]
  [ memstatistics-file path_name; ]
  [ min-roots number; ]
  [ multiple-cnames yes_or_no; ]
  [ named-xfer path_name; ]
  [ notify yes_or_no; ]
  [ pid-file path_name; ]
  [ query-source [ address ( ip_addr | * ) ] [ port ( ip_port | * ) ]; ]
  [ recursion yes_or_no; ]
  [ rfc2308-type1 yes_or_no; ]
  [ rrset-order { order_spec; [ order_spec; ... ] }; ]
  [ serial-queries number; ]
  [ sortlist { address_match_list }; ]
  [ stacksize size_spec; ]
  [ statistics-file path_name; ]
  [ statistics-interval number; ]
  [ topology { address_match_list }; ]
  [ transfer-format ( one-answer | many-answers ); ]
  [ transfer-source ( ip_addr | * ); ]
  [ transfers-in  number; ]
  [ transfers-per-ns number; ]
  [ treat-cr-as-space yes_or_no; ]
  [ use-id-pool yes_or_no; ]
  [ use-ixfr yes_or_no; ]
  [ version version_string; ]
};
```

讲述章节：第四章、第十章、第十一章和第十六章

server

功能：

定义与远程名字服务器有关的特性

语法：

```
server ip_addr {
  [ bogus yes_or_no; ]
  [ keys { key_id [ key_id ... ] }; ]
```

```
  [ support-ixfr yes_or_no; ]
  [ transfer-format ( one-answer | many-answers ); ]
};
```

讲述章节：第十章和第十一章

trusted-keys(8.2+)

功能：

配置用于 DNSSEC 的安全根的公共密钥

语法：

```
trusted-keys {
  domain-name flags protocol_id algorithm_id public_key_string;
  [ domain-name flags protocol_id algorithm_id public_key_string; [ ... ] ]
};
```

讲述章节：第十一章

zone

功能：

配置由名字服务器维护的区

语法：

```
zone "domain_name" [ ( in | hs | hesiod | chaos ) ] {
  type master;
  file path_name;
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-update { address_match_list }; ]
  [ also-notify { ip_addr; [ ip_addr; ... ]
  [ check-names ( warn | fail | ignore ); ]
  [ dialup yes_or_no | notify; ]
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr; [ ip_addr; ... ] ] }; ]
  [ ixfr-base path_name; ]
  [ ixfr-tmp-file path_name; ]
  [ maintain-ixfr-base yes_or_no; ]
  [ notify yes_or_no; ]
  [ pubkey flags protocol_id algorithm_id public_key_string; ]
};

zone "domain_name" [ ( in | hs | hesiod | chaos ) ] {
  type slave;
```

```
  masters [ port ip_port ] { ip_addr; [ ip_addr; ... ] };
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-update { address_match_list }; ]
  [ also-notify { ip_addr; [ ip_addr; ... ] };
  [ check-names ( warn | fail | ignore ); ]
  [ dialup yes_or_no; ]
  [ file path_name; ]
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr; [ ip_addr; ... ] ] }; ]
  [ ixfr-base path_name; ]
  [ max-transfer-time-in number; ]
  [ notify yes_or_no; ]
  [ pubkey flags protocol_id algorithm_id public_key_string; ]
  [ transfer-source ip_addr; ]
};

zone "domain_name" [ ( in | hs | hesiod | chaos ) ] {
  type stub;
  masters [ port ip_port ] { ip_addr; [ ip_addr; ... ] };
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-update { address_match_list }; ]
  [ check-names ( warn | fail | ignore ); ]
  [ dialup yes_or_no; ]
  [ file path_name; ]
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr ; [ ip_addr ; ... ] ] }; ]
  [ max-transfer-time-in number; ]
  [ pubkey flags protocol_id algorithm_id public_key_string; ]
  [ transfer-source ip_addr; ]
};

zone "domain_name" [ ( in | hs | hesiod | chaos ) ] {
  type forward;
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr ; [ ip_addr ; ... ] ] }; ]
};

zone "." [ ( in | hs | hesiod | chaos ) ] {
  type hint;
  file path_name;
  [ check-names ( warn | fail | ignore ); ]
};
```

讲述章节：第四章和第十章

BIND 9 配置文件语句

acl

功能：

创建一个命名的地址匹配列表

语法：

```
acl name {
   address_match_list;
};
```

讲述章节：第十章和第十一章

controls

功能：

配置 *ndc* 用来控制名字服务器的通道

语法：

```
controls {
   [ inet ( ip_addr | * ) port ip_port allow address_match_list keys key_list; ]
   [ inet ... ; ]
};
```

讲述章节：第七章维护

include

功能：

将指定的文件插入遇到 *include* 语句的地方

语法：

