序

“所有程序员都是理想主义者。”大约三十年前，Frederick P. Brooks, Jr.写下了这么一句至理名言（注 1），而 Postfix 邮件系统刚好就是最佳例证。Postfix起源于我在纽约州 IBM Research 的网络安全部门的一个半年期的项目。虽然半年的时光确实足以让我换掉自己工作站上的邮件系统，但是还不足以建构出一套通用的完整邮件系统。在接下来的几年，一组合作无间的专家在测试这套软件时，又为它添加了许多程序代码。五年之后，当我决定将以通用公共许可证（GPL）来发表 Postfix 时，它的规模 —— 程序量与功能性，已经是初期版本的两倍了，而且研发进程比以往更为活跃。

Postfix的主要目标之一是使其具有“宽广的适用性”。写出 Postfix软件只是达成此目标的第一项挑战而已，第二项挑战在于如何使得软件广为流传。虽然高手级专家很乐意阅读 Postfix 随附的“友善说明”，但是大多数人需要更“温柔”的对待。老实说，如果没有一本介绍Postfix 系统所蕴涵的理念并示范如何设定各种基本功能与特殊功能的专业用书，我不认为 Postfix 有机会达到“宽广的适用性”目标。所以我很乐意把这本书的写作机会让给 Kyle Dent。

如同 Postfix一样，我将这本书视为必须持续进行的工作。在本书的第一版手稿刚付梓时，Postfix本身又历经了好几次重大改版。有些改变甚至是与 Kyle讨论的结果，用意在于让Postfix更容易了解；有些改变则是增加了前版软件没做出来的功能；还有些改变是被计算机病毒与垃圾邮件给硬逼出来的。除了新增或扩充的功能之外，还有许多发生在维护与改良过程中的不显眼改变。

注 1: P. Brooks, Jr..The Mythical Man-Month: Essays on Software Engineering. Addison Wesley. 1975 年。

本书内容以 Postfix 2.0版为准并涵盖过去曾经广为流传的某些重要旧版本与其的差异。由于 Postfix仍在持续改进中，所以这本书也会慢慢过时，甚至终有一天需要改版。虽然我很荣幸能将这本书的第一版介绍给你，但是，我已经在盼望下次再见到你的机会——或许就在不久的将来。

Wietse Venema
Hawthorne, New York
September 19, 2003

前言

每当我想起Internet初期的开路先锋，崇拜之情不禁油然而生。他们是一群令人敬佩的高手，他们当初为“小”网络（相较于今日的Internet而言）设计的各种技术与软件仍然延用到今日的Internet，即使其规模早已不可同日而语，而环境也不知道复杂了多少倍。当然，再怎样的高瞻远见，也不可能没有成长的苦痛；早期的工作成果不管再怎么调整，也不见得能够完全适应全新环境的挑战。 Sendmail就是一个活生生的例子，虽然它目前仍是Internet上最主要的邮件系统，但是已有越来越多管理员感受到它当初根本是为另一种构想而设计出来的东西。

Postfix的优势在于它不必面对历史的牵绊。事实上， Wietse Venema最初的设计动机正是要克服老软件摆脱不掉的束缚，尽情发挥“后见之明”。

我第一次开始使用Postfix是因为我需要架设一组讲究高安全性的系统。 Postfix对于灵活性与安全性的承诺引起了我的兴趣，而它确实没让我失望。不仅如此，我还迷上了它。从此以后，每当我需要架设邮件服务器时， Postfix总是我的第一选择。

虽然Postfix很容易上手，也有详尽的在线说明文件，但是缺乏一份随手的参考资料，也没有关于运作原理方面的书籍。为了弥补这方面的缺憾，我决定自己着手整理资料，写出这本书。本书主要宗旨在于解释Postfix背后所隐藏的概念以及在实践中应注意的细节，并示范如何设定Postfix的特定功能。

为一套仍在积极发展中的软件写书，感觉很像“抽刀断水”。所以，很遗憾地，即使这本书已经出版了，它仍然不是完整的。但本书的章节安排已经尽量排除不相干或即将过期的信息，所以你会发觉本书的信息还可以存活一段颇长的时间。当然，你可能还需要从在线文件、网站或Postfix邮件列表（mailing list）找到最新功能的说明，以弥补本书的不足。

读者群

Postfix是为 Unix 系统设计的网络应用系统。你对网络与 Unix 懂得越多，在管理 Postfix server 时就越得心应手。本书的写作原则是尽量让 Unix 新手也能看懂，不过，如果你没有任何 Unix 经验就想从本书学到如何管理 Postfix server，那就不太可能了。这本书的重点放在 Postfix 本身以及理解 Postfix 的功能和配置 Postfix server 所需的必要概念。如果你从来没碰过 Unix，建议你先阅读关于 Unix 系统的入门书籍。Evi Nemeth 所著的《Unix System Administration Handbook》（Prentice-Hall 出版）是个不错的选择，因为该书对于 E-mail 的解释特别详尽。RFC 也是非常有价值的信息来源，只不过看起来非常枯燥乏味了。

章节编排

前三章讲述 Postfix 与 E-mail 的基础知识。第四章到第七章讨论典型 Postfix server 具有的基本功能。从第八章到第十五章，每一章分别描述一种特殊应用，你可能只需要其中几种应用，也可能完全不需要，全看你的 Postfix server 用于何种用途而定。

第一章简介

介绍 Postfix 与 E-mail 的基本概念，同时也探讨 Postfix 的某些设计决策。

第二章基本概念

提供了解本书其他内容所需的基本概念。对 Unix 与 E-mail 已有基本认识的读者，可以直接跳过这一章。

第三章 Postfix 的结构

解释 Postfix 的模块化结构以及 Postfix 处理邮件信息的流程。

第四章基本的配置与管理

配置、管理 Postfix server 所需的基本技能。

第五章队列管理

解释 Postfix 的队列管理系统的基本原理，示范各种队列操作工具的用法。

第六章 E-mail 与 DNS

解释 DNS 如何影响 E-mail 的传递，讨论如何设定 DNS 才能很好地配合 Postfix 的运作。

第七章本地投递与 POP/IMAP

探讨 Postfix 如何运行本地投递操作以及如何搭配 POP 与 IMAP server。

第八章虚拟网域

讨论如何使用 Postfix 来接收多个虚拟网域的电子邮件。

第九章邮件转发

讨论如何使 Postfix 转发邮件或成为网关系统。

第十章邮件列表

示范如何在 Postfix 中设定邮件列表，以及 Postfix 如何搭配 Majordomo 与 Mailman 这两套最热门的邮件列表管理系统。

第十一章反垃圾邮件

讨论如何运用 Postfix 提供的各种过滤机制将恼人的垃圾邮件挡在网络大门外。

第十二章 SASL 身份验证

示范如何使用 SASL 函数库来验证 SMTP client 的身份，让 SMTP client 在任何地方都可以使用 Postfix 的邮件转发服务。

第十三章传输层安全协议

使用 TLS 让 Postfix server 与 client 之间可以进行加密通信。

第十四章内容过滤

探讨 Postfix 如何搭配外部的内容过滤程序。

第十五章外部数据库

探讨如何利用外部数据来源形成 Postfix 的查询表。

附录一配置参数

按字母顺序列出 Postfix 的所有配置参数，方便读者参考查阅。

附录二 Postfix 的工具程序

列出 Postfix 随附的命令行工具，并简略说明它们的用法。

附录三 Postfix 的编译与安装

示范如何编译 Postfix 的源程序，并将其安装到你的系统上。

附录四常见问题集

回答关于 Postfix 的最常问的问题。

排版约定

本书用各种印刷效果来强调不同的文意内容：

斜体字（*Italic*）

用于表示命令、电子邮件地址、 URI、文件名、强调的文字、第一次出现的专有名词以及对书籍或文章的引用。

等宽字（`Constant Width`）

用于表示字面量、常量值、代码列表以及 XML标记。

等宽斜体字（*`Constant Width Italic`*）

用于表示可替换参数以及变量名。

等宽黑体字（**`Constant Width Bold`**）

用于突出表示程序代码列表中需要加以讨论的部分。

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 Media, Inc.
1005 Gravenstein Highway North
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

O'Reilly的每一本书都有专属网页，你可以在此找到关于本书的相关信息，包括可下载的范例程序、勘误表与相关资源的链接。

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

询问技术问题或对本书进行评论，请发电子邮件到：

info@mail.oreilly.com.cn
bookquestions@oreilly.com

最后，您可以在以下站点找到我们：

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

致谢

当然，第一位我应该感谢的人是Wietse Venema —— Postfix的原创者。我不仅要感激他的指导，更要感谢他对Internet 的贡献。能够有机会与他合作这本书，实在是我的荣幸，同时也让我受益良多。这本书有许多内容甚至可说是他的心血结晶。

O'Reilly向来是我很仰慕的一家公司，尤其在与他们合作共事之后，景仰之情更是有增无减。我的编辑 —— Andy Oram，可说是这家公司的代表性人物，与他讨论事情是一种享受，因为他的见解往往一针见血，对我帮助良多，我感谢他无比的睿智。感谢Lenny Muellner 协助我使用文本处理工具，同时还要感谢David Chu花了这么多时间来协助我处理写作上的琐碎细节，感谢Robert Romano为本书绘制出如此专业的图解。最后要感谢 Reg Aubry对于本书制作过程的指导。

本书的多位技术校阅人员也功不可没，他们不仅协助我保证书籍内容的正确性，甚至在写作风格方面也给了我许多有用的建议。感谢 Rob Dinoff、Viktor Dukhovni（Victor Duchovni）、Lutz Janicke与 Alan Schwartz。我真希望我有这么一个梦幻团队。

我还要感谢*postfix-users@postfix.org*论坛里的许多成员。这是一个几乎没有杂音的活跃论坛，成员中有许多令人印象深刻的高手，他们不仅乐于帮助Postfix的用户，同时也对Postfix 本身的改进提出了深切的提案。

我的妻子，也是我的首席编辑 —— Jackie，是我的大债主，我欠她大笔人情债。她谨慎校对我的初稿，进行严格的可信度、严谨度测试（若知道她揪出我多少错误，你会大吃一惊）。她的无比耐心与可贵的付出大幅提升了本书的质量。即使一遍又一遍地阅读与改写，她也不会给我坏脸色。得妻如此，夫复何求？

第一章

简介

Internet E-mail的历史可回溯到20世纪70年代早期，当第一段信息流过Arpanet（今日Internet的前身）时，E-mail就成为Internet上最广泛的应用程序，并持续到今天。以前的电子邮件传递程序相当简单，通常只是将邮件文件从一部大主机搬移到另一部服务很多用户的大主机上而已（译注1）。随着Internet的改进，网络本身变得越来越复杂，邮件系统需要更有兼容性的工具才能在不同的网络之间，甚至在不同类型的网络之间传递邮件。80年代早期出现的Sendmail包就是为了应付各种不同邮件系统而设计的，它很快地成为Internet上最重要的邮差。

今日，大部分的Internet网站使用SMTP协议来收发邮件。虽然Sendmail依然是分布最广的SMTP server，但是它也逐渐显露出不足。Sendmail的单体式结构已经成为许多安全隐患的主因，而且难以配置与维护。

Postfix的原始构想，就是要取代普遍的Sendmail。它的设计结构消除了许多产生安全隐患的机会，同时也降低了维护工作的复杂性。Postfix管理员只需控制管理两个适合“人”看的配置文件即可（译注2），而且打从一开始，Postfix就被刻意设计得能妥善应付意外的软硬件问题。

译注1：现在你知道为何Sendmail支持“UUCP”（Unix-to-Unix-CoPy）这种已经没有人使用的协议了吧！

译注2：Sendmail的配置文件（*sendmail.cf*），主要是为了方便sendmail程序在运行时的解读，所以被刻意设计成适合“程序解读”的格式。

Postfix 的起源与设计理念

Postfix 的原作者是 Wietse Venema，他做的安全防护工具与论文在业界相当有名。在 1998年 12月，他决定让 Postfix正式成为 Open Source软件。IBM Research 赞助了初始版本（IBM称呼此包为 Secure Mailer），并不断支持其研发工作。可靠性、安全性、效率、灵活性、容易使用、兼容于 Sendmail，是 Postfix 研发初期就订下的理想目标。

