BIND9 Administrator Reference Manual

(Simplified Chinese Version)

BIND9 管理员参考手册

www.runstone.com 2003

目录

第一章：DNS 的起源和背景 1

第二章：DNS 的体系结构和原理 3

2.1 域名空间和体系结构 3
2.2 域和域名 3
2.3 区和域的不同 4
2.4 域名服务器的类型 5
2.4.1 主域名服务器 6
2.4.2 辅域名服务器（次级域名服务器） 6
2.4.3 隐藏服务器（stealth server） 6
2.4.4 高速缓存域名服务器（caching only server） 6
2.4.5 转发服务器（forwarding server） 6
2.5 DNS 基本原理 6

第三章：建立 DNS 系统服务 8

3.1 DNS 系统的来源 8
3.2 系统的要求 8
3.2.1 硬件需求 8
3.2.2 CPU 需求 8
3.2.3 内存需求 8
3.2.4 域名服务器的高配置问题 8
3.2.5 支持操作系统 9
3.2.6 获得 bind 9.2.3 9

第四章：域名服务器配置 10

4.1 配置实例 10
4.1.1 高速缓冲域名服务器（Caching-Only DNS） 10
4.1.2 授权的域名服务器（Authoritative-only DNS） 10
4.2 负载分担 11
4.3 通告（NOTIFY） 12
4.4 域名服务器的运行 12
4.4.1 域名服务器后台进程工具的使用 12
4.4.1.1 诊断工具 12
4.4.1.2 管理工具 13
4.4.2 信号（Signals） 16

第五章 高级理念 17

5.1 动态更新 17
5.1.1 日志文件（journal file） 17
5.2 增量域传输（IXFR） 17
5.3 拆分 DNS 18
5.4 TSIG（信号安全处理） 21
5.4.1 为每对主机产生共享密匙 21

5.4.1.1 自动产生.....21
5.4.1.2 手工生成.....22
5.4.2 把共享密匙拷到两台机器中.....*22*
5.4.3 通知服务器密匙的存在.....*22*
5.4.4 通知服务器使用密匙.....*22*
5.4.5 基于 TSIG 密匙的访问控制.....*23*
5.4.6 错误.....*23*
5.5 TKEY.....23
5.6 SIG(0).....24
5.7 DNSSEC.....24
5.7.1 产生密匙.....*24*
5.7.2 产生 keyset.....*25*
5.7.3 标志子域系列（Child's Keyset）.....*25*
5.7.4 对域进行标记.....*26*
5.7.5 配置服务器.....*26*
5.8 BIND9 的 IPv6 支持.....26
5.8.1 使用 AAAA 记录的地址查找.....*26*
5.8.2 使用 A6 记录查询地址.....*27*
5.8.2.1 A6 链（Chains）.....27
5.8.2.2 DNS 服务器的 A6 记录.....27
5.8.3 使用 Nibble 格式进行地址到名字的查询.....*28*
5.8.4 使用 bitstring 格式进行地址到名字的查询.....*28*
5.8.5 用 DNAME 来标示 IPV6 的反向地址.....*28*

第六章 BIND9 LIGHTWEIGHT 解析.....30

6.1 LIGHTWEIGHT 解析库.....30
6.2 运行后台解析.....30

第七章 BIND9 配置参考.....31

7.1 配置文件的组成元素.....31
7.1.1 地址匹配表.....*32*
7.1.1.1 语法.....32
7.1.1.2 定义和使用.....32
7.1.2 语法注释.....*33*
7.1.2.1 语法.....33
7.1.2.2 定义和用法.....33
7.2 配置文件语法.....33
7.2.1 acl 语句语法.....*34*
7.2.2 acl 语句定义和使用.....*34*
7.2.3 控制语句语法.....*35*
7.2.4 controls 语句定义和用法.....*35*
7.2.5 include 语句语法.....*35*
7.2.6 包含语句定义和使用.....*36*
7.2.7 键语句语法.....*36*
7.2.8 key 语句的定义和使用.....*36*

7.2.9 logging 语句语法.....36
7.2.10 Logging 语句定义和使用.....37
7.2.10.1 channel 短语.....37
7.2.10.2 category 短语.....39
7.2.11 lwres 语句语法.....40
7.2.12 lwres 语句定义和用法.....41
7.2.13 options 语句语法.....41
7.2.14 options 语句定义和用法.....43
7.2.14.1 Boolean 选项.....44
7.2.14.2 转发.....47
7.2.14.3 访问控制.....48
7.2.14.4 接口.....49
7.2.14.5 查询地址.....49
7.2.14.6 域传输.....50
7.2.14.7 操作系统资源限制.....51
7.2.14.8 服务器资源限制.....52
7.2.14.9 周期性任务间隔.....52
7.2.14.10 拓扑.....53
7.2.14.11 sortlist 语句.....54
7.2.14.12 RRset 排序.....55
7.2.14.13 合成的 IPV6 响应.....56
7.2.14.14 调谐.....56
7.2.14.15 统计文件.....57
7.2.15 服务器语句语法.....57
7.2.16 服务器语句定义和使用.....58
7.2.17 trusted-keys 语句语法.....59
7.2.18 trusted-keys 语句定义和使用.....59
7.2.19 视图语句语法.....59
7.2.20 视图语句定义和使用.....59
7.2.21 zone 语句语法.....60
7.2.22 zone 语句定义和使用.....61
7.2.22.1 域文件类型.....61
7.2.22.2 类.....63
7.2.22.3 zone 选项.....63
7.2.22.4 动态更新政策.....65
7.3 域文件.....66
7.3.1 资源记录类型及使用.....66
7.3.1.1 资源记录.....66
7.3.1.2 RRs 的原文表达.....68
7.3.2 MX 记录的讨论.....69
7.3.3 设置 TTLs.....69
7.3.4 IPV4 的反向解析.....70
7.3.5 其他的域文件指令.....70
7.3.5.1 $ORIGIN 指令.....70

7.3.5.2 $INCLUDE 指令 71
7.3.5.3 $TTL 指令 71

第八章、BIND9 的安全性 73

8.1 访问控制列表 73
8.2 CHROOT 和 SETUID（对与 UNIX 服务器） 73
8.2.1 chroot 环境 74
8.2.2 使用 setuid 函数 74
8.3 动态更新安全性 74

第九章、疑难解答 75

9.1 一般性问题 75
9.1.1 BIND 不工作了，如何才能找出问题的根源？ 75
9.2 增加和改变序列号 75
9.3 从哪里可以获得帮助 75

附录 A:附录 76

A.1 致谢 76
A.1.1 DNS 与 BIND 的历史 76
A.2. 历史的 DNS 信息 76
A.2.1. 记录资源类 77
A.2.1.1. HS = hesiod 77
A.2.1.2. CH = chaos 77
A.3. 一般的 DNS 参考信息 77
A.3.1 IPv6 地址 (A6) 77
A.4. 参考和建议阅读文献 78
A.4.1. 注释的要求 (RFCs) 78
A.4.2. 互联网草案 80
A.4.3. 关于 BIND 的其他文件 80

第一章：DNS 的起源和背景

前 言

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 ARPA，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够共享昂贵或稀缺的计算资源。

TCP/IP（传输控制协议/Internet 协议）是在 80 年代初发展起来的，并迅速成为 ARPAnet 的标准主机网络协议，这个网络也由原来的屈指可数的几台主机发展到数以千计的主机。原来的 ARPAnet 成为基于 TCP/IP 协议的局域网和区域联合网的主干，被称为 Internet.

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络 NSFNET 取代成为 Internet 的骨干网。在 1995 年春，Internet 完成了由公共 NSFNET 作为骨干网到使用多个商业骨干网的转变，这网络由 MCI、Sprint 长途电信运营商和就负盛名的商业网 PSINet、UUNet 共同组成。

DNS 的历史

到 20 世纪 70 年代末，ARPAnet 是一个拥有几百台主机的很小很友好的网络。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：它包含了所有连接到 ARPAnet 的主机名字到地址的映射（name-to-address mapping）。

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。ARPAnet 的管理员通常是通过电子邮件通知 NIC，同时定期 FTP 到 SRI-NIC 上获得最新的 HOSTS.TXT 文件。

但是随着 ARPAnet 的增长，这种方法行不通了。每台主机的变更都会导致 HOSTS.TXT 的变化，导致所有主机需要到 SRI-NIC 上获得更新文件。当 ARPAnet 采用 TCP/IP 协议后，网络上的主机爆炸性的增长，出现了下面的问题：

流量和负载

由于分发文件所引起的网络流量和分发主机的负载使得 SRI-NIC 的线路不堪重负。

名字冲突

HOSTS.TXT 文件必须要保持里面主机名字的唯一性，但是无法限制网

络上的主机用了相同的名字，这就破坏了网络上的正常应用服务。

一致性

在不断扩张的网络上维持 HOSTS.TXT 文件的一致性变得越来越困难。新的文件还没有到达 ARPAnet 的边缘时，另一端又添加了新的主机或是主机更改了地址。

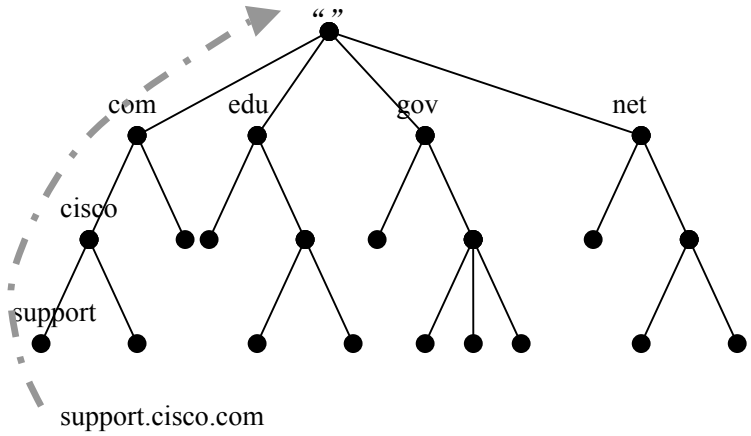
ARPAnet 的管理者们开始研究新的系统，以取代现有的 HOSTS.TXT 模式。1984 年，Paul Mockapetris 发布了 DNS 的管理规范。

DNS 简述

DNS（Internet Domain Name System）中包含了用来按照一种分层结构定义 Internet 上使用的主机名字的语法，还有名字的授权规则，以及为了定义名字和 IP 地址的对应，系统需要进行的所有设置。

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；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 将更强壮和足够的性能。

DNS 数据库的结构如图 1.1 所示，就象一棵倒挂着的树。

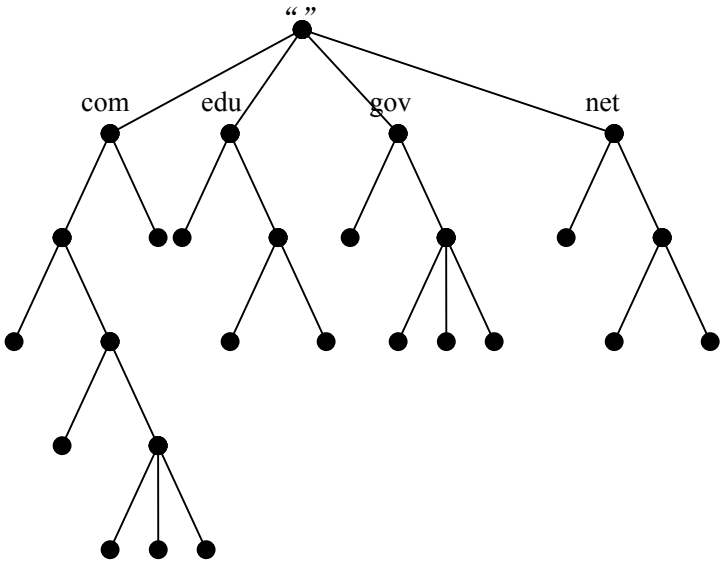


图：1.1

第二章：DNS 的体系结构和原理

2.1 域名空间和体系结构

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路径，这棵逆向树称为域名空间（domain name space）。如图 2.1 所示树的最大深度不得超过 127 层，树中每个节点都有一个可以长达 63 个字符的文本标号



图：2.1

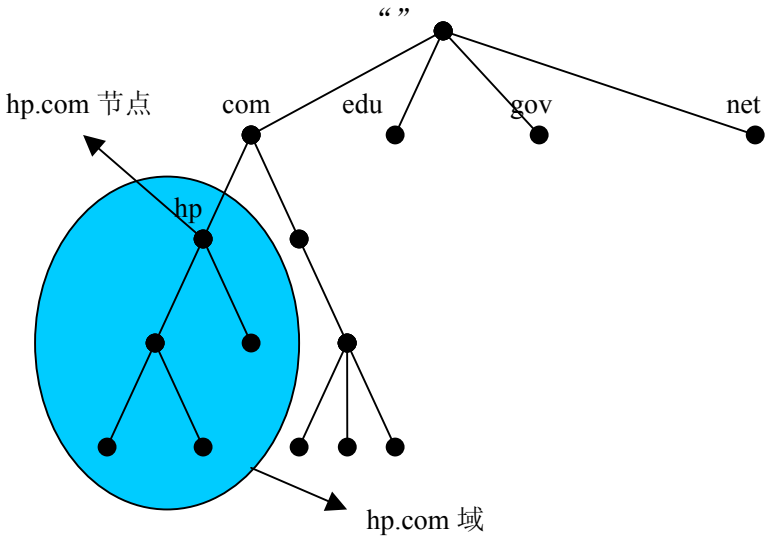
BIND 包含了一个叫 named 的后台进程，和 resolver 库。BIND 服务器程序在后台运行，通过众所周知的端口提供服务。UDP（User Datagram Protocol）和 TCP（Transmission Control Protocol）在 DNS 中的标准端口通常是 53，在/etc/services 中设置。Resolver 就是一系列驻留在系统库中的程序，用来提供出其它程序来访问 DNS 服务的端口。

2.2 域和域名

NS（nameserver）就是一个存储着域名资源信息的程序，它会响应来自叫 resolver 的程序的请求，resolver 类似于一个客户端程序。NS 的基本功能就是通过回答查询请求来提供网络信息。

利用 nameserver，整个网络可以划分成一个域的分层结构。整个的域名空间可以根据组织划分或管理分类，组织成一个树状结构。树上的每个节点叫做 domain，就是一个标示。一个 domain 的名字就是从 root 开始，到当前节点的所有 domain 标志的集合。从书写的结构上看，就是从右到左依次用"."来区分开。这样域名的标志才能够唯一。例如，Example,Inc. 公司一台主机的域名是：

ourhost.example.com
com 就是 ourhost.example.com 的顶级域，example 是 com 的子域，ourhost 是主机名。有关域名服务器的定义可以在 RFC1034、RFC1035 和 RFC974 中找到。



图：2.2

2.3 区和域的不同

整个的域名空间可以被分成多个区域，叫 zone。它开始于一个顶级 domain，一直到一个子 domain 或是其它 domain 的开始。zone 通常表示管理界限的划分。实际上，zone 就是 DNS 树状结构上的一个标识的点。一个 zone 包含了那些相邻的域名树结构的部分，并具有此部分的全部信息，并且它是真正授权的。它包含了这个节点下的所有域名，但不包括其它域里已经制定的。每个树状结构里的节点，在上级域中都有一个或多个 NS 记录。它们是和这个域中的 NS 记录相同的。

为了能正确地运行一台域名服务器，理解 zone 和 domain 的区别就非常重要。

例如，有一个 domain 叫 example.com，它可以包含 host.aaa.example.com 和 host.bbb.example.com 这些名字，但是它的 zone 文件中却只有 2 个 zone 的记录 aaa.example.com 和 bbb.example.com。zone 就是一个一级的 domain，也可以是一个多级 domain 的一部分。这个 domain 中的其它 zone，可以指向其它的域名服务器。DNS 树型结构中的每个名字都是一个 domain，当然它也可以是一个没有子域的、最末端的节点。每个子域（subdomain）也是一个 domain，每个 domain 其实也是一个子域（subdomain），当然最上面的 root 节点除外。详细的描述，可以参考 RFC1033、RFC1034、RFC1035。

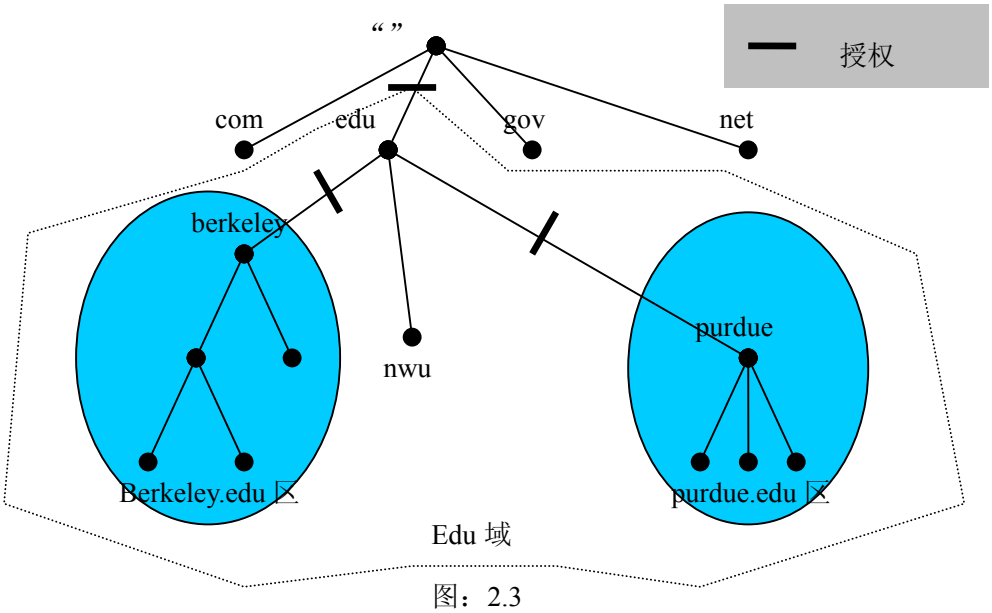
虽然 BIND 是 Domain Nameserver，但是它是按照 zone 来运行的。named.conf 文件中设置的 master 和 slave 就是指的一个 zone，而不是 domain。当用户希望另一台主机成为某个域（domain）的二级域名服务器，其实就是需要对这个 domain 中的所有 zone 进行二级备份服务。

每个 zone 都有一个 primary master server（或就叫 primary server），中文翻译为主域名

服务器，它会从手工编辑的本地文件中读取 zone 的全部内容。同时，zone 还可以有多个 slave server（或叫做 secondary server），中文翻译叫二级域名服务器，或辅域名服务器。它会使用 DNS 协议（辅域名服务器使用 TCP 和注域名服务器联系，并获得 zone 的数据）来读取 zone 的信息。所有这些服务器（包括 primary 和所有的 secondary）都应该列在上级域的 NS 记录中，这样才能形成一个正式的授权。同时，这些服务器也应该列在自己主机中的域文件中，通常是在@下面，@指的是当前域的顶级（不是整个域名树型结构的顶层）。用户可以列出本域的顶层(@)中 NS 记录里的所有服务器，尽管他们可能没有注册在上级域的 NS 记录中，但是不能列出虽然注册在上级域中，却没有出现在本地域的@中的服务器。

任何列在 NS 记录中的服务器就必须配置成那个域的授权域名服务器（authoritative server）。当用户查询这个域的信息时，这台授权的服务器就会提供授权的信息，也就是在返回包中设置了 AA 位（Authoritative Answer）。一台服务器可以是多个域的授权服务器。一个域的授权数据由所有的 RR（Resource Records）组成。

当将一个域配置为 master 或 slave 时，就是让服务器返回有关这个域的授权信息。如果服务器可以正确地将这个域调入内存中，则在回答有关这个域的查询请求时将会设置 AA 位。AA 位的信息在 RFC1034 和 RFC1035 中有详细描述。



图：2.3

2.4 域名服务器的类型

一个 DNS 服务器可以同时作为多个域的主域名服务器和辅域名服务器，也可以只作为主，或只作为辅，或者做任何域的授权服务器而只使用自己的 cache 来提供查询解析。Master 服务器也经常叫做 primary，slave 服务器也经常叫做 secondary。不论是 master/primary/主/一级域名服务器，还是 slave/secondary/辅/二级域名服务器，都是这个域的授权服务器。

所有的服务器都会将数据保存在缓存（cache）中，直到针对这些数据的 TTL（Time To Live）值过期。

2.4.1 主域名服务器

primary master server 是一个 domain 信息的最根本的来源。它是所有辅域名服务器进行域传输的源。主域名服务器是从本地硬盘文件中读取域的数据的。

2.4.2 辅域名服务器（次级域名服务器）

就是 slave server，或叫作 secondary server。次级服务器使用一个叫做域转输的复制过程，调入其它服务器中域的内容。通常情况下，数据是直接从主服务器上传输过来的，但也可能是从本地磁盘上的 cache 中读到的。辅域名服务器可以提供必需的冗余服务。所有的辅域名服务器都应该写在这个域的 NS 记录中。