```
include path_name;
```

讲述章节：第七章

key

功能：

定义密钥 ID（key ID），它能用于 *server* 语句或者将 TSIG 密钥同某个名字服务器联系起来的地址匹配列表

语法：

```
key key_id {
  algorithm algorithm_id;
  secret secret_string;
};
```

讲述章节：第十章和第十一章

logging

功能：

配置名字服务器的日志行为

语法：

```
logging {
  [ channel channel_name {
    ( file path_name
       [ versions ( number | unlimited ) ]
       [ size size_spec ]
     | syslog ( kern | user | mail | daemon | auth | syslog | lpr |
                news | uucp | cron | authpriv | ftp |
                local0 | local1 | local2 | local3 |
                local4 | local5 | local6 | local7 )
     | stderr
     | null );

    [ severity ( critical | error | warning | notice |
                 info  | debug [ level ] | dynamic ); ]
    [ print-category yes_or_no; ]
    [ print-severity yes_or_no; ]
    [ print-time yes_or_no; ]
  }; ]

  [ category category_name {
    channel_name; [ channel_name; ... ]
  }; ]
  ...
};
```

讲述章节：第七章

options

功能：

配置全局选项

语法：

```
options {
  [ additional-from-auth yes_or_no; ]
  [ additional-from-cache yes_or_no; ]
  [ allow-notify { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-recursion { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ also-notify { ip_addr [ port ip_port ] ; [ ip_addr [ port ip_port ] ; ... ] };
]
  [ auth-nxdomain yes_or_no; ]
  [ blackhole { address_match_list }; ]
  [ cleaning-interval number; ]
  [ coresize size_spec; ]
  [ datasize size_spec; ]
  [ dialup yes_or_no; ]
  [ directory path_name; ]
  [ dump-file path_name; ]
  [ files size_spec; ]
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr ; [ ip_addr ; ... ] ] }; ]
  [ heartbeat-interval number; ]
  [ interface-interval number; ]
  [ lame-ttl number; ]
  [ listen-on [ port ip_port ] { address_match_list }; ]
  [ listen-on-v6 [ port ip_port ] { address_match_list }; ]
  [ max-cache-ttl number; ]
  [ max-ncache-ttl number; ]
  [ max-refresh-time number; ]
  [ max-retry-time number; ]
  [ max-transfer-idle-in number; ]
  [ max-transfer-idle-out number; ]
  [ max-transfer-time-in number; ]
  [ max-transfer-time-out number; ]
  [ min-refresh-time number; ]
  [ min-retry-time number; ]
  [ notify yes_or_no | explicit; ]
  [ notify-source ( ip_addr | * ) [ port ip_port ]; ]
  [ notify-source-v6 ( ip6_addr | * ) [ port ip_port ]; ]
  [ pid-file path_name; ]
  [ port ip_port; ]
  [ query-source [ address ( ip_addr | * ) ] [ port ( ip_port | * ) ]; ]
  [ query-source-v6 [ address ( ip6_addr | * ) ] [ port ( ip_port | * ) ]; ]
  [ recursion yes_or_no; ]
  [ recursive-clients number; ]
```

```
  [ sig-validity-interval number; ]
  [ sortlist { address_match_list }; ]
  [ stacksize size_spec; ]
  [ statistics-file path_name; ]
  [ tcp-clients number; ]
  [ tkey-dhkey key_name key_tag; ]
  [ tkey-domain domain_name; ]
  [ transfer-format ( one-answer | many-answers ); ]
  [ transfer-source ( ip_addr | * ) [ port ip_port ]; ]
  [ transfer-source-v6 ( ip6_addr | * ) [ port ip_port ]; ]
  [ transfers-in  number; ]
  [ transfers-out number; ]
  [ transfers-per-ns number; ]
  [ version version_string; ]
  [ zone-statistics yes_or_no; ]
};
```

讲述章节：第四章、第十章、第十一章和第十六章

sever

功能：

定义与远程名字服务器有关的特性

语法：

```
server ip_addr {
  [ bogus yes_or_no; ]
  [ keys { key_id [ key_id ... ] }; ]
  [ provide-ixfr yes_or_no; ]
  [ request-ixfr yes_or_no; ]
  [ transfers number; ]
  [ transfer-format ( one-answer | many-answers ); ]
};
```

讲述章节：第十章和第十一章

trusted-keys

功能：

配置用于 DNSSEC 的安全根的公共密钥

语法：

```
trusted-keys {
  domain-name flags protocol_id algorithm_id public_key_string;
```