可靠性 *(Reliability)*

Postfix真正的价值，要在严苛的条件下才会逐渐显现出来。不管环境如何简单，软件都有可能遇到意外状况。比方说，有许多软件系统在耗光内存或磁盘空间时，其行为就变得不可预测。 Postfix能够侦测出这类状况，让系统有机会恢复正常，而不至于将问题搞得更糟。不管遇到怎样的障碍， Postfix总是采用任何能够采用的预防措施，以稳定、可靠的方式应变。

安全性 *(Security)*

Postfix 假设它自己处于一个充满敌意的环境，设置了多层的保护措施来抵御攻击者。整个 Postfix 系统都贯彻了“最低权限”（least privilege）这个安全理念：每一个可以独立出来的功能，都分别写在不同的模块里，并以最低限度的权限在专属的进程环境（process context）里独立运作。权限较高的进程，决不会信任没有特权的进程。非必要的模块，可以被管理员移出系统或停用，借此提高安全性，并简化维护管理的工作。

效率 *(Performance)*

“效率”是 Postfix 中心理念之一，事实上，它甚至采取了相应步骤来确保它的运行不会影响到其他系统的效率。它使用多种技术来限制新建进程的数量以及处理信息时所需的访问文件系统的次数。

灵活性 *(Flexibility)*

整个 Postfix系统其实是由多个不同程序与子系统所构成的，这样的系统结构使得 Postfix 非常有灵活性，因为每个组件的行为皆可通过配置文件来个别地调整。

容易使用 *(Ease-of-Use)*

Postfix是最容易架设与控制管理的邮件系统之一，它使用直接的配置文件与简单的查询表（lookup table）来转换地址、传递邮件。隐藏在 Postfix 配置文件背后的理念是“平实”(least surprise)，也就是说，Postfix的行为结果能尽可能符合大多数人的预期。当 Venema博士面临设计抉择时，他选择的是最多人会认为合理的方案。

兼容于 *Sendmail*

兼容于 Sendmail 的特性，使得 Postfix 可以轻易替换掉系统上原有的 Sendmail，而不必迫使用户做出任何改变，也不会破坏原本依赖 Sendmail 的任何应用程序。

Postfix 支持 Sendmail的约定，像 */etc/aliases*与 *.forward*文件。而 Sendmail 本身的可执行文件 —— `sendmail`，则被替换成 Postfix 的版本，这个替代版具有相同的命令行自变量，只不过实际的运行工作是交由 Postfix 系统来完成。虽然你的 Sendmail相关程序可以继续使用，但是 Postfix 的改进已与 Sendmail 无关，甚至也没必要以同样的方式来实现出所有 E-mail 功能。

E-mail 与 Internet

在专业的 E-mail解决方案中，整个系统的各个组件通常都是来自同一家公司的软件，你不能随意更换个别组件，因为通常没有“等效程序”可以更换。另一方面， Internet E-mail 系统采取开放式设计，任何人都可以用不同的软件组合来架设一套邮件系统，并且能够与其他 Internet E-mail 系统交换信息，唯一前提是所有软件都必须遵守一组公开的标准与协议。这组协议制定了邮件的标准格式、邮件系统各角色的定位与任务以及邮件从发送方传递到接收方的详细过程。 Postfix本身不是一个完整的 Internet E-mail 邮件系统，它只是整个系统中的一个重要组件。大多数 E-mail 用户只熟悉用来读信、写信的软件，称为“邮件用户代理”（Mail User Agent，MUA），常见的 MUA 实例包括：mutt、elm、Pine、Netscape Communicator 与 Outlook Express。其任务是让用户能够读信、写信、寄信。不过， MUA 并非直接将 E-mail 送到收件人手中，而是由“邮件传输代理”（Mail Transfer Agent，MTA）代为传递。Postfix 扮演的角色正是 MTA。

邮件系统的组成

当你寄信时，信件是交给邮局处理。同理，当你要求 MUA 送出一封邮件，它只是将该邮件交给一台运行 MTA 软件（例如 Postfix）的服务器。图 1-1 是 E-mail 从发信者到收件人所经过的处理流程。MTA 的任务是接受 MUA 的委任，将 E-mail 从一个系统递送到另一个系统，并收下远方 MTA 送来的邮件。每当 MTA收到 MUA 的寄信请求，它会先判断是否应该受理。通常，如果邮件是来自本地系统的用户，或是本地网络上的系统，或是任何特许可以通过它转发（relay）邮件到其他目的地的网络，MTA 都会受理寄信请求。另一方面，MTA也会依据“收件人”来决定是否要收下邮件。如果收件人是本地系统的用户，或是收件人位于它知道要如何转递（forword）的其他系统，MTA 就会收下信息。

MTA收下邮件之后，它必须决定下一步做什么。它有可能将邮件递送给自己系统上的用户，也有可能将邮件交给另一个 MTA 来继续传递。对于要交给其他网络的邮件，有可能会经过多个 MTA 接力传递。如果 MTA无法递送信息，也无法转交给其他 MTA 处理，则退信给原发信者，或是发出通知函给系统管理员。一般，个人的 MTA server 通常是由 ISP 控管；公司员工的，则可能由企业的信息系统部门控管。

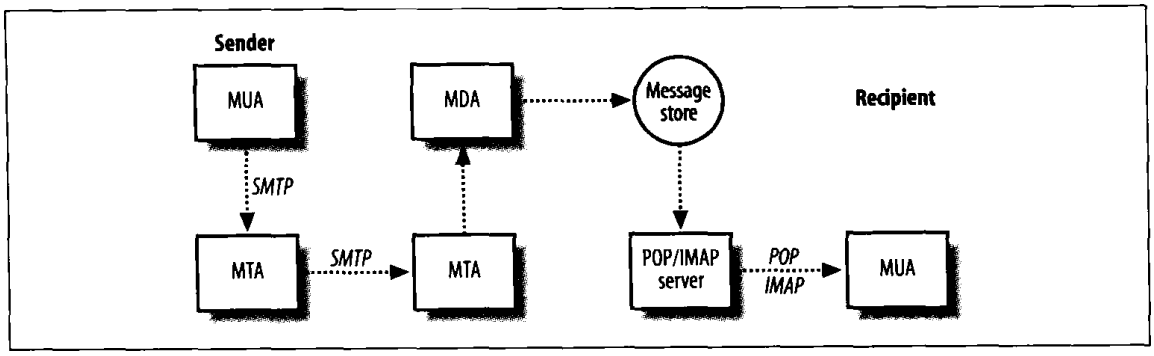

图 1-1 简化的 Internet E-mail 传递流程

邮件终点站的 MTA，在发现收件人是本地系统的用户之后，必须将邮件交给“邮件投递代理”(Message Delivery Agent，MDA)。MDA 可能将信息存放在普通文件夹内，也可能存入专门存储电子邮件的特殊数据库。不管是哪一种形式，任何用来长期保存邮件的机制，我们一律统称为邮箱（message storage），或俗称为“信箱”。

邮件被存入邮箱后，它就待在那里，等待收件人将它收走。收件人使用 MUA来取信、读信。提供邮箱访问服务的服务器软件，并非当初收下信息的 MTA，两者的角色是分离的。MUA 必须让用户成功通过身份验证，才可取走邮箱里的邮件，呈现给用户阅读。

由于 Internet E-mail 采用开放式标准，所以 MTA、MDA、MUA等不同角色，可分别由许多不同的软件包来扮演。实现相同协议的不同包，可以彼此互相交流，而不管它们是在什么系统上运行。如果将一个完整的 E-mail 邮件系统集中在一起，你可能会发现，处理SMTP的是一套软件，处理 POP/IMAP的是另一套软件。但邮件系统中的每一种角色，都有许多不同的软件可以选择。

主要的 E-mail 协议

邮件系统里各种角色之间的通信，受各式各样的标准与协议的规范。定义这些协议与标准的文件，由 Internet Engineering Task Force（IETF）维护管理，IETF以 Request For Comments（RFC）的形式公布它们。 RFC 是一种被赋予正式编号的标准文件，它们用于解释特定的技术或协议。

用于传递信息的标准协议是 *Simple Mail Transport Protocol*（SMTP），它定义于 RFC 2821，主旨在于规范两部主机通过网络交换 E-mail的对话原则。用于收信的协议是 *Post Office Protocol*（POP）或 *Internet Mail Application Protocol*（IMAP）。IMAP（RFC 2060）与 POP（RFC 1939）都是描述如何从邮箱取出邮件。IMAP 协议的出现比 POP

略晚，其功能也较完备。但不管是 POP还是 IMAP，E-mail都是保存在一部中央服务器中，让收件人可通过网络取信。

请注意，既然MUA与SMTP server可以分别位于不同的主机， MUA与POP/IMAP server也没必要非得位于同一个系统上不可。这就是为何 MUA 软件需要你分别设定 POP/IMAP与 SMTP的原因。ISP有可能将寄信与收信服务分散在不同的服务器，这样出差到外地的企业员 工，可使用公司的POP/IMAP server来收信，并使用ISP的 SMTP server来寄信。在 SMTP server 上运作的MTA 软件，会持续在固定的 TCP port（port 25）等待 MUA或其他MTA 的寄信请求。

SMTP 与寄信

SMTP是用于“寄信”（传出邮件）的协议。当 MUA要求 MTA代为送出一封邮件，以及一个 MTA将邮件送到另一个 MTA时，都使用 SMTP协议。原始的 SMTP协议并没有身份验证的设计，但是后来的扩充版本 —— ESMTP，加入了这方面的功能。关于如何验证 SMTP 用户的身份，请参阅第七章。

POP/IMAP 与收信

当用户想从邮箱取出他们的邮件，必须使用 MUA 连接到 POP或 IMAP server，由服务器代为访问邮箱。POP与 IMAP之间最大的差异，在于邮箱的管理方式。 POP用户通常将所有邮件从服务器搬回自己的主机； IMAP则容许用户通过网络要求服务器代为管理邮件。许多服务器同时提供两种协议，所以我们通常统称它们为POP/IMAP server。POP与 IMAP两者都完全没有寄信的能力，只能帮用户处理事先收到的邮件。关于 Postfix如何搭配 POP/IMAP server，请参阅第十二章的说明。

并非所有用户都需要透过 POP/IMAP来访问邮箱。举例来说，拥有 Unix shell 账号的用户，可以设定他们的 MUA直接读取同一机器上的邮件文件。

Postfix 的角色

在整个 Internet邮件系统中，Postfix担任 MTA的角色，负责在服务器之间传递邮件，并收下其他系统寄到本地系统的邮件。它不处理任何 POP或 IMAP 通信内容。

图 1-2 描绘了 Postfix扮 演“ MTA ”与“ 本地信使 ”两种角色时的情况。当作为 MTA 时，Postfix 使用 SMTP协议通过网络收发 E-mail信息；当作为本地信使时，则是直接将邮件分送到邮箱，或是交由特殊的 MDA处理。