2.4.3 隐藏服务器（stealth server）

stealth server 可以针对一个域的查询返回授权的记录，但是它并没有列在这个域的 NS 记录里。Stealth 服务器可以用来针对一个域进行集中分发，这样可以不用在远程服务器上手工编辑这个域的信息了。在这种方式中，一个域的 master 文件在 stealth server 上存储的位置，经常叫做“hidden primary”配置。Stealth 服务器也可以将域文件在本地做一个拷贝，从而可以在所有官方的域名服务器都不能访问的情况下，也能更快地读取域的记录。

2.4.4 高速缓存域名服务器（caching only server）

缓存服务器可以将它收到的信息存储下来，并再将其提供给其它的用户进行查询，直到这些信息过期。它的配置中没有任何本地的授权域的配置信息。它可以响应用户的请求，并询问其它授权的域名服务器，从而得到回答用户请求的信息。

2.4.5 转发服务器（forwarding server）

一台缓存名服务器本身不能进行完全的递归查询。相反，它能从缓存向其它的缓存服务器转发一部分或是所有不能满足的查询，一般被称作转发服务器。

可能会有一个或多个转发服务器，它们会按照顺序进行请求，直到全部穷尽或者请求得到回答为止。转发服务器一般用于用户不希望站点内的服务器直接和外部服务器通讯的情况下。一个特定的情形是许多 DNS 服务器和一个网络防火墙。服务器不能透过防火墙传送信息，它就会转发给可以传送信息的服务器，那台服务器就会代表内部服务器询问因特网 DNS 服务器。使用转发功能的另一个好处是中心服务器得到了所有用户都可以利用的更加完全的信息缓冲。

2.5 DNS 基本原理

域名系统是分级的分布式的数据库。它储存对应于 IP 地址与主机名相对应的信息，邮件路由信息和其他网络应用方面的数据。

用户通过询问解决库（解决库发送询问并对回应进行说明）在 DNS 上查询信息。BIND9 软件分级同时包括了一个域名服务器和一个解决库。

第三章：建立 DNS 系统服务

3.1 DNS 系统的来源

参考下面的网址，获得最新版本的 bind 或了解当前运行 bind 的漏洞和隐患。
http://www.isc.org
ftp://ftp.isc.org

3.2 系统的要求

3.2.1 硬件需求

DNS 硬件要求传统上是比较适度的。对许多配置来说，一些“退休”的服务器也能很好的充当 DNS 服务器。但是 BIND9 的 DNSSEC 和 IPV6 功能对 CPU 要求很高，所以那些频繁应用这些功能的机构应该考虑为这个软件配备大一些的系统。BIND9 现在是完全多线程的，允许完全的多程序系统的利用。

3.2.2 CPU 需求

BIND9 的 CPU 需求从只能提供没有 cache 的静态域的 486 到企业级的计算机，企业级计算机可以处理许多动态更新和 DNSSEC 加密域，每秒处理数千个查询。

3.2.3 内存需求

服务器内存要足够大以适合从磁盘读出的缓存和域。max-cache-size 选项可用来限制缓存占用的内存，代价是降低缓存命中率和导致更多的 DNS 流量。拥有足够的内存以上载所有的域和缓存数据到内存中是正确的吗？不幸的是，在一个给定配置下判定的方法是观察运行中的域名服务器。在几周后，服务器进程会达到相对稳定的容量，那个时候新加入到缓存中的记录和缓存中过期后被删除的记录数量基本平衡。比较理想的是，资源限制应该设定的比稳定容量高一点。

3.2.4 域名服务器的高配置问题

对于域名服务器的配置，有两种备选方案。第一是用户和所有的 2 级内部域名服务器查询一台主域名服务器，主服务器有足够的内存以建立一个巨大的缓存。这种方法最小化了用户大量的对外部名查询使用的带宽。第二种备选方案是建立二级内部域名服务器来进

行独立的查询请求。在这种配置下，没有一台独立的主机需要象第一种配置一样有足够的内存或者 CPU 功率，但是因为没有名服务器共享它们的缓存数据，这样会导致许多外部查询。

3.2.5 支持操作系统

ISC BIND 9 在下列操作系统下编译和运行:
IBM AIX 4.3
Compaq Digital/Tru64 UNIX 4.0D
Compaq Digital/Tru64 UNIX 5 (with IPv6 EAK)
HP HP-UX 11
IRIX64 6.5
Sun Solaris 2.6, 7, 8 ,9
NetBSD 1.5 (with unproven-pthreads 0.17)
FreeBSD 3.4-STABLE, 3.5, 4.0, 4.1
Red Hat Linux 6.0, 6.1, 6.2, 7.0

3.2.6 获得 bind 9.2.3

下载地址 (FTP):
ftp://ftp.isc.org/isc/bind9/9.2.3/bind-9.2.3.tar.gz

Bind 软件包提供了 DNS 服务器和客户端工具。

第四章：域名服务器配置

在这部分，我们提供建议配置和他们的使用指南。我们也介绍主要的选项设定。

4. 1 配置实例

4.1.1 高速缓冲域名服务器（Caching-Only DNS）

下列建议配置适用于缓冲域名服务器，它只能为内部用户使用，任何外部用户的请求都会被拒绝。

```
// 我们希望允许来自两个公司子网的 DNS 请求
acl "corpnets" { 192.168.4.0/24; 192.168.7.0/24; };
options {
directory "/etc/namedb"; //工作目录
pid-file "named.pid";     //把 DNS 进程文件加到工作目录中
allow-query { "corpnets"; };
};

// 根服务器配置
zone "." { type hint; file "root.hint"; };

// 提供针对服务器自己的 loopback 地址 127.0.0.1 的反向解析
zone "0.0.127.in-addr.arpa" {
        type master;
        file "localhost.rev";
        notify no;
};
```

4.1.2 授权的域名服务器（Authoritative-only DNS）

在这个例子中，"example.com"是授权的主域名纪录，"eng.example.com"是辅域名纪录。

```
options {
        directory "/etc/namedb";        // 工作目录
        pid-file "named.pid";           //把 DNS 进程文件加到工作目录中
        allow-query { any; };           // 默认配置
        recursion no;                   // 不提供递归服务
```

```
};

//根服务器配置
zone "." { type hint; file "root.hint"; };

// 提供针对服务器自己的 loopback 地址 127.0.0.1 的反向解析
zone "0.0.127.in-addr.arpa" {
        type master;
        file "localhost.rev";
        notify no;
};

// 本机是“example.com”域的主 DNS 服务器
zone "example.com" {
        type master;
        file “example.com.db”;
        //容许进行域传送的 example.com 的辅域名服务器地址
        allow-transfer {
                192.168.4.14;
                192.168.5.53;
        };
};

// 本机是 eng.example.com 的辅域名服务器
zone "eng.example.com" {
        type slave;
        file “eng.example.com.bk”;
        //eng.example.com 的主域名服务器地址
        masters {192.168.4.12; };
};
```

4.2 负载分担

原始的负载分担是在 DNS 中，通过对一个名字使用多条的 A 记录来实现的。例如，如果用户有三个 WWW 服务器，地址是 10.0.0.1, 10.0.0.2 和 10.0.0.3，下列一系列记录表示用户会有机会访问这 3 个 IP 地址，且几率相同。

Name	TTL	CLASS	TYPE	Resource Record (RR) Data
www	600	IN	A	10.0.0.1
	600	IN	A	10.0.0.2
	600	IN	A	10.0.0.3

为了解析 www 请求响应所有这些记录，BIND 会轮询这些记录，以不同的顺序为结果

来响应用户的解析请求。在上面的例子中，用户将会随机以 1，2，3；2，3，1 和 3，1，2 的顺序得到解析结果。大多数用户会使用返回的第一条纪录而忽略掉其余的纪录。
有关响应顺序的细节，可以查看 options 中有关 rrset-order 的配置。但是 BIND9 中不支持 rrset-order 中的配置，只能按照上述的顺序响应。

4.3 通告（Notify）

DNS 通告是一种机制，它允许主域名服务器将某个域的变化通知它的辅域名服务器。辅域名服务器会响应来自主域名服务器的 NOTIFY 请求，检查自己这个域的配置文件版本，如果和主域名服务器中的不同，则就会进行同步。

DNS 通告在 RFC1996 有详细的介绍。可以参考 zone option 中有关 also-notify 的描述。

4.4 域名服务器的运行

4.4.1 域名服务器后台进程工具的使用

系统管理者可以利用一些不可或缺的诊断，管理和检测工具控制和调试名服务器的后台程序。

4.4.1.1 诊断工具

dig(Domain Information Groper)

域信息搜索器（dig）是一种用于从域名系统服务器那里收集信息的命令行工具。搜索有两种模式：针对一条请求的简单交互式模式和执行多查询的批模式。所有的查询选项都可以通过命令行上来完成。

dig [@服务器]域
[查询-类型] [查询-类 [+查询-选项] [-dig-选项] [%注释]

通常采用以下形式：
dig @服务器 域 查询类型 查询类 （dig @server domain query-type query-class）
更多信息和可利用的命令和选项，参看 dig 的 man 帮助信息。

host

Host 功能利用一个命令行界面，提供查询因特网主机名的功能。在默认状态下，他只将对主机名和 IP 地址进行转换，但是此项功能可以通过选项的使用得到拓展。

host [-aCdlrTwv] [-c class] [-N ndots] [-t type] [-W timeout] [-R retries] hostname [server]

更多信息和可利用的命令和选项，参看 dig 的 man 帮助信息。

Nslookup

是一个向因特网域名服务器发送请求的程序。Nslookup 有两种模式：交互式和非交互式。交互式允许向名服务器查询多个主机和域或者打印出在一个域中的主机列表。非交互式模式用来显示一个主机或域的名字或被请求的信息。

nslookup [-option…][host-to -find]-[server]

当不使用参数（使用缺省的域名服务器），或第一个参数是”-”且第二个参数是一台 DNS 的主机名或 IP 地址话，nslookup 就会进入交互模式。

非交互模式是在第一个参数为要查询的域名或 IP 地址时进入的。也可以配置第二个参数，来选定请求发送的域名服务器。由于神秘的用户界面和经常性的不一致行为，我们不建议使用 nsloopup.而应使用 dig.

4.4.1.2 管理工具

管理工具在服务器管理中起整体性作用

named-checkconf。

named-checkconf 检查 named.conf 文件的句法

named-checkconf [-t directory] [filename]

named-checkzone

named-checkzone 程序检查 host 文件的句法和相容性。

named-checkzone [-dq] [-c class] zone [filename]

rndc(Remote Name Daemon Control)

rndc 允许系统管理员控制名称管理器的运行。

如果不使用任何选项来运行 rndc 将会有一条形如下列的使用信息：

rndc [-c config] [-s server] [-p port] [-y key] command [command...]

其中 command 是下列中的一种:

reload

重新加载配置文件和域（zone）的配置。

reload *zone [class [view]]*

重新加载一个指定的域。

refresh *zone [class [view]]*

定期维护一个指定的域。

reconfig

重新加载 namd.conf 配置文件和新的域，但不会重新加载已存的域文件，即使域文件已经被修改了也不会加载。特别是当有大量的域的时候，这比全部的 reload 要快很多，因为这避免了对域文件修改时间的检查。

stats

将统计信息写入到统计文件中。

querylog

启动用户请求的日志纪录。请求的日志记路也可以通过在 named.conf 中 logging 部分的 queries category 和 channel 来启动。

dumpdb

把服务器缓存中的信息转储到 dump 文件中去。

stop

服务器停机，且将所有最近通过动态更新或 IXFRS 作出的修改，都首先存到了更新域中的 host 文件中。

halt

服务器马上停机。但通过动态更新和 IXFR 做出的最新修改不会存入 host 文件，但当服务器重新启动的时候会写入到日志文件中去。

trace

增加一个服务器的 debug 级别。

trace level

直接设置服务器的 debug 等级。

notrace

将服务器的 debug 等级设成 0。

flush

清除域名服务器缓存中的内容。

status

显示服务器状况

在 BIND 9.2 中, 除了 ndc start 外，rndc 支持 BIND 8 ndc 中的所有命令, 当然，bind8 中 ndc 的通道模式也不支持 bind9 中的其它命令。

这里需要一个配置文件，因为所有和服务器的通信都是利用基于共享加密的数字签名

技术来完成的，而除了提供配置文件就没有其它加密的方法了。rndc 默认的配置文件位置是在/etc/rndc.conf，但是可以通过-c 选项指定改变的位置。如果配置文件没有找到，rndc 也会访问 etc/rndc.key（或者是在 BIND 编译时对 sysconfdir 定义的值）。rndc.key 文件是当 **rndc-confgen -a** 运行的时候产生的。

配置文件的格式类似 named.conf，但被限制在四个部分中：options、key、server 和 include。这些语句把共享密匙和服务器相联系。语句的顺序是不重要的。

option 语句有三个子语句：default-server、default-key 和 default-port。

default-server 采用主机名或 ip 地址，如果在命令行中没有-s 选项的话，代表了将会被连接的服务器。default-key 把 key 中定义的密匙名字作为参数。如果在命令行或者 server 语句中没有指定端口的话，Default-port 则指定了 rndc 和 server 应该连接的缺省端口。

key 语句用名字命名了一串字符串。服务器需要这个字符串来认证。Key 语句中包含 2 个字句：algorithm 和 secret。配置分析器可以任何字符串，做为算法(algorithm)的参数，目前只有字符串”hmac-md5”有含义。secret 是一个基于 64 位编码的字符串。

server 语句使用 key 的子句来连接 key 和服务器。server 语句的参数是一个服务器名或 IP 地址（地址必须用双引号）。Key 子句中的参数是 key 语句中定义好的 key 的名字。port 子句可以用来指定 rndc 和服务器联接的端口。）

最小的配置语句举例如下：

```
key rndc_key {
    algorithm "hmac-md5";
    secret
"c3Ryb25nIGVub3VnaCBmb3IgYSBtYW4gYnV0IG1hZGUgZm9yIGEgd29tYW4K";
};
options {
    default-server localhost;
    default-key rndc_key;
};
```

这个文件，如果安装在/etc/rndc.conf，则可以执行命令：

```
$ rndc reload
```

连接到 127.0.0.1 的 953 端口，并重启域名服务器，如果一个域名服务器在本地计算机下运行下列控制语句：

```
controls {
    inet 127.0.0.1 allow { localhost; } keys { rndc_key; };
};
```

并且它有一个确定的 key 语句给 rndc_key。

运行 rndc-confgen 程序将会很顺利的产生一个 rndc.conf，且可以显示出应该配置在 named.conf 中 controls 语句里的相关内容。你可以运行 rndc-confgen –a 来创建 rndc.key 文件，且并不修改 named.conf。

4.4.2 信号（Signals）

特定的 unix 信号使域名服务器进行不同的操作，就象下表叙述的一样。这些信号可以通过使用 kill 命令发出：

SIGHUP	使服务器重读 name.conf 文件，并重新加载数据库。
SIGTERM	使服务器清空和退出。
SIGINT	使服务器清空和退出。

第五章 高级理念

5.1 动态更新

动态更新一个术语，某些特定情况下，在住域名服务器的 host 文件中增加、修改和删除记录或 RRset。动态更新在 RFC2136 中有完整的描述。

动态更新是在域和域之间实现的，通过域中的 allow-update 或 update-policy 子句完成的。针对安全域（使用 DNSSEC）的更新，可以参考 RFC3007。

安全域的更新（使用 DNSSEC 的域）参见：受更新影响的 SIG 和 NXT 记录可以通过使用一个 online zone key 的服务器自动生成。更新的授权是基于传送签名和对更新服务器策略的直接定义。

5.1.1 日志文件（journal file）

一个域的所有动态更新都会纪录在域的日志文件中。这个文件是在第一次动态更新发生时由服务器自动生成的。日志文件名是通过在相应文件名后添加扩展名.jnl 形成的。日志文件是二进制的，不能手工编辑。

服务器有时会把更新域的所有内容写（“dump”）到它自己的域文件（hosts 文件）中去。这并不是在每个动态更新后马上完成的，因为当一个大的域频繁更新的时候那将太慢。相反，dump 会被延迟 15 分钟，以便使所有其它的更新完成。

当一台服务器在关闭或崩溃后重新启动，它将依靠日志文件把任何在上次 dump 之后发生的更新重新修改到域文件中来。

通过增量域传输（IXFR）的更新也被以一种类似的方式被日志化。动态域的 hosts 文件不能正常的被手工编辑，因为这样就不能保证所有最新的更新都纪录在日志文件中。唯一确保一个动态域文件是最新的方法是运行 rndc stop。

如果用户必须手工改动了一个动态域文件，请参考下面的步骤：使用 rndc stop 命令关闭服务器（用 kill 或者使用 rndc halt 是不行的）。等服务器退出后，删掉这个域的 hosts.jnl 文件，编辑 hosts 文件，再重新启动 DNS。消除.jnl 文件是必要的，因为手工的编辑不会在日志中表现出来，以使之与域文件的内容保持一致。

5.2 增量域传输（IXFR）

增量域传输协议针对辅域名服务器，仅仅从主域名服务器上的 hosts 文件中传送变化的内容，而不用传送整个域文件。IXFR 协议在 RFC1995 中有明文规定。参见推荐标准。

当作为主 DNS 时，BIND9 针对需要历史信息的域使用 IXFR。这些包括由动态更新维护的主 DNS 的域和通过 IXFR 获取数据的辅域，而不包括那些靠手工维护的主域或靠运行完全域传输（AXFR）得到的次辅域。

当作为辅 DNS 时，除非配置中完全禁止，BIND9 都会使用 IXFR。要得到更多关于 IXFR 的信息，参看 server 语句中关于 request-ixfr 的内容。

5.3 拆分 DNS

为针对内部和外部不同的解析，可以通过设置不同的视图（view）或可视性，将 DNS 分成内外 2 个空间。对于一个组织来说，以这种方式来设置 DNS 有多种原因。

一个主要原因是对因特网外部用户隐藏内部 DNS 信息。关于这是否有效存在一些争论。内部 DNS 信息通过许多方式（如通过电子信头）泄漏出去，大多数聪明的“攻击者”可以通过其他途径找到他们需要的信息。

另一个设置 DNS 拆分系统的原因是：允许在过滤器（filter）后或者在 RFC1918 地址段（保留 IP 地址段，RFC1918 中有详细说明）的内部网络，来解析 internet 上的 DNS。

这有一个关于拆分 DNS 设置的一个例子：

一个叫 EXAMPLE 的公司（example.com）拥有若干公司站点，这些站点拥有一个使用保留 IP 地址的内部网和一个外部用户可以使用的域(DMZ)，或者叫可以公共访问的“外部”网络。

Example 公司希望内部用户能够查出外部主机名，并且与外网交换邮件。公司也希望内部用户能够访问一些只对内部开放而不对外网开放域。

为了达到这个目的，公司设置了两组域名服务器。一组在内部网（在保留 IP 地址内），另一组在 proxy 主机上，它可以在外网上（DMZ）与内外双方通信。内部 DNS 服务器设置成转发所有请求，除了对在 DMZ 中的 site1.internal、site2.internal、site1.example.com 和 site2.example.com 的请求。这些内部服务器将会具有 site1.example.com, site2.example.com, site1.internal, 和 site2.internal 这些域的完整信息。

为保护 site.internal 和 site2.internal 域，内部域名服务器应设置为不允许所有外部主机对这些域进行访问，包括 proxy 主机。

在 proxy 主机上的外部 DNS 服务器设置成具有 site1 和 site2.example.com 域的“公开”版本。这包括公开服务器的主机记录（www.example.com 和 ftp.example.com），邮件交换（MX）记录（a.mx.example.com 和 b.mx.example.com）。

此外，公开的 site1 和 site2.example.com 域应该具有包括通配符（“*”）的特殊 MX 记录，指向 proxy 主机。这是必要的，因为外部邮件服务器没有其他的办法了解如何发送邮件到这些内部主机上。有了通配符记录，邮件可以发送给 proxy 主机，然后就可以转发给内部主机。

下面是一个带通配符 MX 记录的例子：

```
*    IN   MX 10   external1.example.com.
```

现在 proxy 主机接收所有去往内部网的邮件，它们需要知道如何将邮件发送到内部主机上。为了让此项运转更加顺利，proxy 服务器上的解析将指向内部的 DNS。

来自内部主机的查询请求将会由内部服务器回答，而来自外部主机的查询请求将会转回到 proxy 主机上的 DNS 服务器。

为了使所有操作顺利进行，内部用户需要设置成只通过内部 DNS 进行查询。这也

可以通过网络设备上的配置来强制执行。如果所有都设置好了，example 公司的内部用户就可以：

- 查询任何在 site1 和 site2.example.com 域的主机。
- 查询任何处于 site1.internal 和 site2.internal 域的主机。
- 查询任何在因特网上的主机。
- 与内部和外部交换邮件。

在因特网上的主机将能够：

- 查询任何在 site1 和 site2.example.com 域的主机
- 与任何在 site1 和 site2.example.com 域的主机交换邮件。