```
    [ domain-name flags protocol_id algorithm_id public_key_string; [ ... ] ]
};
```

讲述章节：第十一章

view

功能：

创建并配置一个视图

语法：

```
view "view_name" [ ( in | hs | hesiod | chaos ) ] {
  match-clients { address_match_list };
  [ allow-notify { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-recursion { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ also-notify { ip_addr; [ ip_addr; ... ] }; ]
  [ auth-nxdomain yes_or_no; ]
  [ cleaning-interval number; ]
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr; [ ip_addr; ... ] ] }; ]
  [ key ... ]
  [ lame-ttl number; ]
  [ min-refresh-time number; ]
  [ min-retry-time number; ]
  [ max-cache-ttl number; ]
  [ max-ncache-ttl number; ]
  [ max-transfer-idle-out number; ]
  [ max-transfer-time-out number; ]
  [ max-refresh-time number; ]
  [ max-retry-time number; ]
  [ notify yes_or_no | explicit; ]
  [ provide-ixfr yes_or_no; ]
  [ query-source [ address ( ip_addr | * ) ] [ port ( ip_port | * ) ]; ]
  [ query-source-v6 [ address ( ip6_addr | * ) ] [ port ( ip_port | * ) ]; ]
  [ recursion yes_or_no; ]
  [ request-ixfr yes_or_no; ]
  [ server ... ]
  [ sig-validity-interval number; ]
  [ sortlist { address_match_list }; ]
  [ transfer-format ( one-answer | many-answers ); ]
  [ transfer-source ( ip_addr | * ) [ port ip_port ]; ]
  [ transfer-source-v6 ( ip6_addr | * ) [ port ip_port ]; ]
  [ trusted-keys ... ]
  [ zone ... ]
};
```

讲述章节：第十章和第十一章

zone

功能：

配置名字服务器维护的区

语法：

```
zone "domain_name" [ ( in | hs | hesiod | chaos ) ] {
  type master;
  file path_name;
  [ allow-notify { address_match_list }; ]
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-update { address_match_list }; ]
  [ allow-update-forwarding { address_match_list }; ]
  [ also-notify { ip_addr [ port ip_port ]; [ ip_addr [ port ip_port ]; ... ]
  [ database string; [ string; ... ] ]
  [ dialup yes_or_no | notify; ]
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr; [ ip_addr; ... ] ] }; ]
  [ max-refresh-time number; ]
  [ max-retry-time number; ]
  [ max-transfer-idle-out number; ]
  [ max-transfer-time-out number; ]
  [ min-refresh-time number; ]
  [ min-retry-time number; ]
  [ notify yes_or_no | explicit; ]
  [ sig-validity-interval number; ]
  [ update-policy { update_policy_rule; [ ... ] }; ]
};

zone "domain_name" [ ( in | hs | hesiod | chaos ) ] {
  type slave;
  masters [ port ip_port ] { ip_addr [ port ip_port ] [ key key_id ]; [
ip_addr [ port ip_port ] [ key key_id ]; ... ] };
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-update { address_match_list }; ]
  [ allow-update-forwarding { address_match_list }; ]
  [ also-notify { ip_addr [ port ip_port ]; [ ip_addr [ port ip_port ]; ... ]
};
  [ dialup yes_or_no | notify | notify-passive | refresh | passive; ]
  [ file path_name; ]
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr; [ ip_addr; ... ] ] }; ]
  [ max-refresh-time number ; ]
  [ max-retry-time number ; ]
  [ max-transfer-idle-in number; ]
  [ max-transfer-idle-out number; ]
  [ max-transfer-time-in number; ]
  [ max-transfer-time-out number; ]
  [ min-refresh-time number ; ]
```

```
  [ min-retry-time number ; ]
  [ notify yes_or_no | explicit; ]
  [ transfer-source ( ip_addr | * ) [ port ip_port ]; ]
  [ transfer-source-v6 ( ip6_addr | * ) [ port ip_port ]; ]
};

zone "domain_name" [ ( in | hs | hesiod | chaos ) ] {
  type stub;
  masters [ port ip_port ] { ip_addr [ [port ip_port ] [ key key_id ]; [ ip_addr
[ port ip_port ] [ key key_id ]; ... ] };
  [ allow-query { address_match_list }; ]
  [ allow-transfer { address_match_list }; ]
  [ allow-update { address_match_list }; ]
  [ allow-update-forwarding { address_match_list }; ]
  [ dialup yes_or_no | passive | refresh; ]
  [ file path_name; ]
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr ; [ ip_addr ; ... ] ] }; ]
  [ max-refresh-time number ; ]
  [ max-retry-time number ; ]
  [ max-transfer-idle-in number; ]
  [ max-transfer-idle-out number; ]
  [ max-transfer-time-in number; ]
  [ max-transfer-time-out number; ]
  [ min-refresh-time number ; ]
  [ min-retry-time number ; ]
  [ transfer-source ( ip_addr | * ) [ port ip_port ]; ]
  [ transfer-source-v6 ( ip6_addr | * ) [ port ip_port ]; ]
};