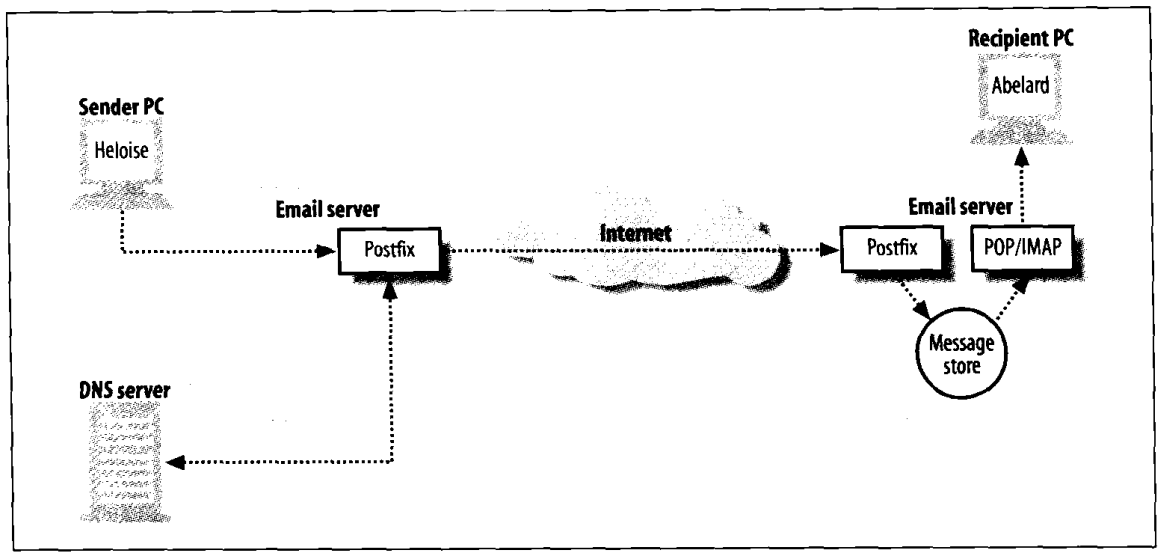

图 1-2 典型的 E-mail传递流程

在图 1-2的例子中，收发双方的 SMTP server都是 Postfix，但其实这不是必要条件。因为 Postfix 的行为符合 Internet标准的规范，所以，即使另一方的 MTA 不是 Postfix ，双方也可以互相交换邮件。事实上， Postfix 可以跟任何使用 SMTP 语言的 MTA 软件交谈（即使某些软件使用得不是那么规范）。在我们的例子中，Heloise (*heloise@oreilly.com*) 想要寄出一封信给 Abelard (*abelard@postfix.org*)，她使用自己的 MUA 编写好邮件，然后用 SMTP提交到她的 MTA。担任 MTA 角色的 Postfix server 答应帮她将邮件送到目的地，所以收下她的 MUA 所提交的邮件。这时候， Postfix server必须依据收件人的地址（即 *abelard@postfix.org*）来决定该邮件应该送到哪里去。在查询了 DNS server之后，Postfix 可以知道 Abelard 网域 (*postfix.org*) 的邮件应该要送到哪一部 SMTP server（关于 DNS 与 E-mail 的关系，请参阅第六章）。 Abelard的 Postfix server 收下邮件并存放在适当的邮箱之后，剩下的事就等 Abelard 主动上线收信了，换言之， Postfix 的工作就到此为止，因为当 Abelard 上来收信时，他是使用自己的 MUA 通过 POP或 IMAP 来取得邮箱中的邮件。

这个例子并没有详述 Postfix在递送邮件时的复杂细节。比方说，如果有多位收件人，Postfix还必须要搞清楚如何分批传送；如果因系统或网络问题使得 Postfix无法联络其中一两台收信主机时，Postfix还必须暂时将信息排入队列，然后定期试着重新递送。从用户的观点来看，这些琐碎细节几乎是隐形的；但从 Internet 邮件系统的观点来看，Postfix承担了绝大部分的递送工作。

Postfix 的安全性

邮件系统免不了要面对可能的攻击，因为它们的功能需要它们能够接受来自陌生系统的数据。这项挑战形同要建构出能抵御攻击的系统，而任何优良的安全防护策略都涉及多层保护。处于易受攻击环境的公众系统，其安全威胁的问题更加严峻。 Postfix 采取了预防性的多层防卫措施来确保安全。 Postfix本身的结构设计，减小了因本身安全隐患而造成安全问题的严重性，而对于Sendmail这种采用单体式设计的特权程序，结构上或程序上的小错误，就可能造成严重的损伤。

模块化设计

Postfix 的模块化结构，成为其保证安全性的重要基础。每个 Postfix 进程的权限都被压制到最低限度，仅刚好足够完成其工作内容。 Sendmail 的安全问题特别严重，原因之一就是Sendmail大部分时间都是以特权身份运行，追根究底，这正是因为 Sendmail将MTA的所有功能都集中在同一个程序里，难以分工所致。另一方面， Postfix 将不同的功能分别写成独立程序，以个别进程运作，而每个进程都只需要能够完成其工作的最低限度权限即可，甚至可将不需要的功能拿掉，这就彻底杜绝该功能遭受利用的机会。举例来说，如果 Postfix 是在一台网络防火墙系统上运行，它只需要转发邮件，而不必担任本地信使，则可以关掉分送本地邮件的功能。 Postfix 的各进程之间是彼此隔离的，而且几乎不需要任何IPC （进程间通信），每个进程都可以靠自己得到所需的信息，不必仰赖其他进程的协助。

Shell 与进程

在大部分情况下，投递邮件的过程并不需要 Unix shell process ,只有一种情况例外 在 Postfix 将信息放到环境变量之前，必须靠 shell process来“净化”信息。但 Postfix 必须试着先消除任何对 shell 有特殊意义的字符，然后才能将数据开放给 shell。

大部分的 Postfix 进程都是由一个可靠的 master daemon 所启动的，它们不是 master 的子进程，所以，它们没有任何关于父子继承关系或通信的安全问题。即基本上，任何使用信号(signal)、共享内存、公开文件或任何其他形式的IPC的攻击手法，对 Postfix都是无效的。

与生俱来的安全性

对于应用软件的另一种常见攻击手法是“缓冲区溢出”（buffer overflow）。这类攻击手法的原理，是想办法使程序将数据写到它不应该写的内存地址，借此改变程序的运行流

程，进而获取进程的控制权。先前已经说过， Postfix的每一个进程都只有最低限度的权限，所以，就算进程被胁持也不至于造成重大破坏。事实上， Postfix程序本身不会将动态数据放在固定容量的缓冲区中，这使得“缓冲区溢出”的攻击手法几乎不可能得逞。

Unix 系统提供了一种重要的安全防护机制 —— *chroot*，来改变应用程序见到的“根目录”。管理员可选择文件系统中的某一个子目录（例如 `/var/spool/postfix/` ）来作为应用程序的新的根目录，当程序运行时，它见到的文件系统仅限于指定节点之下的子目录，而见不到节点之上的部分。这样做的好处在于即使进程被外界胁持，也无法访问重要的系统目录（例如／etc/、/sbin／等）。并非所有应用程序都适合用 chroot，因为有些程序需要能够看到整个文件系统，或是所需要的文件分布得太过零散，但是 Postfix本身的设计使得它非常适合搭配chroot（关于 chroot的工作细节，请参阅第四章）。将Postfix隔离到一个专属的文件系统空间，最大的好处是攻击者就算成功入侵Postfix ，也挣脱不了 chroot的限制，让我们可将系统损害限制在一定范围内。

由于 Postfix被设计成能接受有限制的运行条件，所以“拒绝服务”（Denial-of-Service，DoS）这种攻击手法不太有效。如果因受到 DoS攻击或其他原因使得系统耗光磁盘空间或内存，Postfix 会谨慎地避免雪上加霜，它会停止尝试原本设定应该运行的工作，让系统有机会腾挪出资源来复原。你可以在配置文件中限制Postfix进程可使用的磁盘空间或内存，以免在遭受猛烈攻击时变得一发不可收拾。

安全防护的困难，在于没人知道下次会遇到何种攻击手法。 Postfix 被设计成无论如何都要能够应付逆境情况。 Postfix所能提供的安全保障，主要来自它内置的稳定性。事实上，Venema 博士说，他对于安全性的兴趣，不及写出病毒软件的兴致。姑且不管实情如何，安全是软件的一个必备要求。

如何取得 Postfix

从 Postfix的正式网站可以取得更多信息。 *http://www.postfix.org/* ，这个网站提供了源程序、说明文件、附加软件的链接、文章以及关于 Postfix的额外信息。你也可以在此找到如何参与 Postfix 邮件列表（mailing list）的方法，与网络上的其他感兴趣的人一起关心 Postfix。如果你还没取得 Postfix软件，你可从该网站下载最新版的源程序。如果你没有自己编译软件的经验，该网站也提供预先编译好的包，你可以寻找看有没有适合你所使用的平台的版本。

有许多理由值得你自己编译 Postfix。首先，不一定有适合你的平台的预编包；第二，就算有，包的封装者预设的环境也不见得完全适合你的系统，而你所需的额外支持软件也不见得封装在包里。最后，预编包的版本往往是过时的，如果你想要安装最新的版本，

那就只有从源程序开始编译一遍。如果你自己有编译软件的经验，你会发现Postfix其实是最容易编译的开放源码的软件之一。

Postfix 网站上的 Download 链接会打开出一页含有世界各地镜像站点的链接页，你可以挑出离你最近的镜射站点来下载软件。 Postfix 分成两种版本，即“正式版”（Official Release）与“实验版”（Experimental Release）。虽然名为“实验”，但其实已经非常稳定。实验版含有仍在测试的新功能，这些新功能的实现细节在被纳入正式版之前，可能会有所改变。如果你需要实验版才有的新功能，你可以放心使用实验版，只不过要记得下次升级版本时，必须注意当初吸引你使用的“新功能”是否有所改变。所有 Postfix 软件都是经过严格测试与复查之后才发布的。关于实验版与正式版之间的差异，请参阅附件包的 *RELEASE NOTES* 文件。

第二章

基本概念

本章将说明 Unix 与 E-mail的基本概念，可帮助你看懂后面章节的内容与范例。如果你已熟悉邮件系统的结构，并且也有 Unix 系统管理的实践经验，大可放心跳过这一章。本章还没有论及 E-mail 或 Unix 的管理，因为这方面的信息太多了。本章的宗旨在于理清一些基本术语的定义，以便后文提到这些术语时，读者能精确知道我们想表达的意思。

Unix 的概念

显然，你对 Unix越熟悉，就越能够成为称职的 Postfix管理员。Postfix是个地道的 Unix 程序，它需要 Unix 系统的支持才能顺利发挥功能。如果你是使用 Unix 的新手，应该先研读相关入门书籍，本节的宗旨在于交代本书常用词汇的意义，而不是带你入门。

登录名称与 UID 编号

系统认识的用户名单记录在 */etc/passwd*文件，每一位用户都有自己专属的“登录名称”（login name）与“用户标识符”（User ID Number，通常简写成 uid 或 UID）。当 Unix 系统检查拥有权限与身份时，主要是依据 UID，而非登录名称。登录名称只是方便人类记忆的符号，其唯一用途是与 UID配成对，构成所谓的“标识”（identity）。Postfix 有时候会需要分辨不同用户的标识，其关于用户标识的参数是以 UID来表达，而非登录名称。当我们说某进程需要使用某“账户”的权限时，其实真正的涵义即是“标识”。

虚账户

虚账户（pseudo-account）也是正常的 Unix 系统账户，与一般用户账户之间的主要差异在于虚账户不能登录系统。虚账户主要是用于管理，或是限定程序的运行权限。你的系

统应该已经预先安装了好几个虚账户，诸如 bin、daemon 等都是常见的虚账户名称。一般而言，“不能登录”的效果，是借由刻意给一个无效的密码、不存在的主目录（home directory）或无效的 login shell 而造成的。在 Postfix 的管理实践上，你至少需要准备一个虚账户给 Postfix 进程；如果你需要运行其他配套程序，例如，邮件列表管理程序或垃圾邮件过滤程序，则可能还需要多增设几个虚账户。

标准输入／输出

Unix 系统上的几乎每一个进程，在启动之初都有一个“标准输入流”与一个“标准输出流”，通常分别简称为 stdin 与 stdout。进程从 stdin 读取数据，将数据写出到 stdout。一般而言，stdin 指键盘，而 stdout 指屏幕，也就是用户用来与程序交互的两个设备。stdin/stdout 可以被导引到其他地方，让程序可以从文件或其他程序读取数据，或是将数据写入文件或传给其他程序。Unix 系统下的工具程序，通常可将这种方式结合在一起，合力完成更复杂的工作。在邮件系统的管理实践上，服务器程序的标准 I/O 也可以连接到其他程序的标准 I/O 上以达成交互的效果。举例来说，滤信程序可能从其 stdin 得到邮件内容，将修改过的内容送到 stdout。除了标准 I/O 之外，程序通常还会有一个专门用来输出错误信息的流管道，称为“标准错误输出流”（简称为 stderr）。此流通常是接到用户的屏幕，但像 stdout 一样，也可以被转接到其他地方。如果你需要更进一步了解 *stdin*、*stdout*、*stderr*，请参阅 Unix 的入门书籍。

超级用户

每个 Unix 系统都至少有一个管理用途的账户，称为“超级用户”（superuser），其账户名称通常是 *root*。管理账户拥有系统的最高特权，相对地，破坏力也特别惊人。你应该只有在需要进行特殊管理工作时，才以 *root* 身份登录系统。管理 Postfix 时，有时候也需要 *root* 权限。事实上，如果你没有 *root* 权限，你将无法控管 Postfix。