以下是一则如上所述的配置。注意到这只是配置信息；要了解如何配置你的域文件，参见其它章节。

内部 DNS 服务器配置：

```
acl internals { 172.16.72.0/24; 192.168.1.0/24; };
acl externals { proxy 主机 ip 地址; };
options {
        ...
        ...
        forward only;
        forwarders {                            //转发到外部服务器
                bastion-ips-go-here;
        };
        allow-transfer { none; };
        allow-query { internals; externals; };  //限制请求
        allow-recursion { internals; };         //限制第归请求
        ...
        ...
};
zone "site1.example.com" {                      //主域
        type master;
        file "m/site1.example.com";
        forwarders { };                         //不做转发
        allow-query { internals; externals; };
        allow-transfer { internals; };
};

zone "site2.example.com" {
        type slave;
        file "s/site2.example.com";
        masters { 172.16.72.3; };
```

```
        forwarders { };
        allow-query { internals; externals; };
        allow-transfer { internals; };
};

zone "site1.internal" {
        type master;
        file "m/site1.internal";
        forwarders { };
        allow-query { internals; };
        allow-transfer { internals; }
};

zone "site2.internal" {
        type slave;
        file "s/site2.internal";
        masters { 172.16.72.3; };
        forwarders { };
        allow-query { internals };
        allow-transfer { internals; }
};
```

外部（proxy 主机）DNS 服务器设置：

```
acl internals { 172.16.72.0/24; 192.168.1.0/24; };
acl externals { proxy 主机 ip 地址; };
options {
        ...
        ...
        allow-transfer { none; };                       //不允许域传送
        allow-query { internals; externals; };          //限制请求
        allow-recursion { internals; externals; };          //限制第归查询
        ...
        ...
};

zone "site1.example.com" {
        type master;
        file "m/site1.foo.com";
        allow-query { any; };
        allow-transfer { internals; externals; };
};

zone "site2.example.com" {                      //辅域
```

```
        type slave;
        file "s/site2.foo.com";
        masters { 可能是另一台 proxy 主机地址; };
        allow-query { any; };
        allow-transfer { internals; externals; }
};
```

在 proxy 服务器中的 resolv.conf 文件（或等价的）

```
search …
nameserver 172.16.72.2
nameserver 172.16.72.3
nameserver 172.16.72.4
```

5.4 TSIG（信号安全处理）

这是一个基于 BIND 中的安全处理的 Transaction SIGnature (TSIG)。它描述了配置文件的更新和在不同情况下的更新要求，包括产生处理密匙和使用 BIND TSIG 的过程。

BIND 主要支持服务器对服务器之间通讯的 TSIG。包括域传送（zone transfer），通报（notify）和递归查询信息。基于 BIND8 的新版本对 TSIG 的支持较为有限。

TSIG 可能对动态更新最有用了，一个动态域的主 DNS 服务器使用访问控制来控制更新，而基于 IP 的访问控制是不够的。基于密匙的访问控制要高级的多了，参看推荐标准。nsupdate 程序通过-k 和-y 命令选项支持 TSIG。

5.4.1 为每对主机产生共享密匙

产生一个共享的加密方式就是在 host1 和 host2 之间共享使用。可选择任意的密匙：“host1-host2”。但密匙必须在两个主机上是一样的。

5.4.1.1 自动产生

下列命令将会产生一个如上所述 128 位（16 字节）HAMC-MD5 的密匙。越长的键越好，但是较短的键比较容易读取。注意键的最大长度是 512 比特；更长的键将会被 MD5 消化以产生 128 位的密匙。

```
dnssec-keygen -a hmac-md5 -b 128 -n HOST host1-host2.
```

密匙存在于 Khost1-host2.+157+00000.private 文件中。文件不直接被调用，但是在”Key:”之后的 base-64 编码字符串可以直接拷贝出作为共享密匙：

Key: La/E5CjG9O+os1jq0a2jdA==

字符串"La/E5CjG9O+os1jq0a2jdA=="可以作为共享密匙使用

5.4.1.2 手工生成

共享密匙仅仅是使用 base-64 编码的随机序列结果。大多数 ASCII 字符串是有效的 base-64 字符串（假设长度是 4 的倍数，只有有效的字符被使用），所以共享密匙可以被手工生成。

而且，一个熟知的字符串可以通过 mmencode 或者一个相似的程序以产生 base-64 编码数据。

5.4.2 把共享密匙拷到两台机器中

这超过了 DNS 的范围。使用一种安全传输机制，例如可以是安全 FTP、ssh、电话等。

5.4.3 通知服务器密匙的存在

设想 host1 和 host2 是这 2 台服务器。下列语句将会加到每个服务器中的 named.conf file 中：

```
key host1-host2. {
    algorithm hmac-md5;
    secret "La/E5CjG9O+os1jq0a2jdA==";
};
```

BIND 只支持 hmac-md5 算法。密匙就是在上面产生的这个。既然这是一个密匙，建议将 named.conf 设为不可读，或者在 named.conf 中调用一个包含了密匙的不可读的文件。

这样，key 就被认可了。这意味着如果服务器受到一则被这个 key 标记的消息，它可以对这个签字进行校验。如果校验成功，应答就会被同一个 key 所标记。

5.4.4 通知服务器使用密匙

既然密匙只在两个主机之间共享，服务器就必须被告知什么时候使用 key。下列是加入到 host1 的 named.conf 文件中的配置，如果 host2 的 IP 地址是 10.1.2.3:

```
server 10.1.2.3 {
```

```
    keys { host1-host2. ;};
};
```

多个 key 可能同时被使用，但是只有第一个有效。这个指示不包括任何加密，所以它可能是一个普遍可读文件。

如果 host1 向那个地址发送一个消息，此消息将会被特殊的 key 标记。host1 则会等待任何使用了相同 key 标记的回复信息。

一个相似的语句也会存在于 host2 的配置文件中（使用 host1 的地址），这样 host2 就会在回复 host1 的消息中标记相同的 key。

5.4.5 基于 TSIG 密匙的访问控制

BIND 承认在 ACL 定义中使用 IP 地址和地址段和 allow-{ query | transfer | update }。这也拓展到允许使用 TSIG 密匙。上述 key 可以表示为 key host1-host2。

一个 allow-update 的例子是：

```
allow-update { key host1-host2. ;};
```

它只允许那些带有”host1-host2”标记的动态更新请求被接受。后面的 update-policy 还有更加强大的功能。

5.4.6 错误

在处理用 TSIG 标记信息时会发生一些错误。如果一个标记信息被发送到一个不兼容 TSIG 的服务器中，服务器不能识别记录，就会返回一个 FORMERR。这是配置错误的结果，服务器应该配置清楚要发送到的特定的 server。

如果识别 TSIG 的服务器收到一则由未知 key 标志的信息，响应时就不会用 TSIG 标记，且会带有错误编码 BADKEY。如果一个识别 TSIG 服务器收到一个带着无效标记的信息，回应就不会用 TSIG 标记，且会带有错误编码 BADSIG。如果一台识别 TSIG 服务器接收到一个超过规定时限的信息，响应时就会带有 TSIG 标记的错误代码 BADTIME，且时间值将会被重新调整，使得响应可以被成功验证。在所有这些情况中，消息的错误代码都被设置为 NOTAUTH。

5.5 TKEY

TKEY 是自动在两台主机间产生共享密匙的一种机制。TKEY 有若干模式，来设定如何生成和分配一个 key。BIND 只执行这些模式中的一种，Diffie-Hellman 的 key 交换。两台主机都被要求有 Diffie-Hellman KEY 记录（尽管这个记录并不要求显示在域中）。TKEY 进程必须使用被 TSIG 或 SIG(0)标记的消息。TKEY 的结果是一个可以用于 TSIG 标记信息的共

享密匙。TKEY 也可以用来删除以前产生的共享密匙。

TKEY 进程是由一个用户或服务器，通过向识别 TKEY 的服务器发送带有的 TKEY 的请求（包括任何相应的密匙）来初始完成的。如果是正确的话，服务器的响应将会包括 TKEY 记录和任何相应的密匙。在交换之后，所有参与方都能够有足够的信息来确定共享密匙。确切的处理过成，要依赖于 TKEY 的模式。但是用 Diffie-Hellman TKEY 模式的时候，Diffie-Hellman 键被交换，双方都得到共享密匙。

5.6 SIG(0)

BIND9 部分支持有 RFC2535 指定的 DNSSEC SIG（0）处理加密。SIG（0）public/private 密匙来鉴别信息。访问控制以和 TSIG 相同方式执行；可以依靠密匙内容，来授予或拒绝各种权限。

当收到一个 SIG（0）标记的信息，只有当密匙被服务器知道，且信任的时候才会被确认，即服务器不用定位或核实密匙的内容。

SIG（0）不被支持多信息 TCP 流的识别。

BIND9 不提供任何产生 SIG（0）标记文件的工具。

5.7 DNSSEC

通过由 RFC2535 所定义的 DNS sercurity (DNSSEC)，DNS 信息的加密认证成为可能。本节介绍了用 DNSSEC 加密的域的产生和使用。为了建立 DNSSEC 安全域，必须严格遵循一系列的步骤。BIND9 提供了若干个可以使用的工具。所有工具使用“-h”选项，都可以得到完整的功能参数。注意，DNSSEC 工具要求 keyset 和 signedkey 文件在工作目录中，BIND9.0.X 所提供的工具和当前的不一致。

这也需要和上级域或下级域的管理者进行沟通，从而完成密匙和签名的传输通讯。一个域的安全状况应该由上级域的 DNSSEC 解析指向文件表明信任它的数据。

对于其他服务器对本域中数据的信任需求，则必须用这个域的密匙或上级域的密匙，进行静态的配置。

5.7.1 产生密匙

dnssec-keygen 用来产生密匙。

一个安全的域必须包括一个或多个密匙。域的密匙会对这个域中的所有其它的记录进行加密，识别所有在域中的其他记录，就象其它域的密匙一样。域的密匙必须和域有相同的名字，名字的类型是 ZONE，必须能够能用于 authentication。建议域的密匙应该强制使用加密算法来生成。目前 BIND9 只支持用于 DSA 算法的密匙。

下列命令将会为 child.example 域产生一个 768 位的 DSA 密匙：

```
dnssec-keygen -a DSA -b 768 -n ZONE child.example.
```

会生成两个文件：*Kchild.example.+003+12345.key* 和 *Kchild.example.+003+12345.private*（12345 是一个假设的密匙标签(key tag)）。密匙文件的名字包括密匙名（child.example.），算法（例如 3 是 DSA，1 是 RSA，等）和密匙标签（这里是 12345）。私人密匙（在.private 文件中）用来生成签名，公开密匙（在.key 文件中）用于签名的认证。

用相同的属性（应使用不通的密匙标签）产生其它的密匙，可以重复上述命令。

公开密匙用$INCLUDE 语句将.key 文件插入到域文件中。

5.7.2 产生 keyset

dnssec-makekeyset 程序用一个或多个密匙产生一个密匙系列（key set）。

一个域的密匙生成后，上级域的管理员就需要一个用于传输的密匙系列（key set），这样上级域的信息才能用自己的域密匙来标记，从而正确的表明这个域的安全状态。创建一个密匙系列（key set）时，需要指定要包含的密匙列表，设置 key set 的 TTL 值和上级域签名的有效期。

要被插入 key set 中的密匙列表，也可以包含出现在域的开始位置的非域的密匙（non-zone key）。dnssec-makekeyset 也可以使用域中的其它名字。

下列命令产生一个包含上面的密匙和 nother key similarly generated，TTL 为 3600，签名有效期为从现在开始的 10 天。

dnssec-makekeyset -t 3600 -e +86400 Kchild.example.+003+12345 Kchild.example.+003+23456

产生一个输出文件：child.example.keyset。这个文件应被传送到上级域去用于标示。它包括密匙，还有基于域密匙自己生成的密匙系列的签名，这些签名用于私有密匙属主的证明，并且可以用于有效期的加密。

5.7.3 标志子域系列（Child’s Keyset）

dnssec-signkey 程序是用来标识子域系列。

如果 child.example 域有一些保密子域，例如，grand.child.example，于是 child.example 的管理员应该从每个保密的子域收到域密匙系列（keyset）文件。这些密匙使用这个域自己的域密匙加密。

下列命令用域密匙来加密子域密匙 key set：

dnssec-signkey grand.child.example.keyset Kchild.example.+003+12345 Kchild.example.+003+23456

生成了一个文件: signedkey-grand.child.example.。这个文件应该保存在本地，并传送回子域。它包含来自 keyset 文件的所有的密匙（子域的密匙），以及被这个域地域密匙生成的签名。

5.7.4 对域进行标记

dnssec-signzone 程序用来标志一个域。

所有保密域的相关 signedkey 文件都应该存在，当然，这个域的 signedkey 文件是由它的上级域（如果有的话）生成的。域的标志器会为本域生成 NXT 和 SIG 记录，并和来自上级域的密匙签名相结合，来标明所有纪录的安全状态。

下列命令用来标记一个域，假设它的 hosts 文件叫做 zone.child.example。默认情况下，所有具有有效私人密匙的域密匙，都会用来生成签名。

dnssec-signzone -o child.example zone.child.example

产生了一个输出文件：zone.child.example.signed。作为一个域的输入文件，named.conf 会引用这个文件。

5.7.5 配置服务器

不象 BIND8，BIND9 不用在启动的时候对数据进行校验，所以授权域的密匙不用在配置文件中设置。但所有保密域根域的公开密匙，则应该写在配置文件的 trusted-keys 后面。

5.8 BIND9 的 IPv6 支持

BIND9 完全支持所有当前 IPv6 中对名字到地址和地址到名字查询的定义。当运行在 IPV6 兼容系统的时候，它也使用 IPV6 地址进行请求。

对于转发的查询，BIND9 同时支持 A6 和 AAAA 记录。我们不赞成使用 AAAA 记录，但主机同时具有对 AAAA 和 A6 记录的支持也是有用的，尤其是维护那些使用向下兼容安装方式的系统，这些系统上仍然使用了 AAAA 纪录。事实上，大多数操作系统所带的解析器只支持 AAAA 的解析查询，因为 A6 的解析要比 A 和 AAAA 的解析实现起来困难得多。

对于 IP6 反向解析，BIND9 支持在域 ip6.arpa 域中使用的新“bitstring”格式，同时也对于在 ip6.int 域使用的老版本的，不建议使用的“单元组”格式化也是一样的。

BIND9 包括新的轻量级的解析库和解析器后台进程，它使用了一些新的程序来避免 A6 的 chain following 和 bitstring 标签的复杂性。

5.8.1 使用 AAAA 记录的地址查找

AAAA 记录和 IPV4 的 A 记录相似。它在一个单独的记录中设定了全部地址。例如：

```
$ORIGIN example.com.
host    3600    IN    AAAA 3ffe:8050:201:1860:42::1
```

尽管不提倡使用它们，但是它们支持老版 IPV6 软件。当完全不需要的时候，就不应该

使用。

5.8.2 使用 A6 记录查询地址

A6 记录比 AAAA 记录更灵活，也就更复杂。A6 记录可以用来形成一个 A6 记录的链，其中每个记录设定了一部分的 IPV6 地址。它也可以用来指定整个的记录。例如，这条纪录和前例中的 AAAA 记录一样：

```
$ORIGIN example.com.
host    3600    IN    A6 0 3ffe:8050:201:1860:42::1
```

5.8.2.1 A6 链（Chains）

A6 记录允许 network renumbering。但只有当一个 A6 记录设置了域自主控制的地址段时起作用。例如，一个主机名字可能是公司名“company”。它有两个为之提供 IPV6 地址空间的 ISP。这两个 ISPs 完全设定了他们提供的 IPV6 前缀。

在公司的地址段中：

```
$ORIGIN example.com.
host    3600    IN    A6    0:0:0:0:42::1 company.example1.net.
host    3600    IN    A6    0:0:0:0:42::1 company.example2.net.
```

ISP1 会使用:

```
$ORIGIN example1.net.
company    3600    IN    A6    0 3ffe:8050:201:1860::
```

ISP2 会使用:

```
$ORIGIN example2.net.
company    3600    IN    A6    0    1234:5678:90ab:fffa::
```

当需要解析 host.example.com 时，解析器（在解析器后台进程或 caching 域名服务器中）会发现 2 个部分的 A6 记录，并使用其它的名字来找到其余的数据。

5.8.2.2 DNS 服务器的 A6 记录

当 A6 记录设置了一个域名服务器的地址，它应该使用完整的地址而不是部分地址。例如：

```
$ORIGIN example.com.
@              14400      IN          ns0
               14400      IN          ns1
ns0            14400      IN        A6        3ffe:8050:201:1860:42::1
```

```
ns1             14400     IN      A       192.168.42.1
```

建议不使用 IPv4-in-IPv6 的地址映射。如果主机是 IPV4 地址，就使用 A 记录，而不是 :ffff:192.168.42.14.8.3 的 A6 记录，

5.8.3 使用 Nibble 格式进行地址到名字的查询

虽然不建议使用 Nibble 格式查找名字，但它能向下兼容支持现有的 IPV6 应用。当解析一个 nibble 格式的地址时，地址块被简单的翻转了，在名字的结果后面附加上一个 ip6.int.，就象在 IPv4 中一样。例如，下面的例子会为一台主机提供反转名称查询，地址是 3ffe:8050:201:1860:42::1

```
$ORIGIN 0.6.8.1.1.0.2.0.0.5.0.8.e.f.f.3.ip6.int.
1.0.0.0.0.0.0.0.0.0.0.0.2.4.0.0     14400     IN      PTR     host.example.com.
```

5.8.4 使用 bitstring 格式进行地址到名字的查询

位串标签可以在任何位边界上开始和结素，而不是象在单元组中那样在四个字节的倍数上。它们使用 ip6.arpa 而不是 ip6.int
用位串复制前面的例子：

```
$ORIGIN [x3ffe805002011860/64].ip6.arpa.
[x0042000000000001/64]     14400     IN     PTR       host.example.com.
```

5.8.5 用 DNAME 来标示 IPV6 的反向地址

在 IPV6 中，同一台主机可能有许多来自不同网络提供商的地址。既然地址得结尾部分常常不变，DNAME 可以帮助减少需要维护的域反向映射文件的数量。例如，一台有两个提供商（example.net 和 example2.net）的主机，这样有两个 IPV6 地址。既然主机选择了它自己的 64 位主机地址部分，变化得部分就只是提供商的地址了：

```
$ORIGIN example.com.
host        IN   A6   64    ::1234:5678:1212:5675 cust1.example.net.
                 IN   A6   64    ::1234:5678:1212:5675 subnet5.example2.net.

$ORIGIN example.net.
cust1       IN   A6   48    0:0:0:dddd:: ipv6net.example.net.
ipv6net     IN   A6   0     aa:bb:cccc::

$ORIGIN example2.net.
```

```
subnet5   IN   A6   48   0:0:0:1:: ipv6net2.example2.net.
ipv6net2  IN   A6   0    6666:5555:4::
```

这里设置了正向查询。为了解决反向解析，提供者 example.net 应该设置：

```
$ORIGIN \[x00aa00bbcccc/48].ip6.arpa.
\[xdddd/16]    IN   DNAME       ipv6-rev.example.com.
```

并且 example2.net 应该设置:

```
$ORIGIN \[x666655550004/48].ip6.arpa.
\[x0001/16]    IN   DNAME ipv6-rev.example.com.
```

example.com 只需要一个域文件处理这些反向映射:

```
$ORIGIN ipv6-rev.example.com.
\[x1234567812125675/64]    IN   PTR       host.example.com.
```

第六章 BIND9 lightweight 解析

6.1 lightweight 解析库

传统的程序与一个存根解析库连接，解析库向一台本地缓存服务器发送递归 DNS 查询。IPV6 给解答进程引入了新的复杂性，如 A6 链和 DNAME 下的记录，和对 IPV4 和 IPV6 地址的同时查询。这些都很难在一个传统的存根服务器中实现。

相反，BIND9 给本地用户提供解决服务，通过混合使用 lightweight 解析库和在本地主机上运行的后台解答进程。这些通信使用一个简单的基于 UDP 的协议，“lightweight 解析协议”区别于并且比完全 DNS 协议简单。

6.2 运行后台解析

为了使用 lightweight 解析接口，系统必须后台运行 lwtesd。

在默认形式下，使用 lightweight 解析库的程序会把 UDP 请求对应于端口 921 的 IPV4loopback 地址（127.0.0.1）.地址会被/etc/resolv.conf.的 LW 服务器行 overridden(忽略)。后台运行程序试图找到“什么是主机 foo.example.com 的地址”和“什么是 IPV4 地址 10.1.2.3 的名字？”

后台程序目前只在 DNS 中访问，但将来它可能使用其他资源如/etc/hosts, NIS。

lwresd 后台程序基本上一个缓存名称服务器，回答是用 lightweight 解析协议而 DNS 协议的请求。因为它需要在每个主机上运行，它设计成要求很低的配置。除非另外被配置，它使用/etc/resolv.conf 中列在名称服务器行上的名称服务器作为转发器，它也能够在没有被设定的情况下自动被解析。Lwest 程序也可以使用 named.conf 文件类型进行配置，默认是在/etc/lwresd.conf。一个名称服务器也可以配置作为一台 lightweight 解析器，使用 named.conf.的 lwres 语句。