zone "domain_name" [ ( in | hs | hesiod | chaos ) ] {
  type forward;
  [ forward ( only | first ); ]
  [ forwarders { [ ip_addr ; [ ip_addr ; ... ] ] }; ]
};

zone "." [ ( in | hs | hesiod | chaos ) ] {
  type hint;
  file path_name;
};
```

讲述章节：第四章和第十章

BIND 解析器语句

下面这些是有关解析器配置文件 */etc/resolv.conf* 的语句。

domain

功能：

定义你的解析器的本地域名

语法：

```
domain domain-name
```

例子：

```
domain corp.hp.com
```

讲述章节：第六章

search

功能：

定义你的解析器的本地域名和搜索列表

语法：

```
search local-domain-name next-domain-name-in-search-list
... last-domain-name-in-search-list
```

例子：

```
search corp.hp.com pa.itc.hp.com hp.com
```

讲述章节：第六章

nameserver

功能：

告诉你的解析器查询特定的服务器

语法：

```
nameserver IP-address
```

例子：

```
nameserver 15.255.152.4
```

讲述章节：第六章

; 和 # (4.9+)

功能：

给解析器的配置文件加注释

语法：

; free-format-comment

或

 # free-format-comment

例子：

```
# Added parent domain to search list for compatibility with 4.8.3
```

讲述章节：第六章

sortlist (4.9+)

功能：

指定你的解析器优先使用的网络

语法：

sortlist *network-list*

例子：

```
sortlist 128.32.4.0/255.255.255.0 15.0.0.0
```

讲述章节：第六章

options ndots (4.9+)

功能：

指定一个参数中必须有的圆点的数量,以便解析器在应用搜索列表之前查找它

语法：

options ndots:*number-of-dots*

例子：

```
options ndots:1
```

讲述章节：第六章

options debug (4.9+)

功能：

打开解析器的调试输出

语法：

```
options debug
```

例子：

```
options debug
```

讲述章节：第六章

options no-check-names(8.2+)

功能：

在解析器中关闭名字检查选项

语法：

```
options no-check-names
```

例子：

```
options no-check-names
```

讲述章节：第六章

options attemps（8.2+）

功能：

指定解析器应该查询每个名字服务器的次数

语法：

```
options attempts:number-of-attempts
```

例子：

```
options attempts:2
```

讲述章节：第六章

options timeout(8.2+)

功能：

指定解析器的每个名字服务器的超时时间

语法：

```
options timeout:timeout-in-seconds
```

例子:

```
options timeout:1
```

讲述章节：第六章

options rotate(8.2+)

功能：

轮转解析器查询名字服务器的顺序

语法：

```
options rotate
```

例子：

```
options rotate
```

讲述章节：第六章

词汇表

authority
　　权威

caching-only
　　只缓存

checksum
　　校验和

CNAME（canonical name）
　　规范名

domain name space
　　域名空间

forward mapping
　　正向映射

forwarder
　　转发器

forward-only
　　只转发

header
　　首部

inverse query
　　逆向查询

IXFR（Incremental Zone Transfer）
　　增量区传送

loopback address
　　回送地址

mail hub
　　邮件分发器

mailer
　　邮件收发器

MD（mail destination）
　　邮件目的地

MF（mail forwarder）
　　邮件转发器

MX（mail exchanger）
　　邮件交换器

name server
　　名字服务器

name-to-address mapping
　　名字 - 地址映射

negative caching
　　否定缓存

NIS（Network Information Service）
网络信息服务

NLA（Next-Level Aggregator）
下级聚集器

origin
起点

overhead
开销

polling
轮询

resolver
解析器

retransmission
重传

reverse mapping
反向映射

root hint
根线索

round robin
循环反复

RP（responsible person）
负责人

RPC（Remote Procedure Call）
远程过程调用

RTT（roundtrip time）
往返时间

shuffle
混洗

SMTP（Simple Mail Transfer Protocol）
简单邮件传输协议

SOA（start of authority）
权威起始

subdomain
子域

TLA（Top-Level Aggregator）
顶级聚集器

transaction sigature
事务签名

TTL（time to live）
生存期

view
视图

WINS（Windows Internet Naming Service）
Windows Internet 命名服务