命令行提示

当你登录 Unix 系统时，引导你操作的那个程序称为“shell”。正如其名称一样，shell 是操作系统的“外壳”，让你可以对操作系统下达工作命令。当 shell 等待你下命令时，它会显示出一段提示信息，其中可能包含你当前的身份、主机名称、当前的工作目录，还有一个不太起眼的提示符号（通常是 $、% 或 #）。详细的信息格式随你所面对的 shell 而定，一般而言，如果你当前的身份是一般用户，提示符号可能是 $ 或 %；但如果是 *root* 身份，则会看到 #。在本书的范例中，我们将沿用 bash shell 的惯例，即如果你看到 # 提示符号，表示该命令需要 *root* 权限才有效；但如果是 $，表示有一般的用户账号就足够了。

过长的文字行

在操作 Unix时，如果命令行过长（超过屏幕的宽度）时，则在末端加上一个反斜线符号（\），以表示下一行其实是前一行的延续。以\符号连接的多行文字，应该被视为逻辑上的同一行。用作延续的反斜线符号不仅可用在命令行，也常被运用在配置文件里（但是Postfix的配置文件例外，请参阅第四章）。在本书，如果我们要展示的命令无法印在同一行，也会以相同手法来处理。如果你要自己练习这些超长的命令，你可以按照书面上的格式输入（包括末端的\符号在内），也可以自己将它们凑成同一行。

在线说明书

Unix 系统的说明文件，是以“在线说明书”（manpages）的形式放在系统中，你可以用 man 命令来阅读它们。举例来说，如果你想阅读 mailq 命令的说明：

```
$ man mailq
```

接着屏幕就会出现一整页说明，按下空格键可以继续观看下一页。

在线说明书有标准的编排格式，通常一开始是命令的语法，然后是所有选项的说明，最后还有一些额外的参考信息。有人可能会觉得在线说明书很枯燥乏味，但是就像字典一样，再怎么无趣，你迟早还是要养成查阅在线说明书的习惯。所有 Unix 与 Postfix的命令都有在线说明书可供查阅。

E-mail 的概念

Internet E-mail是一个涉及许多层面的复杂主题。当你管理一个邮件系统时，不管使用哪一种 MTA 有些重要原则是不变的。本节说明这些关于 Internet E-mail 的重要的基本原则，这些概念不仅有助于你理解本书后面的章节，也有助于你从其他渠道获得更详细的信息。

RFC

RFC（Request for Comments document）定义 Internet 的各项标准。关于 Internet E-mail 的 RFC有好几个，如果你要管理 Internet 上的邮件系统，这些 RFC 不可不读。谈到 E-mail 时，最常被提到的两个 RFC 分别是 RFC 821 与 RFC 822，前者规范系统之间如何传输邮件，后者规范邮件信息本身的格式。这些规范文件都已经有 20 年以上的历史，在 2001年 4 月，它们分别被 RFC 2821 与 RFC 2822 这两个提案标准取代，不过，你仍

然会时常看到旧标准。 RFC 文档由 IETF（Internet Engineering Task Force）负责维护，他们的网站是 *http://www.ietf.org/*。

代理制度

前一章在描述邮件的传输程序时，曾经提到好几种“代理程序”（agent）。为了方便参考，表 2-1整理了这些代理程序的功能。

表 2-1 ：邮件系统的各种代理程序

简称	全名	任务
MUA	Mail User Agent	供用户写信、读信、寄信的软件。寄信时，以 SMTP协议将邮件提交给 MTA ;收信时，以 POP 或 IMAP 协议访问服务器上的邮箱。
MTA	Mail Transfer Agent	负责接收、递送邮件的服务器软件。决定邮件的递送路径，进行必要的地址改写。应该由本地系统收下的邮件，委托给 MDA 进行最后的投递操作。
MDA	Mail Delivery Agent	负责投递本地邮件到适当的邮箱。MDA可以过滤邮件内容，或是依照用户设定的准则，将邮件分类到适当的邮箱；甚至可以将邮件转回给 MTA ，以寄到另一个邮箱。

邮件管理员

负责管理邮件系统的人，通常称为“邮件管理员”（postmaster）。担任此职务的人，要承担起确保邮件系统正确运作、适应环境的改变、增加／移除邮箱账户、过滤垃圾邮件等责任。依据 RFC 2142 的规定，每一个网域都要有一个 *postmaster* 别名，此别名指向该网域实际的邮件管理员（或团队）。

“拒收”与“退信”

一个正在收信的 MTA 如果在 SMTP 对话过程中（参阅本章稍后的“ SMTP 协议 ”)决定不接受该邮件，它会当场拒收（reject）。这时候，拒收的一方，应该产生一份错误报告给寄件人。另一方面，如果 MTA 在收下邮件之后，才发现无法将邮件递交给收件人——有可能是收件人不存在或是最后的递送过程中出了问题，这时候 MTA 应该将邮件退回（ bounce）给寄件人，并随附一份错误报告，让寄件人知道他的信无法寄达以及为何无法寄达。

SMTP 协议要求收下邮件的 MTA 要负责将邮件寄达（交给 MDA）或至少由另一个 MTA 接手。如果在收下邮件之后才发现无法寄达， MTA 有义务通知寄件人，使其知道他的邮件并没有送达收件人手中。

信封地址与邮件标题

E-mail 用户常常搞不清楚一件事：邮件实际会被送到哪里，这其实与标题（header）里的 To: 后的地址无关。实际决定邮件终点站的是“信封地址”（envelope address），也就是 SMTP 对话过程中， SMTP client 以 RCPT TO 命令所指定的邮件地址（稍后的“SMTP 协议”一节会展示 SMTP 的对话过程是怎么回事）。一般而言，当 MUA 将用户写好的信交给 MTA 时，这时候确实是以用户提供的 To: 地址来作为信封地址。然而，并非所有 MUA 都如此守规矩 —— 至少，寄垃圾邮件的软件肯定不会守规矩。从 MTA 的观点来看，标题属于邮件内容的一部分，邮件会被送到哪里，决定于 SMTP 对话过程中所指定的地址 —— 也就是所谓的“信封地址”，而 SMTP server 不管邮件标头的 To: 字段指定了些什么。

邮件列表（Mailing list）与垃圾邮件，是“信封地址”不等同于“To: 地址”的两个实例。更详细的信息，请参阅 RFC 2821 与 RFC 2822 以及稍后的“电子邮件信息格式”。如果你想实验例 2-2 的对话过程，不妨试着将邮件内容（DATA 命令之后的文字段）的 To: 换成任何其他地址，看看是否会影响邮件的递送。

邮件地址的“人名部分”

邮件地址的详细格式定义在 RFC 2822，包括批注之类的东西要如何编进邮件地址， RFC 2822 都有明确规定。如果忽略不显眼的小细节，一个简单的邮件地址大致可分成三部分：local part、@ 分隔符、网域部分（domain part）。local part 通常是用户的账户名称，但也是可以代表另一个地址的别名（alias）。为了避免混淆，某些文件以“ LHS”（lefthand side）一词来代替容易令人误解的‘ local part”,而‘ 网域部分 ’则被称为‘ RHS”（righthand side）（译注 1）。

邮件信息格式

最早定义 Internet 邮件信息格式的标准是 RFC 822，符合 RFC 822 格式标准的邮件，通常简称为“ RFC 822 邮件”。你应该了解 RFC 822 格式的基本结构，因为本书会时常提

译注 1: 为了方便说明，将 local part 翻译成比较符合一般称谓习惯的“人名部分”，虽然不是很精确，但是比较方便解释。

到它，而且任何与 E-mail相关的技术文件也经常使用这个名词。我们将使用比较新的提案标准，以“ RFC 2822 邮件 ”表示。

RFC 2822邮件

RFC 2822不仅定义了邮件信息本身的格式，也规范了邮件地址在邮件标头里的格式（与“信封地址”无关）。规范文件所描述的格式是用于传输的格式，而非存储在邮箱里的实际格式，虽然有些邮件系统使用相同或类似的格式来存储邮件。一封信可分成两大部分：标头（header）与正文（body)。标头含有许多特定名称的字段，诸如 `To:`、`From:`、`Subject:`等，这些字段有一个共同特点，那就是名称之后都有一个冒号（：）；在冒号之后是字段的内容。一个标头字段的内容可以跨越好几行，开头为空格符（space或tab字符都算）的文本行，逻辑上都属于前一行的延伸。

RFC 2822详细规定了标头字段的格式与用途，有些字段甚至是彼此相关的，它们必须被一起解读才有意义。最简单的电子邮件至少要包含 `Date:`与 `From:`这两个字段，其余字段由 MTA、MUA、MDA随情况添加。除了标准字段之外，有些邮件系统也定义了自己专用的特殊字段。

标头与正文之间以一个空白行为分界。正文包含了邮件内容本身。原则上，邮件正文的格式是没有限制的，不过，为了帮助 MUA解读邮件内容，RFC 2822定义了一些标头字段来描述正文的编排格式。除此之外，邮件正文还有一项严苛的限制条件：只能包含 ASCII字符。诸如图像、16bits 字符（汉字）之类的二进制数据，必须事先以特殊编码法转换成ASCII字符，才可以编出符合标准的电子邮件。如果要夹带文件，必须以MIME或其他编码标准，将文件转换成可传输的字符。例 2-1是一封具有标头与正文的典型邮件。

例 2-1：邮件信息的标准格式

```
Return-Path: <info@oreilly.com>
Delivered-To: kdent@mail.example.com
Received: from mail.oreilly.com (mail.oreilly.com [192.168.145.34])
        by mail.example.com (Postfix) with SMTP id 5FA26B3DFE
        for <kdent@example.com>;
        Mon, 8 Apr 2003 16:40:29 -0400 (EDT)
Date: Mon, 8 Apr 2003 15:38:21 -0500
From: Customer Service <info@oreilly.com>
To: <kdent@example.com>
Reply-To: <info@oreilly.com>
Message-ID: <01a4e2238200842@mail.oreilly.com>
Subject: Have you read RFC 2822?

This is the start of the body of the message. It could continue
for many lines, but it doesn't.
```

此例中的大部分标头字段，不用解释你也知道其意义，唯一需要特别说明的是 Received 字段。RFC 2822并未硬性规定一定要有此字段，但是 RFC 2821（SMTP 协议）规定收到邮件的每一个MTA，都必须在标头顶端加上自己的 Received: 字段，以描述该封邮件传递过程中的信息。Received:字段暗藏许多玄机，请参阅下一节的解释。

SMTP 协议

SMTP 协议定义于 RFC 2821。这是一个相当容易遵守的协议 —— 不管是对人或对计算机而言，因为它的设定很人性化，也颇为宽松。 SMTP是用来送信的协议，送出邮件的一方称为“客户端”(client)，接受邮件的一方称为“服务器端”（server)。客户端要传送邮件时，必须主动连接到服务器端，并展开所谓的“ SMTP对话”，对话内容是一系列简单的命令（client → server）与响应（server → client）以及要传送的信息本身。

认识 SMTP协议的最佳方法是实际观察两端之间的对话内容。在你架设好MTA之后（或是你知道哪里有现成的 MTA）,可以使用 telnet亲身体验如何使用 SMTP协议来送出邮件。例 2-2示范了送出一封邮件的基本步骤。

例 2-2:使 用 Telnet模拟 SMTP对话过程

```
$ telnet mail.example.com 25
Trying 10.232.45.151
Connected to mail.example.com.
Escape character is '^]'.
220 mail.example.com ESMTP Postfix
HELO mail.oreilly.com
250 mail.oreilly.com
MAIL FROM:<info@oreilly.com>
250 Ok
RCPT TO:<kdent@example.com>
250 Ok
DATA
354 End data with <CR><LF>.<CR><LF>

Date: Mon, 8 Apr 2003 15:38:21 -0500
From: Customer Service <info@oreilly.com>
To: <kdent@example.com>
Reply-To: <service@oreilly.com>
Message-ID: <01a4e2238200842@mail.oreilly.com>
Subject: Have you read RFC 2822?

This is the start of the body of the message. It could continue
for many lines, but it doesn't.
.

250 Ok: queued as 5FA26B3DFE
quit
221 Bye
```

```
Connection closed by foreign host.
$
```