第七章 BIND9 配置参考

BIND9 配置大致与 BIND8.X 类似；尽管如此，也存在一些不同的地方，如视图（view）。BIND9 对 BIND8.X 配置文件只做了一小部分变动，尽管应该做更复杂的检查以确定执行 BIND9 中的新功能是否更有效率

contrib/named-bootconf/named-bootconf.sh 是个脚本文件，BIND4 配置文件可以被其转换成新的格式。

7.1 配置文件的组成元素

下列是 BIND 配置文件的元素清单：

acl_name	通过 acl 语句来定义访问控制列表名。
address_match_list	一个或多个 ip_addr，ip_prefix，key_id，或者 acl_name 的表列。
domain_name	用引号扩起来的 DNS 名，例如："my.test.domain".
dotted_decimal	一个或多个整数，值从 0 到 255，用点（”.”）分割（如 123.45.67 或 89.123.45.67。）
ip4_addr	十进位数字表示的，分 4 段的 IPV4 地址。
ip6_addr	一个 IP6 地址，形如：fe80::200:f8ff:fe01:9742
ip_addr	一个 IP4 或者 IP6 地址
ip_port	一个 IP 端口号：值限制在从 0 到 65535，超级用户的进程使用小于 1024 的端口。在一些情况下星号（’*’）作为一个占位符选择一个随机高位端口。
ip_prefix	表示一段网络。即在 ip_addr 后面跟‘/’和网络掩码的位数。 IP 地址是 0 的可以忽略。例如，127/8 是网络和 127.0.0.0，掩码是 127.0.0.0，1.2.3.0/28 是网络 1.2.3.0，掩码是 255.255.255.240。
key_id	共享密匙的名字，一般用 doamin_name，用于安全传输。
key_list	一个或多个 key_id 的表列，以分号来分割和结束。
number	一个非负 32 位无符号整数（例如，包括从 0 到 4294967295）的数。它的数值可以进一步在使用时由上下文所限制
path_name	一个用作路径名的用引号引起来的字符串，如 zones/master/my.test.domain。
size_spec	一个数字、unlimited 或是 default。size_spec 是 unlimited，则说明可以无限使用或是使用最大的值。size_spec 是 default，则说明服务器启动的时候会使用的有限的值。 如果是数字的话，可以跟上单位，如，K 或 k 表示 kilobytes（千字节），M 或 m 表示 megabytes(兆字节)，G 或 g 表示 gigabytes(十亿字节)，大小分别是 1024, 1024*1024, and 1024*1024*1024。值必须可以表示成一个 64 位的无符号整数（0 到 18446744073709551615）。使用 unlimited 是最好的设定一个大数字的方法。
yes_or_no	YES 或 NO。TRUE 和 FALSE 也可以，也就是数字 1 和 0。

dialup_option	可以是 Yes, no, notify, notify-passive, refresh, passive 之一。当在域中使用时，notify-passive, refresh, passive 限制在辅域和末端域中使用。

7.1.1 地址匹配表

7.1.1.1 语法

address_match_list = address_match_list_element ;
[address_match_list_element; ...]
address_match_list_element = [!] (ip_address [/length] |
key key_id | acl_name | { address_match_list })

7.1.1.2 定义和使用

地址匹配列表主要用于多服务器操作中的访问控制。它也被用来定义查询其他名称服务器的优先级和设置接受查询的命名的地址。组成一个地址表列的元素可以是下列内容：

- 一个ＩＰ地址（ＩＰＶ４或者ＩＰＶ６）
- 一个ＩＰ段（带‘／’）
- key 语句中定义的一个 key ID
- 已经由 acl 语句定义了的老的地址表列名
- 用大括号扩起来的地址匹配列表

地址前可以使用感叹号（“！”），表示非。表列名中的"any" "none" "localhost"和"localnets"已经被提前定义了。更多关于这些名称的信息可以在 acl 语句的叙述中找到。

当使用一个访问控制表列时，一个不带!号的访问列表表示允许访问，而带!的列表则表示拒绝访问。如果都不匹配的话，访问会拒绝。子句 allow-notify, allow-query, allow-transfer, allow-update 和 blackhole 都可以这样使用地址匹配表。类似的，listen-on 选项使得服务器不接受任何不匹配列表的主机地址的请求。

当与 topology 字句同时使用时，一个不带!号的列表，地址在列表中的位置（越靠近开始部分，则它和服务器之间的距离越短），将表示一个距离。一个带!号的列表则表示它和服务器之间分配了最大的距离。如果没有匹配的话，地址将会得到比任何不带！的列表地址都远，但比任何带！的列表都近的的一个距离。

由于顺序优先匹配算法的定义，一个小的范围定义一定要在更大的范围定义之前，不管是否带有!号。例如：1.2.3/24; !1.2.3.13。1.2.3.13 地址定义是完全无用的，因为算法会通过 1.2.3/24 的定义，而先匹配任何到 1.2.3.13 的查询。使用!1.2.3.13;1.2.3/24 则解决了这个问题，除了 1.2.3.13 外，而所有其它 1.2.3.*的地址都可通过。

7.1.2 语法注释

BIND9 语法注释允许注释出现在任何 BIND9 配置文件可能存在空白空间的地方。为适用于任何程序员，它们可以用 C，C++或 shell/perl 编写。

7.1.2.1 语法

/* 这是一个类似于 C 中的 BIND 注释 */
// 这是一个类似于 C++中的 BIND 注释
这是一个类似于普通 Unix 的 shell 和 perl 中的 BIND 注释

7.1.2.2 定义和用法

注释允许出现在任何 BIND9 配置文件可能存在空白空间的地方。C 风格的注释用两个符号/* 开头，以 */ 结束。因为它们是完全以这些符号划界的，它们可以用来注释一行的一部分或者多行空间。

C 型注释不能嵌套。例如，下列是无效的因为整个注释以第一个*/结尾：

/* 注释的开头
这仍然是注释的一部分
/* 这是一个错误的注释嵌套 */
这不是任何注释 */

C++风格的注释以 // 开头，直到行尾。它们不能延续于多个物理行；需要一个的多行注释的话，每行必须使用一对 //。例如：

// 这是注释的开头。下一行
// 新注释，即使是逻辑上的
// 部分上一级注释

shell 风格的（或者 perl 风格）注释以符号# 开始，直到物理行的结尾，就象在 C++注释。
例如：

#这是注释的开始。下一行
#新注释，即使是逻辑上的
#上一级注释的部分。

注意：不能象在域文件中的那样，使用分号‘;’开始一个注释。分号表示配置语句的结尾。

7.2 配置文件语法

一个 BIND9 配置包括语句和注释。语句以分号结尾。语句和注释是唯一可以不使用大扩号的内容。许多语句包括一块子语句，也以分号结尾。
支持语句如下：

acl	定义一个ＩＰ地址表列名，用语接入控制和其他用法。
controls	宣告 rnde utility 使用的控制通道（channel）
include	包含一个文件
key	设置密匙信息，它应用在通过 TSIG 进行授权和认证的配置中
logging	设置日志服务器，和日志信息的发送地
options	控制服务器的全局配置选项和为其它语句设置默认值
server	在一个单服务器基础上设置特定的配置选项
trusted-keys	定义信任的 DNSSED 密匙
view	定义一个视图
zone	定义一个域

logging 和 options 语句只在每个配置中出现一次。

7.2.1 acl 语句语法

```
acl acl-name {
    address_match_list
};
```

7.2.2 acl 语句定义和使用

acl 语句给一个地址匹配表列赋了一个象征名称。它的名字来自于地址匹配列表的最基本功能：访问控制表列（ACLs）。
注意，一个地址表列名必须首先在 acl 中定义了，然后才能在别处使用；提前调用是不允许的。
下列 ACLs 组成：

any	匹配所有主机
none	不匹配任何主机
localhost	匹配主机上所有 IPV4 的网络接口
localnets	匹配所有 IPV4 本地网络的主机

localhost 和 localnets 的 ACLs 目前不支持 IPV6（也就是说，localhost 不匹配主机的 IPV6 地址，localnets 不匹配连上 IPV6 网络的主机），因为缺乏确定本地 IPV6 主机地址的标准方法。

7.2.3 控制语句语法

```
controls {
      inet ( ip_addr | * ) [ port ip_port ] allow { address_match_list }
            keys { key_list };
            [ inet ...; ]
};
```

7.2.4 controls 语句定义和用法

controls 语句定义了系统管理员使用的，有关本地域名服务器操作的控制通道。这些控制通道被 rndc 用来发送命令，并从域名服务器中检索非 DNS 的结果。

inet 控制通道是一个监听在 ip_addr（可以是 ipv4 或 ipv6 地址）地址上的 ip_port 端口的 TCP socket。Ip_addr 是*的话，则说明是一个 ipv4 的统配符；允许接受系统上的任何 IPV4 的地址上的用户连接。要想监听所有 ipv6 地址上的连接，则应该使用::作为 ip_addr。如果仅想在本地主机上使用 rndc，建议使用 loopback 地址(127.0.0.1 或::1)以获得最大安全性。Allow 和 keys 子语用来限制通过控制通道发出命令的能力。根据 address_match_list 中的定义，来监控控制通道上的用户连接。address_match_list 中的 key_id 成员则被忽略，反之则根据 key_list 来单独的表示。每个在 key_list 中的 key_id 都允许用来鉴别通过控制通道传送的命令和响应，这些服务器和用户端之间的命令和响应都经过了数字签名的技术处理。所有经过控制通道的命令都必须使用设置的密匙进行加密。如果没有设置 controls 语句，named 就会建立一个默认控制通道，监听 loopback 地址 127.0.0.1 和对应的 IPV6 地址::1。当有一个 controls 语句，但没有 key 子句时，named 将会试着从文件/etc/rndc.key（或者任意在 BIND 编译时设定的 sysconfdir）中读取命令通道密匙。用 rndc-confgen –a 命令，创建一个 rndc.key 文件。

rndc.key 是用来简化从 BIND8 系统上的升级过程，因为 BIND8 在它的命令通道中没有使用数字签名，也就没有 key 子句。这样，在 BIND9 安装后执行 rndc-confgen –a 命令，BIND9 就可以继续使用和 BIND8 一样的配置文件，并仍然使用类似于 BIND8 中的 ndc 一样的工作方式。

既然 rndc.key 的特性只是用于兼容 BIND8 配置文件，所以它的配置就没有什么高难度。当用户想修改加密的信息时，当然不能简单的修改密匙名称和加密程度，而应该用用户自己的密匙来生成新的 rndc.conf 文件。rndc.key 文件也有自己的读取权限设置，只有文件的所有者（运行 named 的用户）可以访问。如果用户要求能有更多的用户能够使用 rndc，则生成 rndc.conf 文件时，将文件属组的权限置为可读，并将这些用户归入这个组就可以了。BIND8 中的 UNIX 控制通道类型在 BIND9 中不支持，而且以后也不会加入到新版本中。如果于来自 BIND8 配置文件中存在 controls 语句中，它会被忽略并且会记录下一个警告。
要想禁止掉命令通道，则使用一条空 controls 语句：controls { }; 即可。

7.2.5 include 语句语法

include 文件名;

7.2.6 包含语句定义和使用

include 语句可以在 include 语句出现的地方插入指定的文件。Include 语句通过允许对配置文件的读或写，来简化对配置文件的管理。例如，它可以包含多个只能由域名服务器读取的私人密匙（private key）。

7.2.7 键语句语法

key *key_id* {
    algorithm *string*;
    secret *string*;
};

7.2.8 key 语句的定义和使用

key 语句定义了一个用于 TSIG 的共享密匙。

key 语句可以出现在配置文件的开始或者在一个 view 语句中。定义在开始的 key 语句可以在所有视图中应用。在 controls 语句中使用的 key 必须事先定义在文件的开始。

key_id，也叫做密匙名，是确认一个域名的唯一密匙。可以在一个“server”语句中使用，使得发给这个服务器的请求都会用这个密匙进行加密，或者用于确认来自于地址匹配列表中的主机的请求，是否已经用这个名字、算法和 secret 的密匙进行了加密。

algorithm_id 是一个标记安全/鉴定的字符串。目前唯一由 TSIG 鉴别支持的算法是 hmac-md5。secret_string 是算法要使用的机密级，是一个 64 位编码的字符串。

7.2.9 logging 语句语法

logging {
    [channel *channel_name* {
        (file *path name*
            [versions (*number* | unlimited)]
            [size *size spec*]
            | syslog *syslog_facility*
            | stderr
            | null);
        [severity (critical | error | warning | notice | info |debug [*level*] | dynamic);]
    [print-category yes ot no;]
    [print-severity yes or no;]
    [print-time yes or no;]
};]
[category *category_name* {

channel_name ; [*channel_name ; ...*]
};]
...
};

7.2.10 Logging 语句定义和使用

logging 语句为域名服务器设定了一个多样性的 logging 选项。它的 channel 短语对应于输出方式、格式选项和分类级别，它的名称可以与 category 短语一起定义多样的日志信息。只用一个 logging 语句就可以用来定义多个 channel 和 category。如果没有 logging 语句的话，logging 设置就是：

```
logging {
    category "unmatched" { "null"; };
    category "default" { "default_syslog"; "default_debug"; };
};
```

在 BIND9 中，logging 的配置只有在整个配置文件被读取后才被执行。而在 BIND8 中，logging 部分被读取后就开始执行了。当服务器启动时，所有在配置文件中关于语法错误的 logging 信息都转到缺省通道（channel）中，或者使用”-g”选项，指定转成标准错误。

7.2.10.1 channel 短语

所有日志会输出到一个或多个 channel 中；你可以定义所有你想要的通道。

每个通道的定义必须包括一个目的字句，用来确定所选的相关同到的信息，将会输出到一个文件，或是到一个特殊的 syslog 工具，或是到一个标准错误流，或者被忽略。它也可以随意的限制通道能接受的信息级别（默认值 info），定义是否包含一个由 named 产生的时间标记，或者是否包含分类的名称、级别等（默认是不包含任何内容）。

目的子句为 null 时，会使所有发送给通道的信息被丢弃；那样的话，其他通道选项就没有意义了。

目的子句为 file 时，会使通道的内容输出到一个磁盘文件。它可以包含这个文件的大小和该文件可以保存多少个版本。

如果使用 versions 日志文件选项，named 就会自动保留多个版本的日志文件。例如，如果选择保存文件 lamers.log 的三个老版本，那么在它被打开的时候 lamers.log.1 被更名为 lamers.log.2，lamers.log.0 被更名为 lamers.log.1 ，lamers.log 被更名为 lamers.log.0。也可以设置 version unlimited，这样就没有备份版本的限制了。如果对日志文件设置了 size 选项，那么仅当此文件超过了设定的大小时，系统就会进行更名。默认情况下不储存备份文件；所有存在的日志文件被简单进行追加。

文件的 size 选项用来限制日志的增长。如果文件超过了限制，又没有 version 选项，则 named 就会停止写入文件。如果保留了备份版本，则备份文件如上所述进行滚动命名，然后开始创建一个新的文件。如果没有 versions 选项，也没有其它的机制来删除或减小日志文件，则系统就不会有数据继续写入日志中。默认状态是不限制文件的大小的。

size 和 versions 选项的使用例子：

```
channel "an_example_channel" {
    file "example.log" versions 3 size 20m;
    print-time yes;
    print-category yes;
};
```

syslog 目的子句是把通道指向系统日志。它的参数是一个 syslog 的前缀，如 syslog 帮助中所述。Syslog 是怎样处理带有这些前缀的信息，可以参考 syslog.conf 的帮助信息。如果你有一个使用很老版本的 syslog 的系统，只对 openlog()函数使用两个参数，那么这个子句就被忽略了。

severity 子句象 syslog 中的”priorites”一样工作，唯一区别的是用户可以直接写入一个文件，而不是使用 syslog 写入一个文件。不到严重级的信息将不会被通道选择；高严重级的信息将会被接受。

如果用户正在使用 syslog，那么 syslog.conf 的优先级也会决定什么会最终通过。例如，将 channel facility 和 severity 定义成 daemon 和 debug，就不会只记录通过 syslog.conf 的 daemon.warning 信息，后者会使 severity 是 info 和 notice 的信息被丢弃。如果情况相反，named 就会只记录 warning 或更高级别的信息，而 syslogd 则会记录来自于通道的所有信息。

Stderr 的目的子句将通道输出到服务器的标准错误流。它的用于服务器在前台运行的情况下，例如，debug 调试配置的时候。

当处于 debug 模式的时候，服务器能提供丰富的调试信息。如果服务器的全局 debug 级别（globe debug level）大于 0，debug 模式将被激活。全局 debug 级别可以通过在启动 named 时设置“-d”参数加一个正数，或运行 rndc trace 来设置。

全局 debug 级别可以设置成 0，或运行 rndc notrace，则 debug 模式被关闭。服务器中所有的 debug 信息有一个 debug 级别，高调试级给出更详细的输出。例如，指定调试严重级别的通道：

```
channel "specific_debug_level" {
    file "foo";
    severity debug 3;
};
```

上例中，服务器在处于 debug 模式的时候都会收到 3 级和比 3 级小的级别的调试信息，全局的调试级别在这里不起作用。dynamic 严重级别的通道将使用服务器全局 debug 级别决定打印哪些信息。

如果使用了 print-time 参数，则日期和时间也将会记录下来。print-time 也可以针对 syslog 的通道进行设置，但因为 syslog 也打印日期和时间，所以一般来讲，这没有什么意义。如果设置了 print-category 参数，则信息的分类也会记录下来。如果设置了 print-severity 参数，则信息的严重级别也会记录下来。print-xxx 选项可以进行多重组合，单输出的格式都为这个顺序：时间、分类、严重级别。下面是一个当三个打印选项都设置的例子：

```
28-Feb-2000 15:05:32.863 general: notice: running
```

下面是四个 named 提前定义的通道，用于指定缺省的日志。

```
channel "default_syslog" {
        syslog daemon;                          //   发送给 syslog 的 daemon facility
        severity info;                          //只发送此优先级和更高优先级的信息
};
channel "default_debug" {
        file "named.run";                       // 写入工作目录下的 named.run 文件
                                                // 注意：如果服务器用-f 参数启动，则
                                                // "named.run"会被 stderr 所替换
        severity dynamic;                       // 按照服务器当前的 debug 级别记录日
志
};
channel "default_stderr"{                       //写到 stderr
        stderr;
        severity info;                          //只发送此优先级和更高优先级的信息
};
channel "null" {
        null;                                   // 丢弃所有发到此通道的信息
};
```

default_debug 通道有特殊的性质：只有当服务器的 debug 级别非 0 的时候，它才产生输出。一般来说，它会在服务器的工作目录中写入 named.run 文件。

因为安全原因，当在命令行选项中使用了“-u”参数后，只有当 named 使用了新的 UID 后，named.run 文件才会产生，以 root 身份启动和运行的 named 所产生的 debug 信息将会被丢弃。如果用户需要得到这些输出，则必须使用“-g”参数运行服务器，并重新将标准错误定向到一个文件中去。

一旦定义好一个通道，它就不能被重新定义。这样就不能修内置的通道，但是可以通过把分类指向你已经定义的通道，来修改默认的日志记录。

7.2.10.2 category 短语

这里存在许多分类，用户可根据需要定义想看到或不想看的日志。如果你不将某个分类指定到某些通道的话，那么在这个分类的日志信息就会被发送到 default 分类通道中。如果用户没有设定缺省的分类，下列"default"则会被系统使用：

```
category "default" { "default_syslog"; "default_debug"; };
```

作为一个例子，假定你要在文件中记录安全事件，但您也要保留缺省的日志文件。最好按照下面配置：

```
channel "my_security_channel" {
        file "my_security_file";
        severity info;
```

```
};
category "security" {
        "my_security_channel";
        "default_syslog";
        "default_debug";
};
```

为了丢弃一个分类中的所有信息，可以设定 null 通道：

```
category "xfer-out" { "null"; };
category "notify" { "null"; };
```

下面是可用的分类和相关的简明描述，以后的 BIND 版本中会包含更多的分类。

default	没有配置的分类会使用 default 的分类的日志配置。
General	许多没有分类的内容都归在此分类中
Database	Named 使用的，用来存储域和缓存数据的内部数据库信息。
security	接受和拒绝的请求
config	配置文件分析和处理
resolver	DNA 解析，如由缓存域名服务器执行的代表用户的递归查询。
xfer-in	服务器收到的域传输
xfer-out	服务器发送的域传输
notify	NOTIFY 协议
client	用户请求的处理
unmatched	由于没有匹配的视图，Named 无法确定的类别。单行的概要信息记录在 client 分类中。这个分类最好发送到一个文件或者 stderr，缺省情况下发送到一个 null 通道。
network	网络操作。
update	动态更新。
queries	请求，使用 queries 分类将会产生日志用户请求。
dispatch	将进入的数据包分发到能够处理它们的服务器模块中去。
dnssec	DNSSEC 和 TSIG 协议的处理。
lame-servers	未知服务器。这些都是有于其它 DNS 服务器中的错误配置引起的。

7.2.11 lwres 语句语法

这是在 name.conf 文件中的 lwres 语句语法

```
lwres{
   [ listenon-on { ip_addr [port ip_port] ; [ ip_addr [port ip_port] ; ... ] }; ]
   [ view view_name; ]
   [ search { domain_name ; [ domain_name ; ... ] }; ]
   [ ndots number; ]
};
```

7.2.12 lwres 语句定义和用法

lwres 语句也可以同时将域名服务器设置为一个轻量级的解析服务器。由很多的 lwres 语句可以将这个轻量级的解析服务器配置成不同的属性。

listen-on 语句可以设定一个地址（和端口）的列表，解析服务器的后台进程会在这些地址和端口上接受请求。如果不指定端口，就使用 921 端口。如果这个语句遗漏了的话，请求将会在 127.0.0.1，端口 921 接收。

view 语句将 lightweight 解析进程和 DNS 域名空间的视图相结合，这样用户得到的响应，就象是一个根据 view 中的内容返回的，一个普通的 DNS 请求。如果语句被忽略，就使用默认视图，如果没有默认视图，就会出发错误。

search 语句类似于/etc/resolv.conf 中的 search 语句。它提供一个域名的列表，这些域名将会被附加在相关请求的名字后面。

Ndot 语句类似于/etc/resolv.conf 中的 ndots 语句。它标明了在一个相对域名中最少的点的数量。

7.2.13 options 语句语法

这是在 named.conf 文件中 options 语句的语法：

```
options {
    [ version version_string; ]
    [ directory path_name; ]
    [ named-xfer path_name; ]
    [ tkey-domain domainname; ]
    [ tkey-dhkey key_name key_tag; ]
    [ dump-file path_name; ]
    [ memstatistics-file path_name; ]
    [ pid-file path_name; ]
    [ statistics-file path_name; ]
    [ zone-statistics yes_or_no; ]
    [ auth-nxdomain yes_or_no; ]
    [ deallocate-on-exit yes_or_no; ]
    [ dialup dialup_option; ]
    [ fake-iquery yes_or_no; ]
    [ fetch-glue yes_or_no; ]
    [ has-old-clients yes_or_no; ]
    [ host-statistics yes_or_no; ]
    [ minimal-responses yes_or_no; ]
    [ multiple-cnames yes_or_no; ]
    [ notify yes_or_no | explicit; ]
    [ recursion yes_or_no; ]
    [ rfc2308-type1 yes_or_no; ]
```

[use-id-pool yes_or_no;]
[maintain-ixfr-base yes_or_no;]
[forward (only | first);]
[forwarders { ip_addr [port ip_port] ; [ip_addr [port ip_port] ; ...] };]
[check-names (master | slave | response)(warn | fail | ignore);]
[allow-notify { address_match_list };]
[allow-query { address_match_list };]
[allow-transfer { address_match_list };]
[allow-recursion { address_match_list };]
[allow-v6-synthesis { address_match_list };]
[blackhole { address_match_list };]
[listen-on [port ip_port] { address_match_list };]
[listen-on-v6 [port ip_port] { address_match_list };]
[query-source [address (ip_addr | *)] [port (ip_port | *)];]
[max-transfer-time-in number;]
[max-transfer-time-out number;]
[max-transfer-idle-in number;]
[max-transfer-idle-out number;]
[tcp-clients number;]
[recursive-clients number;]
[serial-query-rate number;]
[serial-queries number;]
[transfer-format (one-answer | many-answers);]
[transfers-in number;]
[transfers-out number;]
[transfers-per-ns number;]
[transfer-source (ip4_addr | *) [port ip_port] ;]
[transfer-source-v6 (ip6_addr | *) [port ip_port] ;]
[notify-source (ip4_addr | *) [port ip_port] ;]
[notify-source-v6 (ip6_addr | *) [port ip_port] ;]
[alsonotify { ip_addr [port ip_port] ; [ip_addr [port ip_port] ; ...] };]
[max-ixfr-log-size number;]
[coresize size_spec ;]
[datasize size_spec ;]
[files size_spec ;]
[stacksize size_spec ;]
[cleaning-interval number;]
[heartbeat-interval number;]
[interface-interval number;]
[statistics-interval number;]
[topology { address_match_list }];
[sortlist { address_match_list }];
[rrset-order { order_spec ; [order_spec ; ...] } };
[lame-ttl number;]

```
    [ max-ncache-ttl number; ]
    [ max-cache-ttl number; ]
    [ sig-validity-interval number ; ]
    [ min-roots number; ]
    [ use-ixfr yes_or_no ; ]
    [ provide-ixfr yes_or_no; ]
    [ request-ixfr yes_or_no; ]
    [ treat-cr-as-space yes_or_no ; ]
    [ min-refresh-time number ; ]
    [ max-refresh-time number ; ]
    [ min-retry-time number ; ]
    [ max-retry-time number ; ]
    [ port ip_port; ]
    [ additional-from-auth yes_or_no ; ]
    [ additional-from-cache yes_or_no ; ]
    [ random-device path_name ; ]
    [ max-cache-size size_spec ; ]
    [ match-mapped-addresses yes_or_no; ]
};
```

7.2.14 options 语句定义和用法

options 语句设立可以被整个 BIND 使用的全局选项。这个语句在每个配置文件中只有一处。如果出现多个 options 语句，则第一个 options 的配置有效，并且会产生一个警告信息。如果没有 options 语句，每个选项择使用缺省值。

version

回答针对服务器版本的请求时的内容。缺省返回的是服务器的真实版本。

directory

服务器的工作目录。配置文件中所有使用的相对路径，指的都是在这里配置的目录下。大多数服务器的输出文件（如 named.run）都缺省生成在这个目录下。如果没有设定目录，工作目录缺省设置为服务器启动时的目录‘.’。指定的目录应该是一个绝对路径。

named-xfer

这个选项已经被废弃了。它在 BIND8 中，它用来给 named-xfer 程序设定路径名。在 BIND9 中，不需要单独的 named-xfer 程序；它的功能已经内置在域名服务器中。

tkey-domain

这个域名将会附带在由 TKEY 生成的所有共享密匙名字的后面。当用户请求进行 TKEY 交换时，它会为密匙设定或不设定所要求的名称。如果设置了 tkey_domain，共享密匙的名字将会是"client specified part"（用户设定的部分） + "tkey-domain"。否则，共享密匙的名字将是"random hex digits"（随机的 16 进制数） + "tkey-domain"。在大多数情况

下，domainname 应该是服务器的域名。

Tkey-dhkey

针对使用 Diffie-Hellman 的 TKEY 模式的用户，服务器用来生成共享密匙的 Diffie-Hellman 密匙。服务器必须可以从工作目录中调入公共和私人密匙。大多数情况下，密匙的名称应该是服务器的主机名。

dump-file

当执行 rndc dumpdb 命令时，服务器存放数据库文件的路径名。如果没有指定，缺省名字是 named_dump.db。

memstatistics-file

服务器输出的内存使用统计文件的路径名。如果没有指定，默认值为 named.memstats。注意：还没有在 BIND9 中实现！

pid-file

进程 ID 文件的路径名。如果没有指定，默认为/var/run/named.pid。pid-file 是给那些需要向运行着的服务器发送信号的程序使用的。

statistics-file

当使用 rndc stats 命令的时候，服务器会将统计信息追加到的文件路径名。如果没有指定，默认为 named.stats 在服务器程序的当前目录中。

Port

服务器用来接收和发送 DNS 协议数据的 UDP/TCP 端口号。默认为 53。这个选项主要用于服务器的检测；因为如果不使用 53 端口的话，服务器将不能与其它的 DNS 进行通讯。

random-device

服务器使用的 entropy 源：entropy 主要用于 DNSSEC 操作，如 TKEY 的数据交换和加密域的动态更新。此选项指定了 entropy 将会从哪个设备（或文件）中读取信息。如果它是一个文件，则当文件耗尽后，需要 entropy 的操作将会失败。如果没有指定，默认值是/dev/random（或等价的），如果它存在，否则就是没有。random-device 选项是在服务器启动时，初始化配置时起作用的，在以后的重启时则被忽略。

7.2.14.1 Boolean 选项

auth-nxdomain

如果是 yes，那么 AA 位将一直设置成 NXDOMAIN 响应，甚至在服务器不是授权服务器的情况下都是这样的。默认值是 no ； 这与 BIND8 不同。如果用户使用的是非常老版本的 DNS 软件，则有必要把它设置成 yes。

deallocate-on-exit

此选项在 BIND8 中用于检查出口处内存泄露。BIND9 忽略此选项，并始终进行检查。

Dialup

如果是 yes，那么服务器将会象在通过一条按需拨号的链路进行域传送一样，对待所有的域（按需拨号就是在服务器有流量的时候，链路才连通）。根据域类型的不同它有不同的作用，并将集中域的维护操作，这样所有有关的操作都会集中在一段很短的时间内完成，每个 heartbeat-interval 一次，一般是在一次调用之中完成。它也禁止一些正常的域维护的流量。默认值是 no。

dialup 选项也可以定义在 view 和 zone 语句中，这样就会代替了全局设置中 dialup 的选项。

如果域是一个主域，服务器就会对所有辅域发送 NOTIFY 请求。这将激活辅域名服务器中的对域的序列号的检验。这样当建立一个连接时，辅域名服务器才能确认这个域的传输合法性。

如果这个域是一个辅域或是末梢域（stub zone），那么服务器将会禁止通常的“zone up to date”（refresh）请求，为了能发送 NOTIFY 请求，只有在 heartbeat-interval 过期之后才执行。

通过下列的设置，可以实现更好的控制。

1、 notify 只发送 NOTIFY 信息。
2、 notify-passive 发送 NOTIFY 信息，并禁止普通的刷新（refresh）请求。
3、 refresh 禁止普通的刷新处理，当 heartbeat-interval 过期时才发送刷新请求。
4、 passive 只用于关闭普通的刷新处理。

fake-iquery

在 BIND8 中，此选项用来模拟陈旧的 DNS 查询类型 IQUERY。BIND9 不再进行 IQUERY 模拟。

fetch-glue

这个选项以不再使用。

has-old-clients

这个选项在 BIND8 中执行有问题，BIND9 则忽略了这个选项。为了达到 has-old-clients yes 的预期效果，可以设定两个独立选项 auth-nxdomain yes 和 rfc2308-type1 no 来代替。

host-statistics

在 BIND8 中，它可以保留每台和域名服务器交互的主机统计信息。BIND9 中不支持。

maintain-ixfr-base

此选项不再使用了。在 BIND8 用于判定是否保存了增量域传输的处理日志。BIND9 任何可能的时候都会保存传输日志。如果需要禁止流出的增量域传输，可以使用 provide-ixfr no。

minimal-responses

如果是 yes，当产生响应的时候，服务器将只会按照需要将记录添加到 authority 和 additional 的数据部分。（例如，delegations，negative responses）。这样会改善服务器的性能。默认值为 no。

multiple-cnames

这个选项在 BIND8 中使用，允许一个域名承认多条 CNAME 记录（与 DNS 标准相违背）。BIND9.2 在主 hosts 文件和动态更新中都严格强制执行 CNAME 规则。

notify

如果是 yes（默认），当一个授权的服务器修改了一个域后，DNS NOTIFY 信息被发送出去。此信息将会发给列在域 NS 记录上的服务器（除了由 SOA MNAME 标示的主余名服务器）和任何列在 also-notify 选项中的服务器。

如果是 explicit，则 notify 将只发给列在 also-notify 中的服务器。如果是 no，就不会发出任何报文。

Notify 选项也可能设定在 zone 语句中，这样它就替代了 options 中的 notify 语句。如果 notify 会使得辅域名服务器崩溃，就需要将此选项关闭。

recursion

如果是 yes，并且一个 DNS 询问要求递归，那么服务器将会做所有能够回答查询请求的工作。如果 recursion 是 off 的，并且服务器不知道答案，它将会返回一个推荐（referral）响应。默认值是 yes。注意把 recursion 设为 no;不会阻止用户从服务器的缓存中得到数据，它仅仅阻止新数据作为查询的结果被缓存。服务器的内部操作还是可以影响本地的缓存内容，如 NOTIFY 地址查询。

rfc2308-type1

设置成 yes 将会使得服务器发送 NS 记录和关于 negative answer 的 SOA 记录。默认值为 no。

注：BIND9 中还不支持。

use-id-pool

此选项已经不再使用。BIND9 始终都是从池中分配请求 ID 的。

Zone-statistics

如果是 yes，缺省情况下，服务器将会收集在服务器所有域的统计数据。这些统计数据可以通过使用 rndc stats 来访问，rndc stats 命令可以将这些信息转储到 statistics-file 中的文件中去。

use-ixfr

这个选项不再使用。如果需要针对一个或多个特殊的服务器关闭 IXFR，可以参考 provide-ixfr 中的内容。

provide-ixfr

参阅 7.2.16 中关于 provide-ixfr 的陈述

request-ixfr

参阅 7.2.16 中关于 request-ixfr 的陈述

treat-cr-as-space

这个选项应用于 BIND8 中，使服务器正确处理回车（"\r"）字符，就象其它的空格或 tab 字符一样。这样可以便于在 unix 系统上加载由 NT 或 DOS 系统生成的域文件。在 BIND9 中,UNIX"的\n"和 DOS 的"\r\n"都可以正确处理为换新行，这个选项就被忽略了.

additional-from-auth

additional-from-cache

当回答具有 additional 数据的请求，或者当在 CNAME 和 DNAME 串的后面时，这些选项控制一个权威服务器的操作。

当这两个选项都被设成 yes(默认状态)，并且查询的是授权的数据（这个域就配置在本地服务器中）时，回答中的 additional 部分的数据将使用来自于其它授权域和 cache。在许多情况下这是不需要的，比如在缓存内容的正确性受到怀疑的情况下，或是在某些辅域可能被非法修改的服务器。还有，避免对这些 additional 数据的搜索将会加速服务器运转。

例如，如果一个查询需要主机 foo.example.com 的 MX 记录，找到的记录是"MX 10 mail.example.net"，如果知道的话, mail.example.net 的地址记录(A,A6 和 AAAA)也会被提供出来。把选项设置为 no，则禁止了这种操作。

这些选项用于授权的服务器，或者是授权的视图中。把它们设成 no，但没有同时设置 recursion no，将会使得服务器忽略这些选项，并记录一个警告日志。

设定 additional-from-cache 为 no 实际上针对 additional 信息的查询和正在响应的查询，都禁止了缓存的使用。这常常使用在一台授权的服务器中，因为在这里缓存数据的正确性非常重要。

当一台域名服务器不提供递归查询时，并且查询的名称并不在本地域中，一般会对根服务器或者其他已知的上级服务器回答"upwards referral(向上推荐)"。既然在向上查询中的数据来自于缓存，那么当 additional-from-cache 被设定为 no 时，服务器就不能提供向上推荐。相反，它会使用 REFUSED(拒绝)回答这些查询。因为向上推荐不是在用户解析过程中需要的，所以就不会出任何问题。

match-mapped-addresses

如果是 yes，那么一个 ipv4 映射成的 ipv6 地址就会匹配任何地址匹配表中能匹配于对应的 ipv4 地址的记录。打开这个选项，对于运行了 ipv6 的 linux 系统有时非常有用，这样通过地址映射，就可以使得 ipv4 的 TCP 连接（如域传送）实现在 Ipv6 的 soket 上，因为地址匹配列表是给 Ipv4 设计的。

7.2.14.2 转发

转发功能可以用来在一些服务器上产生一个大的缓存，从而减少到外部服务器链路上的流量。它可以使用在和 internet 没有直接连接的内部域名服务器上，用来提供对外部域名的查询。只有当服务器是非授权的，并且缓存中没有相关记录时，才会进行转发。

forward

此选项只有当 forwarders 列表中有内容的时候才有意义。当值是 First，默认情况下，使服务器先查询设置的 forwarders，如果它没有得到回答，服务器就会自己寻找答案。如果设定的是 only，服务器就只会把请求转发到其它服务器上去。

forwarders

设定转发使用的 ip 地址。默认的列表是空的(不转发)。转发也可以设置在每个域上，这样全局选项中的转发设置就不会起作用了。用户可以将不同的域转发到服务器上，或者者对不同的域可以实现 forward only 或 first 的不同方式，也可以根本就不转发。

7.2.14.3 访问控制

可以根据用户请求使用的 IP 地址进行限制。

allow-notify

设定哪个主机上的辅域（不包括主域）已经进行了修改。allow-notify 也可以在 zone 语句中设定，这样全局 options 中的 allow-notify 选项在这里就不起作用了。但它只对辅域有效。如果没有设定，默认的是只从主域发送 notify 信息。

allow-query

设定哪个主机可以进行普通的查询。allow-query 也能在 zone 语句中设定，这样全局 options 中的 allow-query 选项在这里就不起作用了。默认的是允许所有主机进行查询。

allow-recursion

设定哪台主机可以进行递归查询。如果没有设定，缺省是允许所有主机进行递归查询的。注意禁止一台主机的递归查询，并不能阻止这台主机查询已经存在于服务器缓存中的数据。

allow-v6-synthesis

设定哪台主机能接收对 ipv6 的响应。

allow-transfer

设定哪台主机允许和本地服务器进行域传输。allow-transfer 也可以设置在 zone 语句中，这样全局 options 中的 allow-transfer 选项在这里就不起作用了。如果没有设定，默认值是允许和所有主机进行域传输。

Blackhole

设定一个地址列表，服务器将不会接收来自这个列表的查询请求，或者解析这些地址。从这些地址来的查询将得不到响应。默认值是 none。

7.2.14.4 接口

接口和端口(服务器回答来自于此的询问)可以使用 listen-on 选项来设定。listen-on 使用可选的端口和一个地址匹配表列（address_match_list）。服务器将会监听所有匹配地址表列中所允许的端口。如果没有设定端口，就将使用 53。
允许使用多个 listen-on 语句。例如:

```
listen-on { 5.6.7.8; };
listen-on port 1234 { !1.2.3.4; 1.2/16; };
```

将在 5.6.7.8 的 ip 地址上打开 53 端口，在除了 1.2.3.4 的 1.2 网段上打开 1234 端口。如果没有设定 listen-on，服务器将在所有接口上监听端口 53。

listen-on-v6 选项用来设定监听进入服务器的 ipv6 请求的端口。

服务器并不象在 ipv4 中那样对每个 IPV6 端口地址绑定一个独立的 socket。相反，它一直监听 ipv6 通配的地址。这样，对于 listen-on-v6 语句唯一的 address_match_list 的参数就是：

```
{ any; }
```
和
```
{ none;}
```

多个 listen-on-v6 选项可以用来监听多个端口
:

```
listen-on-v6 port 53 { any; };
listen-on-v6 port 1234 { any; };
```

要使服务器不监听任何 ipv6 地址，使用：

```
listen-on-v6 { none; };
```

如果没有设定 listen-on-v6 语句，服务器将不会监听任何 ipv6 地址。

7.2.14.5 查询地址

如果服务器查不到要解析的地址，它将会查询其它域名服务器。query-source 可以用来设定这类请求所使用的地址和端口。对于使用 ipv6 发送的查询，有一个独立的 query-source-v6 选项。如果 address 是*或者被省略了，则将会使用一个通配的 IP 地址 (INADDR ANY)。如果 port 是*或者被省略了，则将会使用一个随机的大于 1024 的端口。默认为:

```
query-source address * port *;
query-source-v6 address * port *;
```

注：query-source 选项中设置的地址是同时用于 UDP 和 TCP 两种请求的，但是 port 仅仅用于 UDP 请求。TCO 请求使用的是随机的大于 1024 的端口。

7.2.14.6 域传输

BIND 有适当的机制来简化域传输，并限定系统传输的负载量。下列设定应用于域传输:

also-notify

定义一个用于全局的域名服务器 IP 地址列表。无论何时，当一个新的域文件被调入系统，域名服务器都会向这些地址，还有这些域中的 NS 记录发送 NOTIFY 信息。这有助于更新的域文件集快在相关的域名服务器上收敛同步。如果一个 also-notify 列表配置在一个 zone 语句中，全局 options 中的 also-notify 语句就会在这里失效。当一个 zone-notify 语句被设定为 no，系统就不会向在全局中 also-notify 列表中的 IP 地址发送 NOTIFY 消息。缺省状态为空表(没有全局通知列表)。

max-transfer-time-in

比设定时间更长的进入的域传输将会被终止。默认值是 120 分钟(2 小时)。

max-transfer-idle-in

在设定时间下没有任何进展的进入域传输将会被终止。默认为 60 分钟(1 小时)。

max-transfer-time-out

运行时间比设定的时间长的发出的域传输将会被终止。默认为 120 分钟(2 小时).

max-transfer-idle-out

在设定时间下没有任何进展的发出的域传输将会被终止。默认为 60 分钟(1 小时)。

serial-query-rate

辅域名服务器将会定时查询主域名服务器，来确定域的串号是否改变。每个查询将会占用一些辅域名服务器网络带宽。为限制占用的带宽，BIND9 可以限制每个查询发送的频率。serial-query-rate 的值是一个整数，就是每秒能发送的最大查询数。默认值为 20。