这个例子示范了如何使用 SMTP 将例 2-1 的那封邮件送到 *mail.example.com* 服务器上的 SMTP server （在此例中，这是一台 Postfix server）。如果想自己实验，黑体字就是你要手工键入的部分。一开始，我们使用 telnet 连接到 *mail.example.com* 的 port 25（这是 SMTP server 的公认通信端口），接着出现下列信息：

```
Trying 10.232.45.151
Connected to localhost.
Escape character is '^]'.
```

这段信息是 telnet 自己显示出来的，真正的 SMTP 对话没有这一段。在 telnet 连接成功后，出现的是 SMTP server 的响应信息：

```
220 mail.example.com ESMTP Postfix
```

SMTP server 的响应信息有固定格式，一开始必定是三个数字的“响应码”（response code），接着是一段适合人们解读的简短信息。响应信息代表前次命令的接受状态，对客户端而言，只有第一个数字有意义。表 2-2 整理了各级的响应码与它们对应的意义。

表 2-2：SMTP 的响应状态码

响应码范围	状态
2xx	请求的动作已成功接受并完成，客户端可以继续下一步。
3xx	命令不接受，因为服务器还需要更多信息。客户端应该以其他命令提供充足信息。
4xx	暂时性的失败。若客户端下次尝试同样动作，或许有机会成功。
5xx	永久性的失败。客户端不应该继续尝试同样的动作。

客户端收到欢迎语之后，接着必须使用 HELO 命令介绍它自己。HELO 命令后面必须是客户端自己的完整主机名称：

```
HELO mail.oreilly.com
```

服务器端响应成功，让客户端可以继续下一步：

```
250 mail.oreilly.com
```

客户端使用 MAIL FROM 命令表示发信者的邮箱地址：

```
MAIL FROM:<info@oreilly.com>
```

服务器端表示接受该地址：

```
250 Ok
```

客户端使用 RCPT TO命令指出收件人的邮箱地址：

```
RCPT TO:<kdent@example.com>
```

服务器端同意将邮件传递给你指定的收件人：

```
250 Ok
```

现在，客户端可以开始送出邮件的内容了。 DATA命令让服务器端知道客户端要传送封 RFC 2822 邮件：

```
DATA
```

服务器端响应它接收到的 DATA 命令，并提示如何表示邮件结尾：

```
354 End data with <CR><LF>.<CR><LF>
```

这时候，客户端可以开始传送整封电子邮件。请注意，这里说的是“整封”，包括了标头与正文。在客户端送出完整邮件之后，必须依序送出＜CR><LF>.<CR><LF>这五个字符（在画面上看起来，整列只有一个小点），让 SMTP server 知道整封信已经结束。

服务器表示它已经收到完整邮件：

```
250 Ok: queued as 5FA26B3DFE
```

从这时候开始，由服务器端负责将邮件送到目的地。如果客户端还想要执行其他命令，现在就可以开始了。如果已没有其他邮件要寄到此服务器，则应该使用 QUIT命令结束对话：

```
QUIT
```

接着，服务器便会主动切断连接：

```
221 Bye
```

最后，telnet 告诉你连接已经中断，并回到命令行环境：

```
Connection closed by foreign host.
$
```

当然，这个例子只是最简单的 SMTP对话。HELO、MAIL FROM、RCPT TO、DATA、QUIT 只是最基本的 SMTP 命令而已，还有许多其他额外命令存在。RFC 1869提出了一个架构，定义如何新增额外功能到基本的 SMTP 协议。增强版的协议称为 ESMTP。支

持 ESMTP 的客户端，可以将对话之初的 HELO 换成 EHLO，表示希望使用 ESMTP 与服务器端对话。如果服务器也支持 ESMTP，则会回答它支持哪些功能。

其他许多强化措施都定义在各个 RFC 规范文件里，你可以在 IETF 的网站（*http://www.ietf.org/*）寻找到关于 SMTP 的信息。在 Internet 上有许多关于 SMTP 与 ESMTP 协议的文章，用 Google 找一下，你会有不少收获。

第三章

Postfix 的结构

即使不清楚 Postfix 的内部组织结构，你也可以轻易地控制管理它。如果你已经迫不及待想要试用它，不妨跳过这一章，直接从第四章开始。事实上，如果没有丰富的 Postfix 使用经验，本章内容反而可能使你摸不着头绪。但是，当你累积了足够经验之后，你会发现本章能解答你心中的许多疑惑。因此，对于理论没兴趣的读者，我诚心建议你多累积一点经验之后，再回来看这一章，相信收获会更多。

Postfix 的组件

传统的 Sendmail将所有功能都集中在同一个程序里，这种结构我们称之为“单体式设计”(monolithic)。Postfix 采用专职负责的策略，不同的功能分别交由不同的专门程序处理，这种结构称为“模块化设计”（modular)。这些自成一格的专门程序，我们称之为组件(component)。大多数组件都是以 daemon 的形式存在，也就是常驻在系统内存里的连续运作的后台进程（ background process）。当 Postfix 被启动时，首先启动的是 `master daemon` ，它主导邮件的处理流程，同时也是其他组件的总管。在处理邮件的过程中，master 会启动对应功能的组件来处理相关事宜，被 `master` 启动的组件，在完成交付的工作之后会自行结束；或者，如果组件的处理时间超过时限，或是工作量到达预定限度，组件也会自行结束。 master daemon会常驻在系统中，当管理员启动它时，它从 *main.cf* 和 *master.cf* 这两个配置文件取得启动参数。关于 Postfix 的配置文件，请参阅第四章。

图 3-1从邮件处理流程来描述 Postfix 的结构。粗略来说，整个处理流程分成三个阶段：接收邮件、将邮件排入队列、递送邮件。每个阶段由一组独立的 Postfix 组件负责。当一封邮件被收下并排入队列之后，队列管理器（ queue manager）会启动适当的 MDA，将邮件送到终点。接下来的几个小节将分别讨论每一阶段的详细步骤。

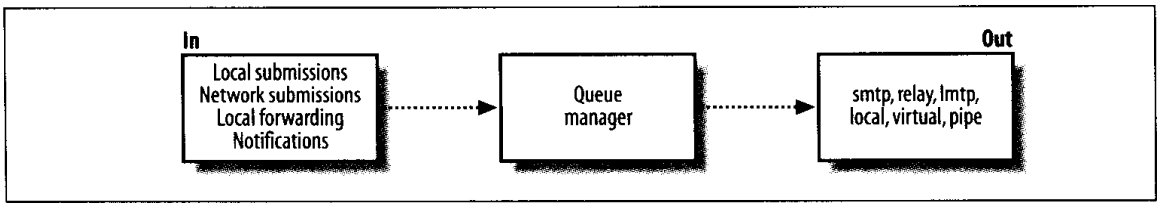

图 3-1：Postfix的邮件处理流程

邮件如何进入 Postfix 系统

邮件有四种渠道可以进入 Postfix系统：

1. Postfix可接受来自本机系统的邮件（本机用户或自主进程提交的邮件）。这类邮件的处理流程，请参阅“来自服务器本机的邮件”。
2. Postfix 可接受网络传入的邮件（来自 MUA 或其他 MTA）。这类邮件的处理流程，请参阅“来自网络的邮件”。
3. 已经被 Postfix收下并交给 MDA的邮件，被 MDA传回到 Postfix（通常是为了转寄到另一个地址）。这类邮件的处理流程，请参阅“来自服务器本机的邮件”与“转寄邮件”。
4. 当 Postfix无法将邮件寄到目的地时，自己会产生退信通知函。这类邮件的处理流程，请参阅“通知函”。

邮件有可能在进入 Postfix 之前就被拒绝了，或者因为暂时性的故障（网络断线、远程服务器响应暂时性的错误等），同样的邮件可能会每隔一段时间就重复进入 Postfix系统一次，重新递送。

来自服务器本机的邮件

各个 Postfix 组件之间的合作全靠队列（queue）交换邮件。Postfix系统有多个队列，这些队列全部由队列管理器（Queue Manager）负责控制管理。Postfix 组件可将邮件交付给 queue manager，由其代为放入适当的队列。当需要处理特定工作时，Queue Manager 将队列里的邮件交付给正确的组件。

图 3-2是本机提交的邮件在进入 Postfix系统之后的处理流程。在 Unix 系统上，当用户要寄出邮件时（不管这封信是寄到哪里），通常是使用 Sendmail 包内的 `sendmail` 命令。Postfix提供了一个与此命令兼容的同名工具，也称为 sendmail。当用户以 Postfix 的 sendmail 寄出邮件后，sendmail（Postfix 的版本）会使用 `postdrop`程序将邮件存入 Postfix队列目录下的*maildrop/*子目录。专门注意 *maildrop/*子目录有无变化的是`pickup`

daemon，每当有新邮件进入 maildrop/ 时，pickup daemon 便会读出新邮件，然后交给 cleanup daemon 进入“清理程序”。当邮件刚进入 Postfix 时，不一定含有构成有效邮件的所有必要字段，而且标头里的地址也可能需要被改写成标准格式（user@domain.tld），并依据规范的或虚拟的查询表（如果有的话）将原本的地址改成其他地址。所谓的“清理程序”就是补足遗漏的标头字段，这部分工作由 cleanup daemon 负责；地址的处理，由 trivial-rewrite daemon 负责。

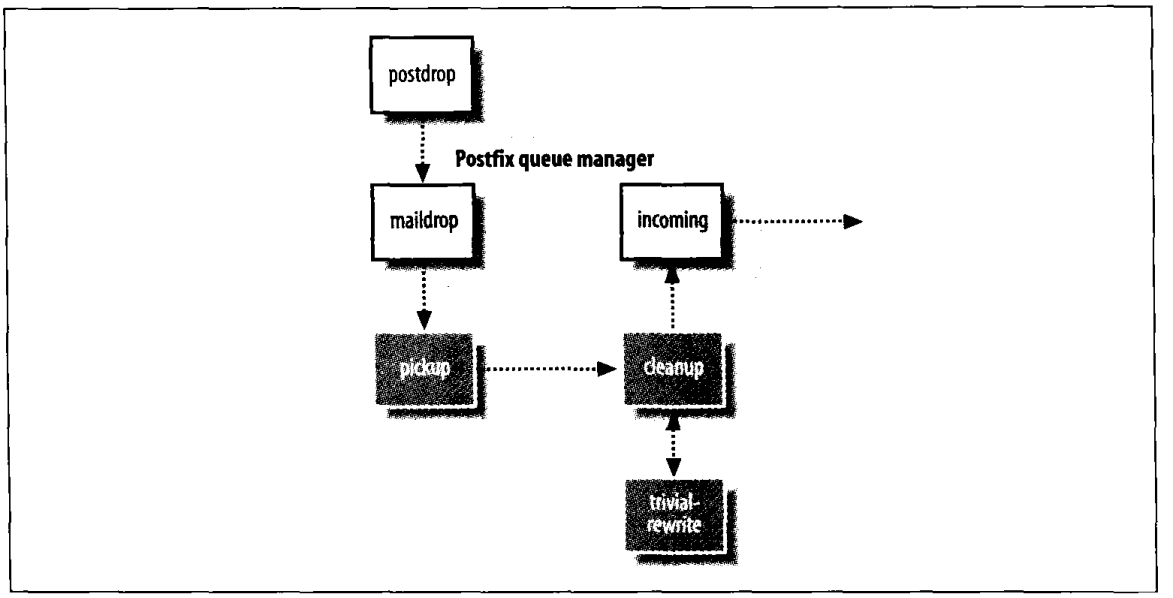

图 3-2：服务器本机用户寄出邮件时的处理流程

经过 cleanup daemon 处理好的邮件，会被传入收件队列（incoming queue）。Queue manager 会不断注意收件队列的变化，每当有新邮件进入收件队列时，便会用适当的 MDA 将邮件送到下一站，或直接送到最终目的地。

来自网络的邮件

图 3-3 描绘了 Postfix 系统从网络收到电子邮件后的处理流程。来自网络的邮件由 Postfix smtpd daemon 接收进来，然后交给 cleanup daemon 运行清理程序，随后排入收件队列，由 queue manager 选择适当的 MDA 将邮件送到下一站或最终目的地。 smtpd 有可能收到两种邮件，第一种是外界寄给 Postfix 所控制的网域的邮件（Postfix 系统本身是邮件的终点站或网关），另一种是要寄到其他网域的邮件。 smtpd 一定会收下第一种邮件（如果收件人存在的话），至于第二种邮件（目的地在其他网域），就要看传邮件过来的客户端是否有资格。网络收下要寄到其他网域的邮件，并代为寄送到目的地的动作称为转发（relay）。在两种情况下，Postfix 愿意提供转发服务：一是客户端符合配置文件