Serial-queries

在 BIND8 中, serial-queries 选项设定了在任何时候允许达到的最大的并发查询数。BIND9 不限制串号查询的数量并忽略了 serial-queries 选项。它会使用 serial-query-rate 选项来限制查询的频率。

transfer-format

域传输可以用两种不同格式，one-answer 和 many-answer。transfer-format 选项使用在主域名服务器上，用来确定发送哪种格式。one-answer 在每个资源记录传输中使用一个 DNS 消息。many-answer 则将尽可能多的资源记录集中在一个消息中。many-answer 是更加有效的，但只有相对比较新的辅域名服务器才支持它，如 BIND9、BIND8.x 和打

了补丁的 BIND4.9.5。默认的设置为 many-answer。使用 server 语句中的相关选项，可以替代全局选项中的 transfer-format 设置。

transfers-in

可以同时运行的进入的域传输的最大值。默认值为 10。增加 transfers-in 的值，可以加速辅域的收敛速度，但也可能增加本地系统的负载。

transfers-out

可以同时运行的发出的传输的最大值。超过限定的域传输请求将会被拒绝。默认值为 10。

transfers-per-ns

从一台指定的远程域名服务器，同时进行的进入的域传输的最大值。默认值 2。增加 transfers-per-ns 的值，会加速辅域的收敛速度，但也可能增加远程系统的负载。使用 server 语句中的 transfer 短语可以替代全局选项中的 transfers-per-ns。

transfer-source

transfer-source 决定在从外部域名服务器上得到域传送数据时，选哪个本地的 ip 地址使用在 IPV4 的 TCP 连接中。它可以选定 IPV4 的源地址，和可选的 UDP 端口，用于更新的查询和转发的动态更新。不过不做设置，它会缺省挑选一个系统中的地址(常常是最靠近远终端服务器的接口地址)。但这个地址必须已经配置在远终端的 allow-tranfer 选项中，才能进行域传送。此语句为所有的域设定了 transfer-source，但如果 view 或 zone 中也使用了 transfer-source 语句，则全局选项中的配置就在这里失效了。

transfer-source-v6

和 transfer-source 一样，只是域传输是通过 IPV6 执行的。

notify-source

notify-source 确定使用哪些本地的源地址和可选的 UDP 端口，用于发送 NOTIFY 消息。这个地址必须在辅域名服务器的 master 域或在 allow-notify 中设置。它会为所有域设定 notify-source, 但如果 view 或 zone 中也使用了 notify-source 语句，则全局选项中的配置就在这里失效了。

notify-source-v6

与 notify-source 类似，但应用于 ipv6 地址的 notify 报文的发送。

7.2.14.7 操作系统资源限制

可以限制服务器对许多系统资源的使用。这些就是通过调节资源限制的数值来完成的。例如，1G 可以代替 1073741824，限定一个十亿字节的限制。Unlimited 要求不限制使用，或者最大可用量。Default 将会使用服务器启动时的缺省值。
下列选项设定了域名服务器进程的操作系统资源占用限制。一些操作系统可能不支持一些或所有的限制。在这样的系统中，当使用不被支持的限制时，会产生一个告警。

coresize

core dump 文件的最大值尺寸。默认值为 default

datasize

服务器可以使用的最大数据内存量。默认值为 default。这是一个在服务器系统内存中已经设置了的参数。如果服务器要超过这个限制的内，则会失败，这将使服务器不能提供 DNS 服务。所以，这个选项作为一种限制服务器所使用的内存量的方式就不太有效，但是它能够将操作系统设置的太小的缺省数据尺寸增大。如果要限制服务器使用的内存量，可以使用 max-cache-size 和 recursive-clients 选项。

files

服务器可以同时打开的最大文件数。默认是 unlimited。

stacksize

服务器可以使用最大的堆栈内存量。默认值为 default.

7.2.14.8 服务器资源限制

下列选项设定了服务器资源使用限制，这是由域名服务内部做的而不是操作系统设定的。

max-ixfr-log-size

此选项比较老；它由 BIND8 兼容接受或者忽略。

recursive-clients

服务起同时为用户执行的递归查询的最大数量。默认值 1000，因为每个递归用户使用许多位内存，一般为 20KB，主机上的 recursive-clients 选项值必须根据实际内存大小调整。

tcp-clients

服务器同时接受的 TCP 连接的最大数量，默认值 100。

max-cache-size

服务器缓冲使用的最大内存量，用比特表示。但在缓存数据的量达到这个界限，服务器将会使记录提早过期这样限制就不会被突破。在多视图的服务器中，限制分别使用于每个视图的缓存。默认值没有限制，意味着只有当总的限制被突破的时候记录才会被缓存清除。

7.2.14.9 周期性任务间隔

cleaning-interval

服务器将在 cleaning-interval 的每一时间中从缓存中清除过期的资源记录。默认为 60 分钟，如果设置为 0，就不会有周期性清理。

heartbeat-interval

服务器将会为所有标记 dialup 的域运行维护任务，无论它的间隔在何时到期。默认为 60 分钟，合理值不超过 1 天(1440 分钟)。如果设定为 0,不会为这些域产生域维护。

interface-interval

服务器将在每个 interface-interval 时间扫描网络接口表。默认为 60 分钟。如果设置为 0，仅当配置文件被加载时才会进行接口扫描。在扫描之后，所有新接口上的监听器将会被打开(listen-on 配置使用的接口)。关闭接口上的监听器将会被清除.

statistics-interval

域名服务器统计将会在每个 statistics-interval 时刻被记入日志。默认值 60 分钟，如果设为 0， 就没有统计数据记入日志。

注意:BIND9 不支持

7.2.14.10 拓扑

当服务器从一个域名服务器列表中选择一个域名服务器查询时，这些域名服务器是没有什么不同的，但是服务器会有先选择在托扑结构上距离自己最近的服务器去做解析。拓扑语句使用一个地址匹配列表并且以一个特殊方式解释它。每个顶层表列元素被赋了一段距离，非否定元素得到它们在表列中的位置的距离，匹配距离表的开头越近，它何服务器的距离就越小。否定匹配元素将会从服务器分配最大距离；没有匹配的地址将会得到一个比任何非否定表元素都远的并且比任何否定元素近的距离。例如：

```
topology {
10/8;
!1.2.3/24;
{ 1.2/16; 3/8; };
};
```

最优先网段 10 的服务器，然后是在网络 1.2.0.0(网络掩码 255.255.0.0)和 3.0.0.0(网络掩码 255.0.0.0)；再就是没列出来的，但是没有否定的网段。否定的网段 1.2.3 的主机(网络掩码 255.255.255.0)。

默认拓扑为:

```
topology { localhost; localnets; };
```

注意：BIND9 不支持拓扑选项。

7.2.14.11 sortlist 语句

对一个 DNS 询问的响应包括形成一个资源记录集(RR 集)的多资源记录(RRs)。名称服务器将会以不确定的顺序返回在 RRset 中的 RRs(参见 7.2.14.12 节中的 rrset-order 语句)。用户端的解答器会重新适当的排列，也就是说，使用任何在本地网上的地址优先于其他地址的地址。尽管如此，不是所有的解答器可以做到或者正确配置。当用户使用一个本地服务器的时候，服务器可以基于用户地址进行分类。这只要求配置名称服务器，而不是所有用户端。

Sortlist 语句(如下)使用一个地址匹配表甚至比拓扑语句(7.2.14.10 节)还要特殊的解释它。每个在 sortlist 的顶层语句必须自己就是一个清楚的拥有一个或两个元素的地址匹配表。每个顶级表的第一个元素(可能是一个 IP 地址，一个 IP 前缀，一个 ACL 名称或者一个地址匹配表)与查询源地址进行匹配检查直到找到匹配的地址。

一旦查询的源地址被匹配，如果顶级语句只包括一个元素的话，真正的匹配于源地址的原始元素就被用来选择地址，对应的转移到了响应的开始。如果语句是两个元素的表，那么第二个元素遵照拓扑语句中地址匹配表的方式进行处理。每个顶级元素被赋予一个距离和与响应的开头距离最近的地址。

在下列例子中，任何来自于任何主机地址的的查询将会得到本地网上第一首选地址的响应。下一个首选地址在网段 192.168.1/24 上，既可以在 192.168.2/24 或 192.168.3/24 网段之后。

从一台在 192.168.1/24 网段上的主机收到的查询将会优先本网段和 192.168.2/24 和 192.168.3/24.网。而来自 192.168.4/24 或 192.168.5/24 上主机的查询将只优先直连的网段。

```
sortlist {
{ localhost;                                    //IF   主机名
{ localnets;
192.168.1/24;                                   //THEN 在下列网中最适合
{ 192.168.2/24; 192.168.3/24; }; }; };          //IF 在 C 类 192.168.1
{ 192.168.1/24;                                 //THEN 使用.1, 或 .2 或 .3
{ 192.168.2/24; 192.168.3/24; }; }; };
{ 192.168.2/24;                                 //IF   C 类   192.168.1
{ 192.168.2/24;                                 //THEN 使用 2, 或 .1 或 .3
{ 192.168.1/24; 192.168.3/24; }; }; };
{ 192.168.3/24;                                 //IF   在 C 类 192.168.3
{ 192.168.1/24; 192.168.2/24; }; }; };          //THEN 使用 .3 或.1 或.2
// if .4 or .5, pre- { { 192.168.4/24; 192.168.5/24; };   //如果    .4 或者.5
fer that net
};
};
```

下个例子将给出一个本地主机和直接连接到网上的主机的合理的状态(behavior)。它很象 BIND4.9.x 分类的地址状态。从本地主机发给查询的响应支持任何直接连接的网络，从其他直接连接网络上的主机发送给查询的响应优先在相同网段上的地址。对其他查询的响应没有分类。

```
sortlist {
    { localhost; localnets; };
    { localnets; };
};
```

7.2.14.12 RRset 排序

当多重记录在一个解答中被返回的时候，设定在响应中的记录的顺序是很有用的.。rrset-order 语句允许对在多记录响应下的记录的顺序的设定。参见 7.2.14.11 节的 sortlist 语句。

一个 order_spec 定义如下：

```
[ class class_name ][ type type_name ][ name "domain_name"] order ordering
```

如果没有设定类，默认值为 ANY。如果没有设定类型，默认值为 ANY。如果没有设定名称，默认值为”*”。

合法的排序值是：

fixed	记录以它们在域文件中的顺序排序
random	记录以随机顺序被返回
cyclic	记录以环顺序被返回

例如：

```
rrset-order {
class IN type A name "host.example.com" order random;
order cyclic;
};
```

将会使得任何处于 IN 类中的 A 类记录的响应以随机顺序返回，IN 类以"host.example.com"为后缀。其他的记录以循环记录被返回。

如果多重 rrset-order 语句出现，它们并不组合在一起，只适用于最后一个条。

注意：rrset-order 语句不被 BIND9 支持，BIND9 目前只支持"random-cyclic"排序，服务器随机选择 RRset 集中的开始点，有顺序返回在那个点开始的记录。如果需要的话围绕 RRset 结尾。

7.2.14.13 合成的 IPV6 响应

许多现存的子域解答器支持 ipv6 的 DNS 查询(定义在 RFC1986 中，使用 AAAA 记录进行前向查询和 ip6.int 域中的"nibble labels"进行反向查询)但是不支持 RFC2874-style 查询(使用 A6 记录和在 ip6.arpa 中的二进制标签)对于那些希望继续使用子域解答器而不是转到 BIND9 lightweight 解答器的人来说， BIND 9 提供一种自动把 RFC1886-型查询转换成 RFC2874-型查询的方法。返回合成的 AAAA 和 PTR 记录。

这个性质默认下是无效的，可以在分用户基础上添加一个 ***allow-v6-synthesis { address_match_list };***子句到到选项或者视图语句中。当它被激活时，递归 AAAA 查询使服务器先进行 A6 查询，如果失败，执行 AAAA 查询。不管哪个成功，结果都作为一个合成的 AAAA 记录返回。

类似的，在 ip6.int 中的递归 PTR 查询将会促使一个 ip6.arpa 查询使用二进制标签，如果失败，执行另一个在 ip6.int 中的查询，结果将会以在 ip6.int 中的合成 PTR 记录返回。

合成记录的 TTL 为 0 值。合成响应的 DNSSEC 确认当前并不被支持；也没有了 AD 标记。

注：allow-v6-synthesis 仅为提供了递归服务的用户执行.

7.2.14.14 调谐

lame-ttl

设定缓存有问题服务器指示的秒数。0 使不缓存(不被推荐)。默认值 600(10 分钟)。最大值 1800(30 分钟)。

max-ncache-ttl

为降低网络流量和提升服务器存储否定回答的性能。 max-ncache-ttl 以秒为单位设定这些回答的保存时间.默认 max-ncache-ttl 是 10800 秒(3 小时)。 max-ncache-ttl 不能超过 7 天，如果设成一个更大的值，则将会被自动减为 7 天。

max-cache-ttl

max-cache-ttl 设定了服务器储存普通(肯定)答案的最大时间。默认值一周(7 天)

min-roots

一个请求要求的最小的根服务器数量。默认为 2。

注意:不被 BIND9 支持

sig-validity-interval

设定未来作为动态更新结果的自动生成的 DNSSEC 信号过期的天数。默认是 30 天。信号的初始时间无条件设为在当前时间的前一个小时，以允许一个有限的时钟偏差。

min-refresh-time
max-refresh-time
min-retry-time
max-retry-time

这些选项控制了服务器在更新一个域(询问 SOA 变化)或者重试失败的传输时的状态。通常域的 SOA 值(但是这些值是由主服务器设定的)几乎不给此级服务器管理者对它们内容的控制。

这些选项允许管理者为每域，每个视图或者全局设定一个最小或者最大更新和重试时间。这些选项对于此级和根域是有效的并且设定 SOA 更新和重试时间。

7.2.14.15 统计文件

由 BIND9 产生的统计文件和由 BIND8 产生的类似，但不完全一样。

一个统计数据开始于行+++ Statistics Dump +++ (973798949)，这里出现的数字是一个标准 UNIX 型的时间戳，从 1970 年 1 月 1 日开始以秒计。紧跟这行的是一系列行，包括一个记数器类型，记数器值，任意的域名和任意的视图名，没有所列的视图和域的行是整个服务器的整体统计。具有域和视图的行以给定的视图和域命名(对默认的视图来说视图名缺省)。

这个统计数据以行--- Statistics Dump ---(973798949)结束，在这数字是和开始行的数字一样的。Success 对服务器或者域做出的成功查询。定义一个成功查询是，查询返回非错误响应而不是返回推荐响应

Referral	导致推荐响应查询
Nxrrset	导致没有数据的非错误查询的响应
Nxdomain	导致 NXDOMAIN 的查询数量
Recursion	使服务器运行递归以找出最后答案的查询数量
Failure	导致失败的查询数量

7.2.15 服务器语句语法

```
server ip_addr {
[ bogus yes_or_no ; ]
[ provide-ixfr yes_or_no ; ]
[ request-ixfr yes_or_no ; ]
[ edns yes_or_no ; ]
[ transfers number ; ]
[ transfer-format ( one-answer | many-answers ) ; ]]
[ keys { string ; [ string ; [...]] } ; ]
};
```

7.2.16 服务器语句定义和使用

服务器语句定义了与远程服务器相关的性质

服务器语句可以出现在配置文件的顶层或者在视图语句的内部。如果一个视图语句包括一个或更多的服务器语句，只有那些视图语句内的被应用到视图和任何顶层的语句被忽略。如果一个视图不包括服务器语句，则任何顶层服务器语句都被当做默认。

如果你发现 一台远端服务器正在输出错误数据，使用 bogus 标记它，将会阻止对它的更多查询.。Bogus 的默认值为 no。

provide-ixfr 子句决定本地服务器是否作为主域名服务器，当远端的一台辅域名服务器要求的时候，响应增量的域传输。如果设定成 yes，增量传输将会在任何可能的时候进行。如果设定为 no，所有对远端服务器的传输都将是非增量的。如果不设置，在视图或者全局选项块中的 provide-ixfr 选项的值将使用默认值。

Request-ixfr 子句决定本地服务器是否作为辅域名服务器，将会向给定的远端服务器（一般为主域名服务器）发送域的增量传送请求。如果不设置，在视图或者全局选项块中的 provide-ixfr 选项的值将使用默认值。

对不支持 IXFR 的服务器的 IXFR 请求将会自动转回 AXFR。这样，就不需要用手工列出那个服务器支持 IXFR 哪个不支持。全局默认值 yes 一直会起作用。provide-ixfr 和 request-ixfr 字句的作用是使 IXFR 无效，甚至当主域名服务器和辅域名服务器宣布支持它。例如当使用 IXFR 时，如果一台服务器受灾并且崩溃或者破坏了数据。

Edns 子句决定本地服务器与远端服务器通讯时是否使用 EDNS.默认值为 yes。

服务器支持两类域传输方式：

第一类，**one-answer**，每个源数据传输使用一个 DNS 报文。

第二类，**many-answer**，在一个报文中汇集了尽可能多的记录。多答案更有效率，但是只被 BIND9，BIND8.X 和 BIND4.9.5 补丁版本识别。你可以使用 transfer-format 选项为一个服务器指定使用哪种方式。如果 transfer-format 没有被指定，就使用 **options** 里的语句指定的 transfer-format。

Transfers

用来限定同时从特定服务器进行数据传的域的数量。如果 transfers 子句没有被指定，限制将会参照 transfers-per-ns 制定。

Keys

子句用来确定一个由 key 语句定义的 **key_id**，用于当于远端服务器通话时的安全处理。key 语句必须在涉及它的服务器语句之前。当一个请求发送到远端服务器，一则请求报文将会产生(使用本地指定键并附在报文上)。一个来自远端服务器的请求不要求被这个键标记。

尽管键子句语法支持多键,但是目前每个服务器只支持一个键.

7.2.17 trusted-keys 语句语法

trusted-keys {
string number number number string ;
[string number number number string ; [...]]
};

7.2.18 trusted-keys 语句定义和使用

trusted-keys 语句定义 DNSSEC 安全根。DNSSEC 的描述在 5.7 节中。当非授权域的公共键是已知的但是却不能安全的通过 DNS 得到，或者因为 DNS 根域或者它的当前域没有被标记时定义安全根。一旦一个键被设置成信任键，它就被认为是有效和安全的。解答器在所有存在于安全根次级域中的 DNS 数据上尝试 DNSSEC 有效性。

trusted-keys 语句能包含多重键入口，每个由键的域名，旗帜，协议算法和键数据的 64 进位组成。

7.2.19 视图语句语法

view view_name [class] {
match-clients { address_match_list } ;
match-destinations { address_match_list } ;
match-recursive-only { yes_or_no } ;
[view_option; ...]
[zone-statistics yes_or_no ;]
[zone_statement; ...]
};

7.2.20 视图语句定义和使用

视图是 BIND9 的强大的新功能，允许名称服务器根据询问者的不同有区别的回答 DNS 查询。特别是当运行拆分 DNS 设置而不需要运行多个服务器时特别有用。

每个视图定义了一个将会在用户的子集中见到的 DNS 名称空间。

根据用户的源地址(“address_match_list”)匹配视图定义的“match_clients”和用户的目的地址(“address_match_list”)匹配视图定义的” matach-destinations”。

如果没有被指定,，match-clients 和 match-destinations 默认于匹配所有地址。一个视图也可以做为 match-recursive-only 来指定，意思是来自匹配用户的递归请求将会匹配该视图。视图语句的顺序是很重要的，一位用户的请求将会在它所匹配的第一个视图中被解答。在视图语句中定义的域只对匹配视图的用户是可用的。通过在多个视图中用相同名称定义一

个域，不同域数据可以传给不同的用户。例如：在拆分 DNS 设置中的”内部”和”外部”用户。

许多在 named.conf 的 options 里的语句中给出的选项也能在视图语句中使用，仅仅用于使用那个视图解答请求的时候。当 view-specific 值没有给出，options 里的语句值就使用默认值。域选项也可以在视图语句中指定默认值；这些 view-specific 默认值的优先级高于那些在 options 里面配置的语句。

视图精确到类。如果没有给定任何类，就假设为 IN 类。注意到所有非-IN 视图必须包含一个暗示域，因为只有 IN 类具有 compiled-in 默认暗示。

如果在配置文件中没有 **view** 语句，在 IN 类中就会自动产生一个默认视图匹配于任何用户，任何指定在配置文件的最高级的 zone 语句被看作是此默认视图的一部分。如果存在外部 **view** 语句，所有的域视图必须会在 **view** 语句内部产生。

这是一则典型的使用视图语句运行的拆分 DNS 设置

```
view "internal" {
match-clients { 10.0.0.0/8; };
// 应该与内部网络匹配.
// 只对内部用户提供递归服务.
// 提供 example.com zone 的完全视图
//包括内部主机地址.
recursion yes;
zone "example.com" {
type master;
file "example-internal.db";
};
};

view "external" {
match-clients { any; };
// 拒绝对外部用户提供递归服务
// 提供一个 example.com zone 的受限视图
// 只包括公共可接入主机
recursion no;
zone "example.com" {
type master;
file "example-external.db";
};
};
```

7.2.21 zone 语句语法

```
zone zone_name [class] [{
type ( master | slave | hint | stub | forward ) ;
[ allow-notify { address_match_list } ; ]
```

[allow-query { address_match_list } ;]
[allow-transfer { address_match_list } ;]
[allow-update { address_match_list } ;]
[update-policy { update_policy_rule [...] } ;]
[allow-update-forwarding { address_match_list } ;]
[alsonotify
{ ip_addr [port ip_port] ; [ip_addr [port ip_port] ; ...] };]
[check-names (warn|fail|ignore) ;]
[dialup dialup_option ;]
[file string ;]
[forward (only|first) ;]
[forwarders
{ ip_addr [port ip_port] ; [ip_addr [port ip_port] ; ...] };]
[ixfr-base string ;]
[ixfr-tmp-file string ;]
[maintain-ixfr-base yes_or_no ;]
[masters [port ip_port] { ip_addr [port ip_port] [key key]; [...] } ;]
[max-ixfr-log-size number ;]
[max-transfer-idle-in number ;]
[max-transfer-idle-out number ;]
[max-transfer-time-in number ;]
[max-transfer-time-out number ;]
[notify yes_or_no | explicit ;]
[pubkey number number number string ;]
[transfer-source (ip4_addr | *) [port ip_port] ;]
[transfer-source-v6 (ip6_addr | *) [port ip_port] ;]
[notify-source (ip4_addr | *) [port ip_port] ;]
[notify-source-v6 (ip6_addr | *) [port ip_port] ;]
[zone-statistics yes_or_no ;]
[sig-validity-interval number ;]
[database string ;]
[min-refresh-time number ;]
[max-refresh-time number ;]
[min-retry-time number ;]
[max-retry-time number ;]
}];

7.2.22 zone 语句定义和使用

7.2.22.1 域文件类型

master

服务器有一个主域（控制域或主域）的配置文件拷贝，能够为之提供授权解析服务。

Slave

辅域（也可以叫次级域）是主域的复制。主域名服务器定义了一个辅域或多个辅域(次级域联系以更新域拷贝)IP 地址.默认下，传输是从服务器上的 53 端口进行的；对所有的服务器来说这是可变的，通过设定一个在 IP 地址表前或者在 IP 地址之后基于每个服务器设定端口数字。对主域名服务器的鉴别也能通过每个服务器上的 TSIG 键来完成。如果文件被指定了，那么任何主域配置信息改变的时候就要复制文件，并且当辅服务器重新启动的时候都会从主域名服务器上重新下载文件。这可能会导致带宽的浪费和服务器重新启动次数的增加。

注意对每个服务器的数量众多的域来说(数万或者数十万)，最好使用两级方式命名配置。

例如：一个域的服务器 example.com 可能把域内容放到一个叫做 ex/example.com 的文件中，在此, ex/只是域名前两个字符(如果把 100K 的文件放入一个单独的目录中，大多数操作系统都会反应缓慢)

stub

子根域与辅域类似,除了只复制主域的 NS 记录而不是整个域。根域不是 DNS 的一个标准部分，它们是 BIND 运行的特有性质。

根域可以用来避免在本机重新获得该域的 NS 记录，代价是保存一个根域入口和一组 named.conf 名称服务器地址。这个用法在新设置中并不建议使用，BIND9 只在有限的情况下才支持它。在 BIND4/8 中当前的域传输包括来自当前域的子根的 NS 记录。这表明，在某些情况下，用户可以为当前域设置只存在于控制服务器里的子根.。BIND9 服务器从不以这种方式把来自不同域的数据混合。这样的话，如果一个 BIND9 控制服务器服务于一个已经设定了子根域的母域，所有的当前域的次级服务器都需要设定相同的子根域。子根域也可以用来作为一种促使一个特定域的解答使用一个授权服务器的特定系。例如,在一个使用 RFC2157 地址的私用网络上缓存名服务器可以用子根域进行设置.

forward

一个”转发域”是一种在每个域基础上进行配置转发的一种方式。forward 类型的域语句包括一个转发语句和转发列表，都应用于在域内的由域名给出的查询。如果当前没有转发器语句，就会给出空列表，在域中就不会转发，也就取消了所有在选项中的转发的作用。如果你要使用此种域来改变整体转发选项的性态(”forward first”，”forward only”但是要用同一服务器作为是全局设置)你需要理解全局转发器的特点。

hint

根名称服务器的在最初设置时指定使用一个”hint zone”。当配置了”hint zone”的服务器启动的时候，它使用根线索的设置找到根的名称服务器并得到根名称服务器的最新表。如果没有为 IN 类设定线索域，服务器使用一个 compiled-in 的默认根服务器列表。

7.2.22.2 类

域名后面的选项可以对应类。如果没有指定类，系统假定为 IN 类。这在大多数的情况下都是正确的。

Hesiod 类是以一个信息服务的名称命名的，信息服务源于 MIT 的 Athena 工程.。Hesiod 类是用来在多系统数据之间共享信息，如用户、组、打印机等等。”HS”对 Hesiod 来说是相同的类。

MIT 发展的另一个是 CHAOSnet，一个在 70 年代中期创立的本地协议。它的域数据可以用 CHAOS 类设定。

7.2.22.3 zone 选项

allow-notify

参见 7.2.14.3 节关于 allow-notify 的描述.

allow-query

参见 7.2.14.3 节关于 allow-query 的描述.

allow-transfer

参见 7.2.14.3 节关于 allow-transfer 的描述.

allow-update

设定哪台主机允许为主域名服务器提交动态 DNS 更新。默认为拒绝任何主机进行更新。

update-policy

设定一个”简单安全更新政策”。参见 7.2.22.4 节.

allow-update-forwarding

设定哪个主机能够向辅域名服务器的次级域提交动态域名服务器更新。默认值为{ none; }，意味着不能进行动态更新转发。要使用更新转发，设定 allow-update-forwarding { any; }；设定其他值而不是{ none; } 或者{ any; } 是常常起反作用的，因为主域名服务器拥有接入控制的权利，而辅域名服务器没有。

注意到激活一台次级服务器上的更新转发性质可能使主域名服务器依赖基于不安全 IP 地址的接入控制而可能使之受到攻击。

详细信息请参见 8.3 节

also-notify

只当本域 notify 被激活时才是有意义的。能够收到本域 DNS NOTIFY 信息的计算机的集合是由所有域中列明的名称服务器加上任何由 also-notify 设定的 IP 地址。一个端口可能用每个 also-notify 地址设定以发送通告信息到那个端口而不是默认的 53 端口.。also-notify 对根域是无意义的。默认值为空。

check-names

此选项应用于 BIND8 中以约束 hosts 文件中的域名的字符集和/或从网络上接收的 DNS 响应。BIND9 不限制域名的字符集也不执行 check-names 选项。

database

设定储存域数据的数据库的类型。 Database 后面的关键词是一列 whitespace-delimited(非限制空白空间)词。第一个词定义数据库类型，后续词作为数据库的参数传给数据库，解释为特殊的数据库类型。默认值为"rbt",BIND9 的本地 native in-memory red-black-tree 库，则此数据库没有参数如果其它数据库驱动连接到服务器上的话其他值也是可能的。这些可能的数据库包括分布式的数据库，但缺省是没有连接的。

Dialup

参见 7.2.14.1 节关于 dialup 的描述.

forward

只当域有一个转发器列表的时候才是有意义的。当配置为”only”时，在转发查询失败和的不到结果时会导致查询失败；在配置为”first”，则在转发查询失败或没有查到结果时，会在本地发起正常查询。

forwarders

用来代替全局的转发器列表。如果如故不在 forward 类型的域中设定，就不会有这个域查询的被转发；全局的转发设置则没有起作用。

ixfr-base

在 BIND8 中设定动态更新和 IXFR 的交易日志文件（journal）的名称。BIND9 忽略这个选项而通过附加".jnl"到域文件名后创建日志文件.

ixfr-tmp-file

BIND8 的非正式选项.BIND9 忽略

max-transfer-time-in

参见 7.2.14.6.节 max-transfer-time-in 的描述.

max-transfer-idle-in

参见 7.2.14.6.节 max-transfer-idle-in 的描述.

notify

参见 7.2.14.1.节 notify 的描述.

pubkey

在 BIND8 中，此选项用以为 DNSSEC 标记域的信号(当它们从磁盘装载的时候)的验证设定一个公共域密钥，BIND9 不在装载的时候验证密钥并忽略此选项。

zone-statistics

如果设为”yes”，服务器将会为本域储存统计信息，可以存储到在服务器选项中定义的统计文件中。

sig-validity-interval

参见 7.2.14.14.节 sig-validity-interval 的描述.

transfer-source

参见 7.2.14.6.节 transfer-source 的描述.

transfer-source-v6

参见 7.2.14.6.节 transfer-source-v6 的描述.

notify-source

参见 7.2.14.6.节 notify-source 的描述.

notify-source-v6

参见 7.2.14.6.节 notify-source-v6 的描述

min-refresh-time
max-refresh-time
min-retry-time
max-retry-time

参见 7.2.14.14.节的描述.

7.2.22.4 动态更新政策

BIND9 支持两个授予用户对一个域执行动态更新权限的备选方案，分别由 allow-update 和 update-policy 设定。

allow-update 的使用与以前的 BIND 版本相同。它授于指定用户对域中的任何名称的任何的记录更新的权利。

update-policy 是 BIND9 中新出现的，允许更多更新控制，定义了一个规则集，规则授予或者取消一个或多个名称被一个或多个用户更新的权利。如果动态更新要求信息被标记(也就是说,可以包含 TSIG 或 SIG(0)记录)，标记人的身份就能被确定。

规则在 update-policy 域选项中指定，并只对主域有意义。当 update-policy 语句出现，这是一个 allow-update 的配置错误.。update-policy 语句只检查信息的签名人，与源地址是不相关的。

这是一则规则的定义：

(grant | deny) identity nametype name [types]

每条规则赋予或者取消授权，一旦一条信息成功的匹配一个规则，则马上执行该规则

的给予或者取消操作，并且不检查其它的规则。一个规则是匹配的，就是当标记人匹配身份字段，名称匹配名称字段并且类型是在类型字段中定义的时候。

身份字段定义一个名称或者一个通配符名称，名称类型字段有四个值：

name, subdomain, wildcard 和 self

Name

当更新名称与原定义的名称字段相同时匹配。

Subdomain

当更新名称与原定义的子域的一个名称字段相同时匹配。(包含它本身的名字)

Wildcard

当更新名称与原定义的一个位于名称字段中的统配符名称的有效延伸相同时匹配

self

当更新名称与信息标记人的名称相同时匹配。忽略名称字段。

如果没有设定任何类型，规则匹配所有类型除了 SIG、NS、SOA 和 NXT；类型可能用名称设定，包括"ANY"。(ANY 匹配所有类型除了 NXT，NXT 不能被更新)。

7.3 域文件

7.3.1 资源记录类型及使用

本节(主要截取至 RFC 1034),描述了资源记录(RR)的概念和它们的使用方法.从 RFC1034 发行开始, 在 DNS 中定义和使用了若干新 RRs 类型.这些也被包括进来了.

7.3.1.1 资源记录

一个域名定义了域名树中的一个节点.每个节点都有一个资源信息集,可以为空.有特定名称资源信息集域有独立的 RRs 组成.在集中 RRs 的顺序是不重要的,也不需要由名称服务器,解答器或者其他 DNS 部分储存.

但是,为优化目的对多重 RRs 设置顺序是允许的.例如,设定首先尝试一个特定的邻近服务器.参见 7.2.14.11 节 7.2.14.12 节.

资源记录（RR）组成：

Owner name	所有者名称，指定域名对应记录的位置
Type	一个 16 位编码的值用来设定这个源记录中的源的类型.类型涉及到抽象记录
TTL	定义 RR 记录的生存时间.这个字段是一个以秒计算 32 位整数,主要设置该记录在缓存里的保留时间.
Class	一个 16 位编码值定义一组协议或者一协议示例

RDATA	描述源头的类型和独立类的数据.

下列是有效 RRS 的类型(其中列出的一些,尽管没有过时,是实验性的(X)和历史的(X)并且不再被经常使用了)

A	一个主机地址
A6	一个 IPV6 地址
AAAA	过时的 IPV6 地址格式
AFSDB	(x)AFS 数据库服务器位置.实验性的
CERT	保持数字验证
CNAME	定义规范的别名
DNAME	用于授权反查地址.使用另一个定义的查询名称取代域名.参见 RFC2672
GPOS	设定整体位置.由通信线路代替
HINFO	设定一台主机使用的 CPU 和 OS(操作系统)
ISDN	(x)ISDN 地址(综合数字业务网)的表示.实验性的
KEY	储存一个关于 DNS 名称的公共键
KX	定义一个 DNS 名称交换器
LOC	(X)储存 GPS 信息.参见 RFC 1876.是实验性的
MX	设定域邮件交换.参见 See RFC 974 for
NAPTR	名称授权指针
NSAP	网络服务接入指针
NS	域的授权名称服务器
NXT	用在 DNSSEC 中以安全的指出 RRS(有一个特定名称域间内的系主名)不在域中
PTR	指向其他域名空间部分的指针
PX	在 RFC 822 和 X.400 间提供映射.
RP	(X)域管理者的信息
RT	(X) 主机的 route-through,主机没有它们自己的直接的广域网地址.实验性的.
SIG	(“签名机制”)在域名安全中的认证信息.细节见 RFC 2535
SOA	设定授权域的起点
SRV	关于熟知的网络服务信息(代替 WKS)
TXT	文本记录
WKS	(h)域支持的熟知网络服务信息,如 SMTP.历史的,由新的 RR SRV 替代
X25	(x)x.25 网络地址代表.实验性的

下列源记录类目前 DNS 中是有效的

IN	因特网系统
RDATA	是描述资源 type-dependent 或者 class-dependent 数据:
A	用于 IN 类,一个 32 位 IP 地址
A6	为一个 IPV6 地址绘制域名,关于主要前缀位的预防
CNAME	域名，或是别名

DNAME	提供对域名空间的整个子树的交替命名,而不是对一个单模式.它使得一些查询名称的后缀被来自 DNAME 记录的 RDATA 名称代替
MX	一个 16 位优先值(越小越好),后接一个主机名,作为所有者域名的邮件交换
NS	完全合格的域名
PTR	一个完全合格的域名，反向记录
SOA	若干字段

所有者名称常常是固定的,而不是形成一个 RR 整数部分.例如,许多名称服务器的域名空间在内部形成树或者复杂树结构和复杂树结构外的节点。剩下的 RR 部分是和 RRs 保持一致的固定的首部 (类型,类,TTL)和一个适合资源需要的变量部分（RDATA）。

TTL 字段的意思是一个 RR 在缓存中保存多长时间的限制。这个限制不用于域中的授权数据；但是通过域更新政策它也可以有效。TTL 是由数据起始的域的管理者分配的。短 TTLS 可以用来最小化缓存，0 值 TTL 禁止缓冲，因特网性能的事实表明对一台典型的主机来说这些时间可以一天的顺序.如果可以预计一个变化，TTL 可以在变化之前最小化，在变化后回恢复原来的数值，或者以原来的值变化。

在 RDATA 节的 RRs 数据被作为是二进制字符串和域名的组合。域名常常被用作指向 DNS 中其他数据的指针。

7.3.1.2 RRs 的原文表达

RRs 在 DNS 协议信息包中以二进制形式表示，并当储存在一个名称服务起或者解答器中时常常以更高的编码形式表示。在 RFC1034 的例子中，与在主域 hosts 文件中相似的类型被引进以展示 RRS 的内容。在这个格式中，大多数 RRS 出现在单行中，尽管使用圆括号的持续性文件是可行的。

每行起始时给出了 RR 的所有者。如果行以空白符开始，那么所有者被认为是与前一个 RR 相同。空白行通常是为了提高可读性。

在所有者之后,我们列出了 RR 的 TTL，类型和类。类和类型使用上述定义的记忆，TTL 是一个在类型字段之前的整数。类型和类记忆是不分开的，TTL 都是整数，类型记忆常常在最后。IN 类和 TTL 类常常在关注清晰度的例子中漏掉。

资源数据或者 RR 的 RDATA 部分被假定使用数据的典型表示法知识
例如，我们可以列出一则信息带的 RRs：

```
ISI.EDU.            MX      10 VENERA.ISI.EDU.
                    MX      10 VAXA.ISI.EDU
VENERA.ISI.EDU      A       128.9.0.32
                    A       10.1.0.52
VAXA.ISI.EDU        A       10.2.0.27
                    A       128.9.0.33
```

MX RRS 由一个 RDATA 部分，由一个 16 位数组成，后接里一个域名。RRS 域名使用一个标准 IP 地址格式以包含一个 32 位因特网地址。
这个例子显示个 6 个 RRS，包括两个 RRS 上的三个域名

相似的我们可能见到：

```
XX.LCS.MIT.EDU.    IN    A                    10.0.0.44
CH                       A                    MIT.EDU. 2420
```

这个例子展示了两个 XX.LCS.MIT.EDU 地址，每个在有一个不同的类。

7.3.2 MX 记录的讨论

如上所述,域服务器储存了一系列资源记录的源数据，每个包括关于给定域名的详细信息(常常但不一定是主机)。最简单的方法是把 RR 想象成为一对标准信息，一个域名，和与其匹配的相关资料以及存储一些额外类型信息，以帮助系统确定什么时候 RR 是相关的。

MX 记录用来控制邮件发送。在记录中设定的记录是一个优先级和一个域名。优先级控制邮件发送的顺序，最小的数字最先发送。如果两个优先级相同，服务器就随机选取。如果在给定优先级下没有服务器响应，邮件传输代理将会退回到下一个最高优先级。优先级数字没有任何绝对含义。它们分别与域名相关。给定的域名是邮件发送的目的地。它必须具有一个相关的 A 记录，CNAME 记录是不够的。

对一个给定域名来说,如果既有 CNAME 记录又有 MX 记录,MX 记录就就会出错,并被忽略.相反邮件将会发给由 CNAME. 指向的 MX 指定的服务器.

```
example.com.          IN      MX      10      mail.example.com.
                      IN      MX      10      mail2.example.com.
                      IN      MX      20      mail.example.com

mail.example.com.     IN      A       10.0.0.1
mail2.example.com.    IN      A       10.0.0.2
```

例如：
将会在 mail.example.com 和 mail2.example.com(任何顺序)上发送邮件，如果都不成功，则以 mail.backup.org 发送。

7.3.3 设置 TTLs

RR 字段上的时间是一个以秒形式表示的 32 位整数，主要被解答器使用(当它们缓存 RRs

的时候)。

TTL 描述当 RR 被放弃前应该在缓存存储多长时间。目前在域文件中下列三种 TTL 类型：

SOA	在 SOA 中的最后字段是负缓存 TTL。它控制其他服务器缓存来自你的 no-such-domain (NXDOMAIN)响应多长时间。最大负缓存时间为三个小时(3h)；
$TTL	在域文件顶部的$TTL 指示为每个没有特殊 TTL 设置的 RR 给出了一个默认的 TTL；
RR TTLs	每个 RR 可以有一个 TTL 作为 RR 中的第二字段，这将控制其他服务器缓存它的时间长短。

这些 TTLs 默认为秒数,尽管秒数可以被任意指定,例如 1h30m.。

7.3.4 IPV4 的反向解析.

反向域名解析(也就是说 IP 地址到主机名的翻译)通过 in-addr.arpa 域和 PTR 记录实现.。in-addr.arpa 域入口可以作成最不重要到最重要(least-to-most)顺序，从左至右阅读。这与 IP 地址的通常顺序相反。这样，一台 IP 地址为 10.1.2.3 的机器将会右对应的 in-addr.arpa 名称: 3.2.1.10.in-addr.arpa。这个名称应该具有一个 PTR 资源记录，它的数据字段是主机名称或者任意的多重 PTR 记录，如果主机右超过一个名字的话。
例如.在[example.com]域中：

```
$ORIGIN          2.1.10.in-addr.arpa
3          IN    PTR       foo.example.com.
```

注意：例子中的$ORIGIN 行只给例子提供描述，在真实使用中不一定出现，在这出现仅仅表明例子是域所列源相关的。

7.3.5 其他的域文件指令

主域 hosts 文件格式最初定义在 RFC1035 中，随后被传播开来。然而，hosts 文件格式本身是独立的类，域的所有记录在一个 hosts 文件中是同类的。主域 hosts 文件指示包括：$ORIGIN、$INCLUDE 和$TTL。

7.3.5.1 $ORIGIN 指令

语法： $ORIGIN domain-name [comment]

$ORIGIN 设置附属于任何不合格记录的域名。当一个域被首先读入时存在一个任意的$ORIGIN <zone-name>，当前的$ORIGIN 附属于域(设定于$ORIGIN 辐角如果它不是绝对的话)

```
$ORIGIN          example.com.
WWW              CNAME        MAIN-SERVER
```

等价于:

WWW.EXAMPLE.COM. CNAME MAIN-SERVER.EXAMPLE.COM.