限定的资格（参阅“其他网域的邮件”一节），二是收信网域是 relay_domain 参数所列出的网域之一（参阅“转发邮件”一节）。

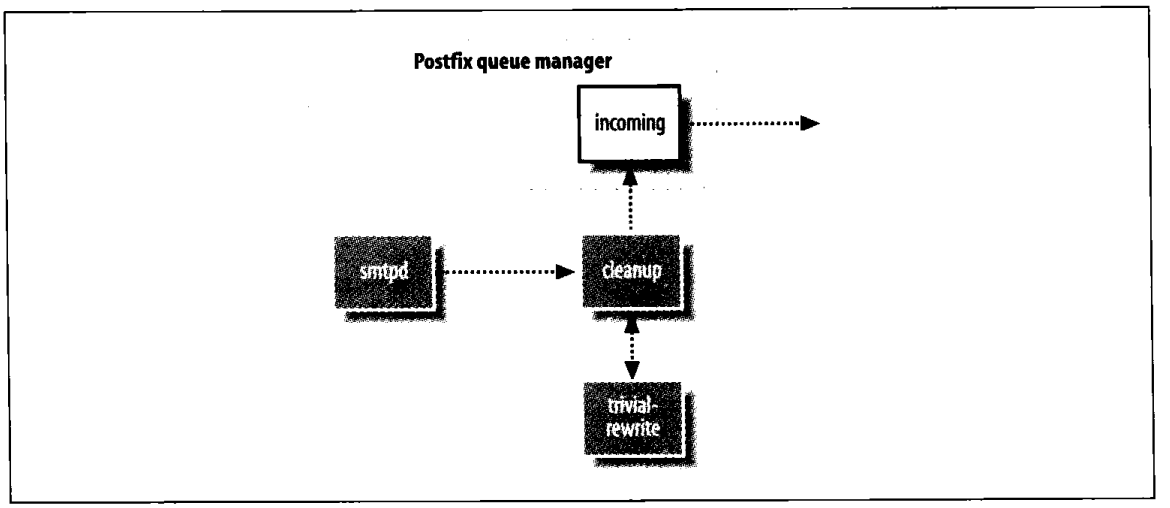

图 3-3：网络邮件的处理流程

通知函

当用户的邮件被延时，或是根本无法递送到目的地时， Postfix 使用 `defer` 或 `bounce` daemon 产生一封新的通知函。这封通知函会被交给 `cleanup` daemon，由它进行例行的清理程序之后再排入收件队列，由 Queue Manager接手处理。

转寄邮件

有时候，邮件在委托给 MDA 之后，MDA 会发现该邮件其实应该寄送到另一个系统的另一个账户。比方说，当 MDA 在用户个人的 *.forward*文件发现了另一个地址时，就会发生这种状况。照理说， MDA 可以直接使用 SMTP client（即 smtp daemon）送出邮件。不过，由于同一封信可能有多位收件人，为了照顾到每一位收件人，同时也为了在邮件日志上留下完整记录，MDA 应该依提交新邮件的方式，让邮件重新回到 Postfix系统，由 Postfix 按照“来自本机的邮件”的程序来处理（参阅“来自服务器本机的邮件”）。

Postfix 的队列管理器

邮件本身的处理工作主要是由队列管理器（Queue Manager）负责。每一个能收到邮件的 Postfix组件，都有一个共同的目的地 —— 队列管理器。邮件进入队列之前的关卡是 `cleanup` daemon ，因为只有经过清理的邮件，才有资格进入队列。`cleanup`将邮件排

入收件队列后，会通知 Queue Manager 去处理新邮件。每当 Queue Manager 察觉收信队列有新信到达时，就会使用 trivial-rewrite 来决定邮件路由信息，这些信息包括传输方法、下一站以及收件人地址。

Queue Manager 总共维护四种队列：收件（incoming）、活动（active）、延迟（deferred）、故障（corrupt）（译注 1）。新邮件通过 cleanup 之后的第一站是“收件队列”。假设系统资源空闲，Queue Manager 会将邮件移入“活动队列”，然后调用适当的 MDA 来投递邮件，而投递失败的邮件则会被移入“耽搁队列”。

如果邮件被耽搁太久，或是被判定无法送达，则 Queue Manager 还要负责协调 bounce 与 defer daemons 来产生一份递送状态报告，并传回给系统管理员或送信者（或两者）。在 Postfix 的队列目录下（默认位置是 */var/spool/postfix/*），除了上述四种邮件队列目录之外，还有 *bounce/* 与 *defer/* 目录。这些目录含有状态信息，解释特定邮件为何被耽搁或无法递送，bounce 与 defer daemon 就是利用这些目录下的状态信息来产生通知函。关于 Queue Manager 的内部运作细节，请参阅第五章。

投递操作

Postfix 依据收件地址的类型，来判断是否要收下邮件以及如何进行投递操作。主要的地址类型有本地（local）、虚拟别名（virtual alias）、虚拟邮箱（virtual mailbox）以及转发（relay）。如果收件地址不在这四种主要类型之中，则邮件会被交给 SMTP client，通过网络寄送出去（假设原信是来自有资格使用转发服务的客户端）；否则，Queue Manager 便依据地址的类型，选择适当的 MDA 来投递邮件。

本地邮件

如果收件人为 Postfix 本地系统的用户（在运行 Postfix 的那台服务器上有 shell 账户的用户），则他们的邮件会被交给 local MDA 处理。凡是收件地址的网域名称与 mydestination 参数列出的任一网域名称相符，这类邮件都算是本地邮件。对于送到任何 mydestination 网域的任何有效账户的邮件，local MDA 会先检查收件人是否有个人的 *.forward* 文件，如果没有，则邮件会被存入用户的个人邮箱；否则，则依据 *.forward* 文件的内容来进行投递操作（或转寄到其他地方，或是交给外部程序处理）。关于本地邮件的投递操作的详情，请参阅第七章。

译注 1：Postfix 系统还有一个“保留队列”（hold queue），作者这里没提出来，是因为此队列不是靠 Queue Manager 来自动维护，而是由邮件管理人员手动控制管理。比方说，将可疑的垃圾邮件暂时留在系统，使之不流入用户的邮箱（参阅第十一章）。

对于需要转寄到他处的邮件，将会被重新提交回 Postfix，以便传送到新地址。如果递送过程发生了暂时性问题， MDA 会通知 Queue Manager，而邮件会被保存在延迟队列等待下次的递送机会；如果发生永久性问题，则要求 Queue Manager将信退给发信者。

虚拟别名邮件

寄给虚拟别名地址的邮件，全部都需要转寄（forward）到其真实地址。虚拟别名的网域名称列在virtual_alias_domains 参数中，每一个虚拟网域都可以有自己的一组用户，不同的虚拟网域可以容许有同名的用户。用户与其真实地址之间对应关系，列在 virtual_alias_maps 参数所指定的查询表中。当 Queue Manager发现邮件的收件地址的网域部分是 virtual_alias_domains所列出的网域之一，则会重新提交邮件，以便传到真实地址。关于虚拟别名，请参阅第八章。

虚拟邮箱邮件

虚拟邮箱的网域名称列在virtual_mailbox_domains参数中。每一个网域都可以有自己的用户群，而且有各自独立的命名空间，换言之，即不同的虚拟邮箱网域可以有同名的用户。用户与邮箱之间的对应关系，定义在 virtual_mailbox_maps参数所指定的查询表中，虚拟邮箱与系统上的shell账户之间没有关联性。虚拟邮箱邮件的投递操作由 virtual MDA 负责运行。关于虚拟邮箱，请参阅第八章的说明。

转发邮件

实际邮箱位于其他 MTA控制管理的网域、但是 Postfix愿意代收并转寄的邮件，称为转发邮件。这类网域的名称列在 relay_domains参数中，其邮件由 smtp MDA 通过网络送到目标网域的 MTA。当你架设邮件网关系统时，便可利用转发功能收下Internet寄到内部网域的邮件，并转寄到内部网络的邮件系统上。关于转发操作的细节，请参阅第九章。

其他邮件

如果邮件的收件地址不属于前述四类，则一律交给 smtp 以递送到正确地点，因为这类邮件必定是要送到系统本身之外的其他网域。先前在“来自网络的邮件”曾说过，并非所有客户端都有资格使用转发服务。一般而言，我们会将转发服务开放给与 Postfix server位于相同局域网络的其他主机，好让这些主机可通过Postfix server寄信到Internet上的外界网域。

当smtp MDA收到外地邮件时，它会依据收件地址来判断邮件应该送到哪个（或哪些）主机，然后依序连接到这些主机，直到有一部主机愿意收下邮件为止。如果投递过程发生暂时性问题，smtp会通知 Queue Manager将邮件放在延迟队列，等待下次传送的机会；如果发生永久性错误，则要求 Queue Manager退信给发信者。

当因暂时故障不能连接的远程主机恢复连接时， Postfix会先采取试探性动作，以免过多的延迟邮件使对方瘫痪。一开始， smtp只会搭建有限度的几条连接通道（数量可通过配置文件调整）到收件方，在发现对方能够成功收下邮件之后，才会慢慢提升连接通道的数量（到一个可设定的上限）；相反地，如果 smtp发现接收方有任何麻烦，它会立刻撤销连接。

其他传送代理程序

Postfix 还提供了其他 MDA，可用来处理特殊的地址或目的地。这些 MDA 需要在 *master.cf*配置文件中设定妥当之后才有作用。此外，你还必须在 *class*_transport 或 transport_maps参数指定的传送表（transport table ）中设定如何启动它们。最常用的两个特殊 MDA是 lmtp与 pipe。

LMTP 投递

LMTP 是一种很类似 SMTP 的协议，但只能用于同一网络上的邮件系统之间，或是同一主机上的不同邮件程序之间（关于 LMTP 的细节，请参阅第七章）。举例来说，如果需要将邮件送到不同的软件包——该软件可能与 Postfix在同一台机器上，也可能位于局域网络上的另一台主机，则 Queue Manager 可以调用 lmtp MDA将邮件传给该软件包。实际上，LMTP最常被用来将邮件交给特殊的 POP/IMAP server ，以便使用特殊邮箱格式来存储邮件（注意， POP与 IMAP 都是让用户用来取信的协议）。在这种情况下，由于只有 POP/IMAP server 知道特殊邮箱格式，所以只好使用标准的 LMTP 来交付邮件。如果 LMTP投递过程中出了任何问题， lmtp MDA会通知 queue manager将邮件放在延迟队列，等待下次传送的机会。

Pipe 投递

Postfix 提供了 pipe daemon 将邮件传给外部程序。实际上，pipe经常被用来将邮件传给外部的内容过滤程序（例如：病毒扫描系统、垃圾邮件分析程序）或其他通信媒介（例如：传真机）。同样，如果 pipe无法顺利传出邮件，它会通知 queue manager将邮件放在延迟队列，等待下次传送的机会。

实际追踪 Postfix 的邮件处理流程

让我们追踪一封典型邮件如何通过 Postfix 系统。图 3-4、图 3-5 与图 3-6描绘的处理流程，是一封邮件从发信方到达目的地（信封地址所指的 MTA），然后又被转寄到最终的 MTA（收件人实际取信处）。图 3-4中，Helene（*helene@oreilly.com*）想要送一封信给 Frank（*frank@postfix.org*）。Helene 的账户位于一台运行 Postfix的服务器。她使用自己习惯的MUA 编写邮件，然后调用 Postfix的 `sendmail`命令送出邮件。Postfix的 sendmail 程序从 Helene 的 MUA 软件中收下邮件，然后放在队列的 *maildrop/* 子目录下。接着，`pickup` daemon 从该目录取出邮件，交给 `cleanup` daemon 运行必要的清理程序，如果 Helene的 MUA 软件没提供地址 `From:`，或是该地址没使用完整的主机名称，则 `cleanup`会主动补齐不足的信息以确保邮件格式符合标准。

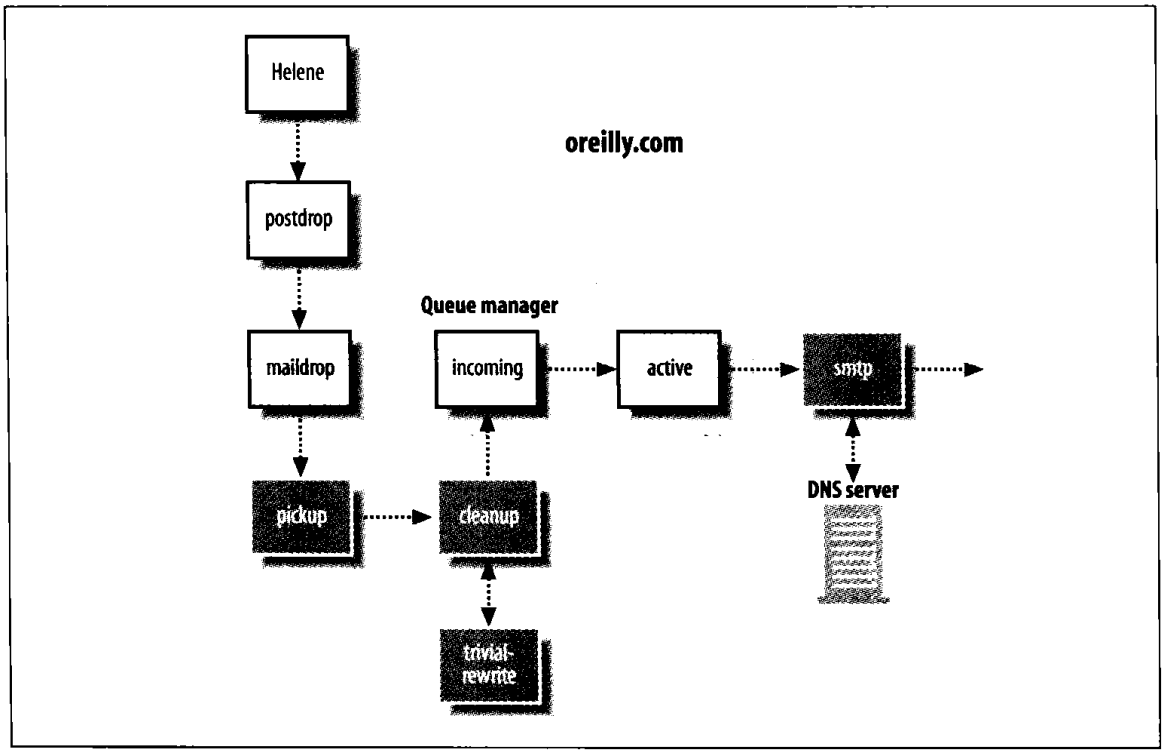

图 3-4：Postfix的邮件处理流程：送出邮件到其他 MTA

完成清理程序之后，`cleanup`将邮件存入收件队列，并通知 queue manager，使其知道有一封新信正在等待投递。如果 queue manager 已经准备好处理新邮件，它会将邮件搬移到活动队列。由于 Helene的信是要送到其他网域系统的用户，所以 queue manager 使用 `smtp` MDA 来投递该邮件。

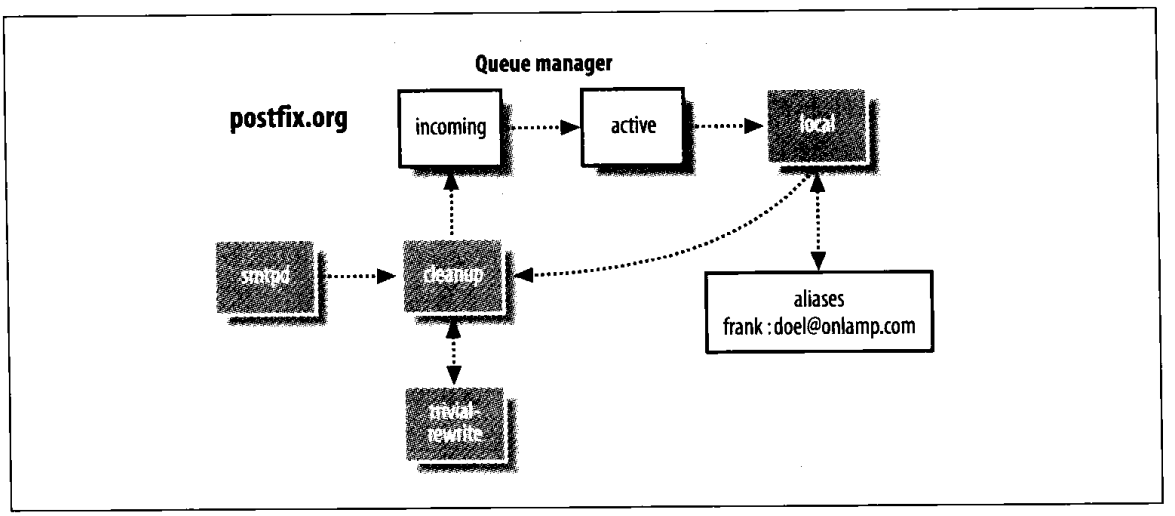

图 3-5：Postfix 的邮件处理流程：从其他 MTA 接收邮件再转寄出去

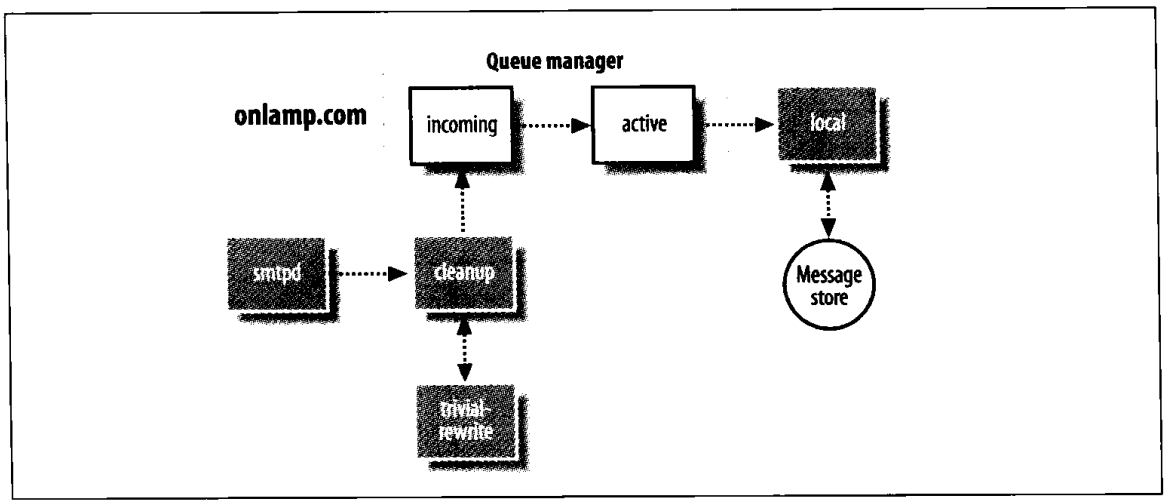

图 3-6：Postfix 的邮件处理流程：从其他 MTA 接收邮件并存入邮箱

`smtp` 使用 DNS （参阅第六章）查出哪些邮件服务器愿意收下 *postfix.org* 网域的邮件，然后从中挑出最优先的邮件交换主机（MX host）并与该主机接洽，以 SMTP 协议送出 Helene 写的邮件。

图 3-5 显示 Frank 在 *postfix.org* 网域的邮件服务器同样也是运行 Postfix （其实，任何符合标准的 MTA 都可以），则在 Frank 的服务器上的 Postfix `smtpd` 收下 Helene 的 `smtp` MDA 送来的邮件。当 `smtpd` 确认它应该收下该邮件之后，它会将收到的信交给 `cleanup` daemon 进行检查，然后存入收件队列。

Queue manager 将邮件搬移到活动队列，检查收件人地址，然后使用 local MDA 来进行

投递操作。接着，`local`发现收件地址*frank*其实是一个别名，其真实地址位于另一个网域，所以将邮件与新地址信息传给 `cleanup` daemon，回到 Postfix的队列系统。

当`cleanup`与queue manager处理邮件时，依靠 `trivial-rewrite` 将地址转换成标准格式，并判断传送方式以及递送流程的下一站。

当 queue manager发现新邮件应该送到另一个网络（此例中的 *onlamp.com*），则会调用`smtp`来进行投递操作，而 `smtp`会先向 DNS查出哪些邮件服务器可能接收 *onlamp.com* 网域的邮件。图 3-6是 *onlamp.com* 的 MTA系统（它也是 Postfix系统），该系统最后会将邮件交给 `local` MDA，由它将邮件存入该系统的邮箱。到这时候， Postfix就完成了它的工作。Frank现在可以使用他自己的MUA来阅读邮件，至于这个MUA是直接从邮箱取信，还是使用 POP或Imap之类的协议通过网络下载邮件，都已经不是Postfix所能控制的了。

我们的例子只假设了最理想的简单状况，实际的传送程序可能会遇到一些意外状况，诸如网络临时断线、远程主机死机、邮箱空间不足等。发生意外时， MDA必须通知 queue manager，将邮件暂时放回延迟队列，等待一段时间之后再重新投递。除了临时意外会影响投递过程，还有一些情况也会影响，比方说， *doel*并非实际的系统账户，而是 Imap邮件系统的一个账户。在这种情况下， Queue manager 可能要通过`lmtp` MDA来投递邮件，或是通过`pipe` MDA将邮件送到一个事先确定的外部程序。

Postfix还要面对各式各样的变化与潜在的复杂性，幸运地，它本身的结构设计足够稳定，几乎能够应付所有可想象到的情况，也有足够的可塑性来适应未来的可能变化。

第四章

基本的配置与管理

Postfix 最令人印象深刻的特性之一大概是它的“即装即用”性。通常，在完成安装工作之后，只要稍微修改一下配置文件（甚至不必修改），它就可以立刻开始运作了。本章第一节解释初期的配置工作，并示范第一次如何启动 Postfix。以后的小节讨论 Postfix 配置的细节。Postfix 的默认运作模式，扮演传统的 Unix 邮件服务器，为本地系统的所有账户收发邮件。这表示每一位用户都必须有邮件服务器的 Unix shell 账号，而且只能使用服务器上的 MUA 软件来收发邮件。

很显然，默认的运作模式通常不能满足你。在大多数情况下，你会希望 Postfix 能够搭配其他软件，构筑出一套完整的邮件系统。当你安装新的配套软件时，应该先个别独立测试，并且在集成进系统之后再测试一次。

在正式开始之前，我们假设你已经将 Postfix 安装到你的系统上了，你可以使用别人预先编译好的包（*.rpm*、*.deb*、*.pkg* 等）；或是从 Internet 下载最新版源程序（*.tar.gz*），自己编译、安装；或者直接使用操作系统安装程序自动帮你预装好的版本。如果你还没安装 Postfix，请参阅附录三，依照指示步骤将 Postfix 装到你的系统上。完成安装后，参阅本章完成后续的配置工作。

为了方便说明，本书范例假设你将 Postfix 安装在下列默认目录：

/etc/postfix/
: 配置文件与查询表

/usr/libexec/postfix/
: Postfix 的各个服务器程序

/var/spool/postfix/
: 队列文件

/usr/sbin/

Postfix的工具程序

此外，本章还假设你已经创建一个*postfix*虚账户与一个*postdrop*组，而且这组账户与组专供 Postfix 系统使用，不做任何其他用途。如果你要改变任何默认条件，或是你的Postfix包使用的默认值不同于本书的假设，当你沿用本书的范例时，请记得依照实际情况调整。

第一次启动 Postfix

第一次启动 Postfix 之前，必须先完成两项准备工作。第一件工作是确定系统本身的标识，也就是你的机器在 Internet 上的完整主机名称（Fully Qualified Hostname），因为 Postfix 的 myhostname配置参数需要设定成这个名称。只要Postfix知道系统的完整主机名称，它就可以推算出其他重要参数的默认值，例如 mydomain。如果你没明确设定myhostname参数，Postfix的默认值是使用本机系统回复的结果。本章稍后还会详细讨论 myhostname参数。现在你只要知道，使用 Unix 的 hostname命令可查出系统的主机名称：

```
$ hostname
mail.example.com
```