7.3.5.2 $INCLUDE 指令

语法：$INCLUDE filename [origin] [comment] *（ 文件名 [源] [注释)*

读和运行文件 filename 就好象它已经被包含进文件中。如果 ORIGIN 被设定，文件将会被设定为 ORIGIN 的值，否则就使用当前的$ORIGIN。

一旦文件被打开，ORIGIN 和当前的域名将会恢复到它们对$INCLUDE 有优先权的值。

注意：RFC1035 设定的当前源应该在一个$INCLUDE 之后恢复，但它并不要求当前域名也应该恢复。BIND9 全都恢复。这可以作为一个从 RFC1035 中的出的偏差，一个特性或者两者都是。

7.3.5.3 $TTL 指令

语法: $TTL default-ttl [comment]

为没有定义 TTLs 的后续记录设定即时时间。有效 TTLs 是在 0-2147483647 秒范围内.
$TTL 在 RFC 2308 中定义。

7.3.6 BIND 控制文件拓展：$GENERATE 指令

语法：

$GENERATE range lhs type rhs [comment]

$GENERATE 指令是用来生成一个资源记录序列,资源记录彼此之间只有一个重复性的不同。$GENERATE 指令可以容易地被用来生成一个记录集合来支持 RFC2317 中所描述的 sub/24 反向授权：无类的 IN-ADDR.ARPA 授权。

```
$ORIGIN 0.0.192.IN-ADDR.ARPA.
$GENERATE 1-2 0 NS SERVER$.EXAMPLE.
$GENERATE 1-127 $ CNAME $.0
```

等价于：

```
0.0.0.192.IN-ADDR.ARPA NS SERVER1.EXAMPLE.
0.0.0.192.IN-ADDR.ARPA NS SERVER2.EXAMPLE.
1.0.0.192.IN-ADDR.ARPA CNAME 1.0.0.0.192.IN-ADDR.ARPA
2.0.0.192.IN-ADDR.ARPA CNAME 2.0.0.0.192.IN-ADDR.ARPA
```

...
127.0.0.192.IN-ADDR.ARPA CNAME 127.0.0.0.192.IN-ADDR.ARPA
.

range	这可以采取下面两种方式当中的一种：开始-结束或者开始-结束/步长。如果使用形式 1，则步长为 1，开始、结束、步长都必须是正值。
Lhs	lhs 描述了要创建的资源记录所有者的名称。任何在 lhs 范围内单一的$符号都可以被重复值所代替。为了在结果中得到一个$，你需要避免与反斜线\一同使用$.例如：\$。$符号可以随意地和修饰成分来连用，这些修饰成分可能改变重复值的偏移、区段的宽度和基础。修饰成分用{来引入，紧接着$，用下面的方式：${offset[，width[,base]]}，例如：${-20,3,d}意味着从当前值中减去 20，以十进制填补宽度为 3 打印这个结果。可能的结果形式有十进制（d）,八进制（o），十六进制（ x 或大写字母 X）。默认的修饰成分是${0,0,d}。如果 lhs 不是绝对的，目前的$ORIGIN 就要被附加到名字的后面。为了与以前的版本兼容，结果中的$$仍然被认为是一个$的含义。
Type	目前支持的类型只有：PTR, CNAME, DNAME, A, AAAA 和 NS.
Rhs	Rhs 是一个域名，它的处理与 lhs 类似。

$GENERATE 指令是 BIND 的一个拓展，并不是标准区域文件的组成部分。

第八章、BIND9 的安全性

8.1 访问控制列表

访问控制列表（ACLs）是地址的匹配列表，你可以建立一个访问地址列表并指定名称，在以后的 allow-notify, allow-query, allow-recursion, blackhole, allow-transfer 等配置里面使用。

使用访问控制列表可以使你对于访问域名服务器的人进行良好的控制，而不使你的配置文件因为大量的 IP 地址而变得混乱。

使用 ACLs 来控制对你服务器的访问是一个很好的选择。由外部的用户来限制对你服务器的访问可以防止对域名服务器的欺骗和 DoS 攻击。

下面是一个如何正确地应用 ACLs 的例子：

```
// 创建一个名称为"bogusnets"的 ACLs 来阻止经常用于欺骗性攻击的（RFC1918）地址空间
acl bogusnets
{ 0.0.0.0/8; 1.0.0.0/8; 2.0.0.0/8; 192.0.2.0/24; 224.0.0.0/3; 10.0.0.0/8; 172.16.0.0/12;
//创建一个名称为 our-nets 的 ACL，并将其配置为实际现网的 IP 地址。
acl our-nets { x.x.x.x/24; x.x.x.x/21; };
options {
...
...
allow-query { our-nets; };
allow-recursion { our-nets; };
...
blackhole { bogusnets; };
...
};
zone "example.com" {
type master;
file "m/example.com";
allow-query { any; };
};
```

没有禁止递归服务时，在缺省状态下，系统允许外部的递归请求。

如果你想获得更多的如何使用 ACLs 来保护你的服务器的信息，你可以在 ftp://ftp.auscert.org.au/pub/auscert/advisory/AL-1999.004.dns_dos 上参考 AUSCERT 的建议。

8.2 chroot 和 setuid（对与 UNIX 服务器）

在 UNIX 服务器中，我们可以用”-t”选项让 BIND 运行在指定的环境中（chroot()），这种方法可以提高系统的安全性，主要是在系统遭到破坏时降低损失，我们称为“沙箱”。

另外一个 BIND 在 UNIX 系统中的有用特性是将 BIND 的后台程序以非特权用户身份运行。我们建议在以非特权用户身份运行 BIND 时，同时使用 chroot 的特性。

下面是一个在“沙箱”中装入 BIND，/var/named，并且以用户 202 运行 named setuid 的命令行举例。

/usr/local/bin/named -u 202 -t /var/named

8.2.1 chroot 环境

为了使得 chroot()环境在一个特定的目录（例如/var/named）中正常地运行，你需要建立一个包含所有 BIND 运行必需内容的环境。从 BIND 的角度来看，/var/named 是文件系统的根。你需要调整 directory 和 pid-file 的用户权限。

不像以前的 BIND 版本那样，你既不需要静态地编译 named，也不需要在新的根下安装共享库。但是，根据你的操作系统，你可能需要设置类似于/dev/zero, /dev/random, /dev/log, 和/或者 /etc/localtime.的内容。

8.2.2 使用 setuid 函数

在运行 named 后台程序之前，使用 touch 功能（来改变文件访问和更改时间）或者 chown 功能（来设置用户 ID 和/或组 ID）来改变 BIND 将要进行写操作的文件，注意如果 named 后台程序是以一个非特权用户方式运行，在服务器重新启动的时候，它不能绑定到新的受限制的端口。

8.3 动态更新安全性

对动态更新功能的访问应该被严格限制，在以前的 BIND 版本中实现这一点的唯一方式是基于要求动态更新的主机的 IP 地址，通过列出 IP 地址或者在 allow-update 选项中的网络前缀来进行。这个方法是不安全的，因为在动态更新 UDP 包中的源 IP 地址很容易被伪造。而且应该注意的是如果被 allow-update 选项所允许的 IP 地址包括将要进行转发动态更新的辅域名服务器地址，主服务器就很容易因为对辅域名服务器的更新而受到攻击，因为这些更新用辅域名服务器自己的源 IP 地址传向主服务器，这将使得主服务器毫无疑问地接受它。

由于这些原因，我们强烈建议通过处理签名（TSIG）来对更新进行密码鉴定。也就是说，allow-update 选项应该只列出 TSIG 名称，而不是 IP 地址或网络前缀。另外，还可以使用新的更新策略选项。

一些网站选择将所有动态更新的 DNS 数据保持在一个子域当中，并将这个子域授权为一个独立的区域。这种方式使得包含类似于公共网页和邮件服务器的 IP 地址这样的最高级别的关键数据根本就不需要允许动态更新。

第九章、疑难解答

9.1 一般性问题

9.1.1 BIND 不工作了，如何才能找出问题的根源？

解决安装和配置问题的最好方法是通过提前建立 log 文件所采取的保护性措施。log 文件提供了提示和信息的资源可以用来寻找问题的根源和解决的办法。

9.2 增加和改变序列号

区域的序列号只是数字——它们与日期是无关的。很多的人都将它们设置为一个表达时间的数字，而且经常采取的方式是 YYYYMMDDRR。很多的人都对这样的数字做过 2000 年问题的检验，他们将这个数字设置为 2000 来看系统是否工作。然后他们试图恢复原先的序列号。这将会导致问题因为序列号是用来指出一个已经被更新的区域的。如果在辅域名服务器上的序列号小于在主服务器上的序列号，辅域名服务器就会试图从区域中进行更新。

设置主服务器上的序列号小于辅域名服务器上的序列号将使得辅域名服务器并不从区域进行更新。

解决这个问题的方法是在这个数字后面加上 2147483647 (2^31-1)，重新装入区域并且确信所有的辅域名服务器都按照新的序列号进行了更新，然后重新设置你想要的序列号，并再此重新装入区域。

9.3 从哪里可以获得帮助

国际软件协会（ISC）提供对 BIND 和 DHCP 服务器的广泛的支持与服务的合同。共有 4 种可以选择的不同费用的支持，而每一个水平都包括对 ISC 项目，对产品和培训的有意义的折扣以及在错误修正上的优先权和无基金的特色需求。另外，ISC 还提供一个标准的支持合同包，包括对错误修正到远程支持的多种服务。还包括对 BIND 和 DHCP 的培训。

要讨论支持的安排，你可以联系 info@isc.org (mailto:info@isc.org)或是访问 ISC 的网页 http://www.isc.org/services/support/。

附录 A:附录

A.1 致谢

A.1.1 DNS 与 BIND 的历史

虽然域名系统是随着 1984 年 RFC920 的发表而“正式”开始的，新系统的核心却已经出现在 1983 年发表的 RFC882 和 RFC883 当中。从 1984 年到 1987 年，ARPA 网（今天 INTERNET 的先驱）成为了在一个快速扩张、操作化的网络环境中进行创建新的域名/地址计划实验的基地。过去写完的在 1987 年才发表的文献都根据这个模型的结果进行了完善。RFC1034：“域名的概念与工具”和 RFC1035：“域名的实现和规范”出版并成为了 DNS 实现建立的标准。

第一个工作的域名服务器，称为“Jeeves”，由 Paul Mockapetris 与 1983-84 年为在南加州大学信息科学中心（USC-ISI）和 SRI 国际网络信息中心（SRI-NIC）的 DEC 的最高级别 20 台机器上操作而写出的。一个基于 UNIX 机器的服务器——BERKELEY 网络名称域——随后不久由加州大学 Berleley 分校的研究生小组在美国安全高级项目研究管理局的同意下开发成功。BIND 直到 4.8.3 版本是由 UC berkeley 的计算机系统研究组（CSRG）持有。Douglas Terry, Mark Painter, David Riggle 和 Songnian Zhou 建立了最初的 BIND 项目组。之后，进一步的关于软件包的工作由 Ralph Campbell.来完成。Kevin Dunlap，一个数字装备公司的雇员在 CSRG 的贷款支持下，从 1985 年到 1987 年对 BIND 进行了两年的研究。很多其他的人在那个时候也为 BIND 的发展做出了贡献，比如 Doug Kingston, Craig Partridge, Smoot Carl-Mitchell, Mike Muuss, Jim Bloom 和 Mike Schwartz.。BIND 的维护问题随后被 Karels 和 O. Kure.所解决。

BIND 版本 4.9 和 4.9.1 由数字装备公司（现在是 Compaq 计算机公司）所发布。Paul Vixie, 后来是了 DEC 的一名员工， 成为了 BIND 主要的管理者。Paul 受到了 Phil Almquist, Robert Elz, Alan Barrett, Paul Albitz, Bryan Beecher, Andrew Partan, Andy Cherenson, Tom Limoncelli, Berthold Paffrath, Fuat Baran, Anant Kumar, Art Harkin, Win Treese, Don Lewis, Christophe Wolfhugel, 和其他人的帮助。

BIND 版本 4.9.2 由 Vixie 公司所发起，Paul Vixie 成为了 BIND 的主要设计者和程序师。

BIND 版本从 4.9.3 开始都是由国际软件协会来制造和维护，而且由 ISC 的成员提供服务。作为合作的设计者/程序师，Bob Halley 和 Paul Vixie 在 1997 年 5 月 8 日发布了 BIND 的第一个可供生产的版本。

今天 BIND 的发展由于许多公司的资助和许多个人的不懈努力工作而变得可能。

A.2. 历史的 DNS 信息

A.2.1. 记录资源类

A.2.1.1. HS = hesiod

[hesiod]类是由 MIT 项目 Athena 所开发的一种信息服务。它被用于共享不同系统数据库的信息，比如用户、组、打印机等等。其关键词 **hs** 是 hesiod 的同义词。

A.2.1.2. CH = chaos

Chaos 类被用来为 MIT 于 70 年代开发的局域网工具 CHAOSnet 具体化区域数据。

A. 3. 一般的 DNS 参考信息

A.3.1 IPv6 地址 (A6)

IPv6 地址是 128 位的地址编码，引入 DNS 以提供升级到 ipv6 网络的域名解析服务。由三种类型的地址：Unicast，用于单一端口的标志符；Anycast，用于一组端口的标志符和 Multicast，也是一组端口的标志符。这里我们描述一下全局 Unicast 方案。要了解更多的信息，请看参考文献 2374。可以集合的全局 Unicast 格式如下：

3	13	8	24	16	64 位
FP	TLA	RES	NLA ID	SLA ID	Interface ID
<……Public Topology……>					
	公 共 拓 扑			<Site Topology>	
				网站拓扑	Interface Identifuer 界面标志符

其中：

FP	格式前缀
TLA ID	最高级别集合标志符
RES	留做以后使用
NLA ID	下一级别集合标志符
SLA ID	站点级别集合标志符
INTERFACE ID	界面标志符

公共拓扑由上级提供者或 ISP 提供，而且地址范围是（严格）根据 IPv4 网段的规定。站点拓扑是你可以建立子网的空间，就像把 IPv4 /16 网络分割成/24 子网一样。而界面标志

符是一个给定网络的单独接口的地址。（使用 IPv6，地址属于接口而不是机器）

IPv6 的划分子网的能力比 IPv4 要灵活得多：子网划分现在可以在位边界进行，方式相同于无类域间路由（CIDR）的方法。

一个 A6 的全局 unicast 地址的公共拓扑内部结构包括：

3	13	8	24
FP	TLA ID	RES	NLA ID

如果三位的 FP（格式前缀）为 001 则说明这是一个全局 unicast 地址。对于其他类型的地址，FP 的长度可能会变化。

13TLA（最高级别集合）位给出了你的最高级别 IP 链路载体的前缀。

8 个保留位

24 个下一级别集合。这允许拥有 TLA 的组织把他们的 IP 空间的一部分向客户组织分发，这样客户就可以通过增加更多的 NLA 位进一步地划分网络，进而将 IPv6 前缀分发给他们的客户而继续这一过程。

站点拓扑段是没有特别的结构的，组织可以按照自己的意愿以任何方式分配这些位。

在网络中，接口标志符必须是唯一的。在以太网络上，保证这一点的方式是将地址设置为硬件地址的前三个字节，"FFFE"，接着是硬件地址的最后三个字节。第一个字节的最不重要的位应该被补充完整。地址可以写作以一个冒号分割的 32 位区块，而区块开始时的 0 可以省略，例如：

3ffe:8050:201:9:a00:20ff:fe81:2b32

IPv6 地址的具体化可能包含一个很长的零字符串，所以设计者们使用一个简化的形式来具体化它们。两个冒号（”::”）意味着它可以代替一长串的零，这只可以在一个地址中使用一次。

A. 4. 参考和建议阅读文献

A.4.1. 注释的要求 (RFCs)

包括 DNS 在内的关于网络协议序列的详细文献是作为技术的注释要求（RFCs）的一部分出版的。标准自身是由 Internet Engineering Task Force (IETF)和 Internet Engineering Steering Group (IESG). 定义。参考文献可以从 ftp://www.isi.edu/in-notes/RFCxxx.txt (ftp://www.isi.edu/in-notes/)（其中 xxx 是 RFC 的编号），同时 RFC 还可以从 http://www.ietf.org/rfc/上获得。

参考文献：

标准

[RFC974] C. Partridge, Mail Routing and the Domain System, January 1986.

[RFC1034] P.V. Mockapetris, Domain Names ?Concepts and Facilities, November 1987.

[RFC1035] P. V. Mockapetris, Domain Names ?Implementation and Specification, November 1987.

被提议的标准

[RFC2181] R., R. Bush Elz, Clarifications to the DNS Specification, July 1997.

[RFC2308] M. Andrews, Negative Caching of DNS Queries, March 1998.

[RFC1995] M. Ohta, Incremental Zone Transfer in DNS, August 1996.
[RFC1996] P. Vixie, A Mechanism for Prompt Notification of Zone Changes, August 1996.
[RFC2136] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound, Dynamic Updates in the Domain Name
System, April 1997.
[RFC2845] P. Vixie, O. Gudmundsson, D. Eastlake, 3rd, and B. Wellington, Secret Key Transaction
Authentication for DNS (TSIG), May 2000.
仍然在发展中的被提议的标准
[RFC1886] S. Thomson and C. Huitema, DNS Extensions to support IP version 6, December 1995.
[RFC2065] D. Eastlake, 3rd and C. Kaufman, Domain Name System Security Extensions, January 1997.
[RFC2137] D. Eastlake, 3rd, Secure Domain Name System Dynamic Update, April 1997.
在 DNS 实现中其他重要的参考资料
[RFC1535] E. Gavron, A Security Problem and Proposed Correction With Widely Deployed DNS Software., October 1993.
[RFC1536] A. Kumar, J. Postel, C. Neuman, P. Danzig, and S. Miller, Common DNS Implementation
Errors and Suggested Fixes, October 1993.
[RFC1982] R. Elz and R. Bush, Serial Number Arithmetic, August 1996.
资源记录类型
[RFC1183] C.F. Everhart, L. A. Mamakos, R. Ullmann, and P. Mockapetris, New DNS RR Definitions,
October 1990.
[RFC1706] B. Manning and R. Colella, DNS NSAP Resource Records, October 1994.
[RFC2168] R. Daniel and M. Mealling, Resolution of Uniform Resource Identifiers using the Domain
Name System, June 1997.
[RFC1876] C. Davis, P. Vixie, T., and I. Dickinson, A Means for Expressing Location Information in the
Domain Name System, January 1996.
[RFC2052] A. Gulbrandsen and P. Vixie, A DNS RR for Specifying the Location of Services., October
1996.
[RFC2163] A. Allocchio, Using the Internet DNS to Distribute MIXER Conformant Global Address
Mapping, January 1998.
[RFC2230] R. Atkinson, Key Exchange Delegation Record for the DNS, October 1997.
DNS 和互联网
[RFC1101] P. V. Mockapetris, DNS Encoding of Network Names and Other Types, April 1989.
[RFC1123] Braden, Requirements for Internet Hosts - Application and Support, October 1989.
[RFC1591] J. Postel, Domain Name System Structure and Delegation, March 1994.
[RFC2317] H. Eidnes, G. de Groot, and P. Vixie, Classless IN-ADDR.ARPA Delegation, March

1998.
DNS 操作
[RFC1537] P. Beertema, Common DNS Data File Configuration Errors, October 1993.
[RFC1912] D. Barr, Common DNS Operational and Configuration Errors, February 1996.
[RFC1912] D. Barr, Common DNS Operational and Configuration Errors, February 1996.
[RFC2010] B. Manning and P. Vixie, Operational Criteria for Root Name Servers., October 1996.
[RFC2219] M. Hamilton and R. Wright, Use of DNS Aliases for Network Services., October 1997.
其他与 DNS 相关的参考文献
[RFC1464] R. Rosenbaum, Using the Domain Name System To Store Arbitrary String Attributes, May
1993.
[RFC1713] A. Romao, Tools for DNS Debugging, November 1994.
[RFC1794] T. Brisco, DNS Support for Load Balancing, April 1995.
[RFC2240] O. Vaughan, A Legal Basis for Domain Name Allocation, November 1997.
[RFC2345] J. Klensin, T. Wolf, and G. Oglesby, Domain Names and Company Name Retrieval, May
1998.
[RFC2352] O. Vaughan, A Convention For Using Legal Names as Domain Names, May 1998.
已废弃的和未实现的实验
[RFC1712] C. Farrell, M. Schulze, S. Pleitner, and D. Baldoni, DNS Encoding of Geographical Location, November 1994.

A.4.2. 互联网草案

互联网草案（IDs）是 Internet Engineering Task Force.所使用的严格的工作文件。它们本质上是处在初始发展阶段的 RFCs，实现者们是非常小心的不把它们当作档案来处理。而且除非声明它们是“正在发展中”，这些文献是不能被摘录和引用的。草案有六个月的生命期，在这之后除非它们被作者所更新，它们都将被删除。

A.4.3. 关于 BIND 的其他文件

参考文献
Paul Albitz and Cricket Liu, DNS and BIND, 1998.