完整主机名称由两部分构成，在第一个点号之前的部分是主机本身的名称（即“简名”），其后是“网域名称”。如果你的系统已设定好完整主机名称，就可以不修改 myhostname参数。然而，有些系统只设定了简名，而没有设定完整名称：

```
$ hostname
mail
```

在这种情况下，Postfix 无法自己推算出完整的网域名称，所以你必须先明确设定myhostname参数。Postfix 提供了一个专门用来设定配置参数的工具程序： postconf，利用此工具，你可以查阅或修改 Postfix系统的各项参数的设定值。例如，若要设定myhostname参数：

```
# postconf -e myhostname=mail.example.com
```

“-e”选项是“编辑”(edit) 之意，即使 postconf 将指定的参数设定成 =之后的值。

启动 Postfix之前的第二项准备工作，是确定系统的别名数据库（aliases database)的格式是否正确。在你的邮件服务器被使用之前，必须事先设定几个必要的“别名”（如postmaster、info等），好让别人可找到到相关程序的管理员。我们稍后会讨论 *aliases* 文件的详细格式，现在，你只要知道它是一个文本文件，可用来产生适合查询的索引数据

库。你的系统上现有的别名数据库不见得符合 Postfix 的默认格式，所以必须使用 newaliases 命令重建别名数据库：

```
# newaliases
```

这个命令不需要任何自变量，它不会修改你的 aliases 文本文件，但会依据文本文件的内容重建新的别名数据库文件。

完成了上述两项准备工作之后，就可使用下列命令来启动 Postfix：

```
# postfix start
```

如果 Postfix 在启动过程中遇到问题，它会回复错误信息——可能出现在终端机画面，也可能被记录到系统上的邮件日志（随实际的 Unix 系统环境而定）。无论如何，只要能够顺利启动，Postfix 会立刻“脱离”终端机，将所有信息导入系统邮件日志（通常是 */var/log/maillog*），也就是说，在终端机画面上看不到任何信息。所以每次你启动或是重新加载 Postfix 时，请务必检查日志文件，看看有没有任何警告或错误信息。关于 Postfix 的日志记录功能以及如何找出它的日志文件，请参阅本章的“日志记录”。

绝大部分情况下，Postfix 都可以顺利启动而不遇到任何问题，所以，你现在应该已经成为一台正在运作中的全功能 Postfix server 的名誉管理人。但是，在你真正担任此一职务之前，还必须为你的用户做两件事。第一件事，让你的用户不需用邮件服务器系统的 shell 账户也能收信。这件事请参考第七章，即如何设定 Postfix 来搭配 Cyrus POP/IMAP server。第二件事，确定远程系统确实能通过 DNS 找到你的 Postfix server。关于 DNS 的事项，请参阅第六章。如果你想停止或重新启动 Postfix，请参阅本章的‘Postfix 的启动、停止与重新加载”。本章的其余内容，重点放在 Postfix 的配置与管理上。

配置文件

Postfix 的配置文件全部都集中在 */etc/postfix/* 目录下，其中最重要的两个文件是 *master.cf* 与 *main.cf*。为了安全起见，这两个文件的拥有者都应该是 root，而且只有 root 拥有 write 权限，但是其 read 权限可以开放给任何人。每当你改变这些文件的内容后，都必须重新加载 Postfix 一次，则所做的改变才会生效（注 1）。

```
# postfix reload
```

注 1：如果你修改 inet_interfaces 参数，则需要重新启动（restart）Postfix，而不可只重新加载（reload）。

诚如第三章所述，整个 Postfix 系统由多个具有不同功能的 daemon 构成，这些 daemon 之所以能够相互配合，是依靠 master daemon 居中控制管理。master 依据 *master.cf* 配置文件来决定如何启动其他 Postfix daemon。*master.cf* 配置文件里的每一行分别描述一种 Postfix 服务或传送程序的工作参数，参数行有统一的格式，各字段的意义都有规定，所以各参数必须按照顺序排列。关于 Postfix 的组织结构以及各种 daemons 之间的交互关系，请参阅第三章。关于 *master.cf* 的细节，请参阅本章“ master.cf 配置文件 ”。一般而言，默认的 *master.cf* 足以应付大多数状况，所以你通常不必修改它。

main.cf 配置文件

Postfix 的主要配置参数都集中在 *main.cf* 文件中，几乎所有配置参数的改变都是发生在这个文件。默认的 *main.cf* 大约包含三百个参数，虽然大部分的参数都不必修改，但是它们也提供了让你自主调整的空间，当你需要修改 Postfix 的行为时，这些预列的参数提供了相当大的灵活性。Postfix 的配置参数相当多，为了方便查找，不同用途的参数被分别放在不同的样本配置文件。这些样本文件存放在 sample_directory 参数所指定的目录下（通常与 *main.cf* 文件位于相同目录）。Postfix 包随附的 *main.cf* 与样本文件都含有详细的注释，这些注释有助于你了解各个参数的意义、语法以及注意事项。

注意：在本书，当我们提到“ 修改……参数 ”时，除非特别注明，否则就是指 *main.cf* 配置文件里的参数。

Postfix 提供了一个让你可在命令行中编辑修改 *main.cf* 配置文件的工具 —— postconf，前一节已示范如何使用它来修改 myhostname 参数。事实上，只要你遵守 Postfix 的配置文件的语法，任何你惯用的文本编辑程序（例如 vi、emacs、jed、pico 等)(注 2) 都可以用来直接编辑 *main.cf* 或任何配置文件。Postfix 配置文件的内容可以包含空白行、注释行、参数行。空白行的含义非常明白不需赘述。注释行是开头字符为 # 的文本行，如果有多行注释，则每一行开头都要有一个 # 符号。对软件而言，空白行与文字行没有作用，Postfix 只会解读参数行；但对管理人员而言，它们有重要的参考价值。每项参数都至少有一个参数名称，其后是一个＝符号，然后是一个或多个参数值。简单来说，参数行的格式如下：

```
parameter = value
```

注 2: Postfix 假设它的配置文件都符合 Unix 的文本文件惯例，也就是以一个 \n 字符代表换行。如果你使用 Windows/Mac 的文本编辑工具来编辑 Postfix 的配置文件，记得要转换成 Unix 的格式（Linux 系统提供了 dos2unix 与 mac2unix 工具来帮你转换）。

在＝符号左右两边的空格是可有可无的。

以下是两行注释、一行空白与一行参数的组合：

```
# myhostname的值必须是完整的主机名称
myhostname = mail.example.com

# 以下是 main.cf 配置文件的其余内容…
```

最常犯的一种错误，是将注释与参数放在同一行。这类错误有时候很难追查，因为你认为是注释的部分，可能被 Postfix 当成参数值。以下是一个错误示范：

```
#
# 这是一个错误示范，注释不能出现在参数行
#
myhostname = mail.example.com    # 必须为完整主机名称
```

另一种常犯的错误，是在参数值的前后加注引号。不管是单引号还是双引号，在 Postfix 配置文件里都没有任何特殊意义，所以它们也会被当成参数值的一部分，而不是你想要的结果。

多行参数值

第一个字符为空格(tab 或 space）的文本行，被视为前一行的延续。当参数值太多而无法放在同一行时，这种格式让我们可将参数值分散在不同行。例如以下的参数行：

```
mydestination = example.com oreilly.com ora.com postfix.org
```

其效果与下面的设定方式相同：

```
mydestination = example.com
  oreilly.com
  ora.com
  postfix.org
```

配置变量

在参数名称之前加上一个 $ 符号，表示引用该参数的设定值。例如：

```
mydomain = example.com
myorigin = $mydomain
```

系统即会将 mydomain参数的值设定给 myorigin参数。这种语法的好处是你只要改一个地方，其他相关参数即可统一修改，而不会遗漏。

被引用的参数并不一定要先定义，即顺序颠倒也无所谓。例如，下面的写法与前述例子同样有效：

```
myorigin = $mydomain
mydomain = example.com
```

多参数值

有许多参数可以同时拥有一个以上的值。参数值之间可以用逗号（,）、空格(space与tab）或换行字符隔开。记住，当你将参数值放在不同行时，参数值之前必须要有空格，以表示它们与前一行是连续的。例如，以下三种语法是等效的：

```
mydestination = $mydomain, example.com, oreilly.com
mydestination = $mydomain example.com oreilly.com
mydestination = $mydomain
    example.com
    oreilly.com
```

有时某些参数的值太多以至不适合直接放在 *main.cf*配置文件里，则可将参数值写在另一个文本文件，而把文件名提供给参数。任何以／字符开始的字符串，都会被视为文件名。举例来说，假设我们的系统要受理来自许多网域的邮件，则可以将这些网域名称写在另一个文本文件，然后让 mydestination参数指向该文件：

```
mydestination = /etc/postfix/destinations
```

原则上，可接受多个参数值而且参数值顺序不甚重要的参数，就可以使用外部文件。例如 mynetworks、mydestination 和 relay_domains都可以接受外部文件。如果不确定特定参数是否支持外部文件，请查阅说明文件。

如果参数值多达上千个，或是参数值本身具有“ key-value ”的形式，这时候使用查询表（lookup table）会是比较好的选择，因为查询表有较高的查询效率。关于查询表，我们在下一节再做解释。

每次修改 *main.cf*之后，都必须重新加载 Postfix 才能使改变生效：

```
# postfix reload
```

关于 Postfix程序运作状态的控制，请参阅本章的“ Postfix 的启动、关闭与重新加载 ”。

查询表

MTA时常需要做各式各样的转换与查询，比方说，改写邮件地址、判断客户端是否来自授权网络等。对于这类工作， Sendmail要使用复杂的改写规则或样式转换，而 Postfix采用比较简单也比较有灵活性的查询表（ lookup table）来实现。有许多重要参数都需要能够查询其对应关系，例如，用于改写邮件地址的 canonical_maps参数。举例来说，有一家公司，他们内部使用账户名称来做邮件地址，但是他们希望公开给外界的任

何地址，都必须是一致的 *first.last@example.com* 格式。因此，若某人的实际邮件地址是 *kdent@example.com*，则外界看到的邮件地址应该是 *kyle.dent@example.com*。很显然，因 *main.cf* 本身的格式限制，我们无法描述这样的对应关系。在这种情况下，最好的办法就是将所有对应关系另外写在一个文件 —— 也就是“查询表”中，而 *main.cf* 中的 canonical_maps 参数只要指向这个文件即可。

查询表让 Postfix 可从一个“索引键”（例如 *kdent@example.com*）查出其“对应值”（*kyle.dent@example.com*）。索引键（key）通常来自 Postfix 于运行时得到的信息，如邮件里的邮件地址、客户端的 IP 地址或主机名称等；而对应值（value）可能是一段信息（例如邮件地址），也可能是影响邮件处理流程的动作关键字（例如 REJECT）。有许多 Postfix 参数都需要查询表，维护查询表是管理员的重要职责之一，所以有必要详加说明查询表的文件格式。

查询表的格式

Postfix 支持多种格式的查询表，最常用的格式是 Unix 数据库文件，这是一种含有索引表的二进制文件，特别适合用于快速的“key-value”查询。查询表的原始数据来自简单的文本文件，文本文件里的每一行定义一组“key-value”对应关系，key 与 value 之间以空格隔开：

```
#
# canonical 查询表
# 格式：
#   key    value
kdent@example.com      kyle.dent@example.com
```

同一个查询表里的每一个 key，都必须独一无二，不可重复。“key”通常被称为 LHS（左手边），而“value”被称为 RHS（右手边）。查询表的 LHS 不区分大小写。如同 *main.cf* 的格式，查询表也可以包含空白行与注释行，而且开头为空格的行也被视为前一行的延伸；引号在查询表里也同样没有特殊意义。

将所有对应关系都编入文本文件之后，必须使用 postmap 来创建做实际查询之用的数据库文件：

```
# postmap /etc/postfix/canonical
```

每次修改文本文件之后，都必须运行 postmap 来重建它的数据库文件。

postmap 不仅可以用来创建数据库文件，也可以用来查询。使用“-q”选项指定你要查询的 LHS，它会显现出对应的 RHS：

```
# postmap -q kdent@example.com /etc/postfix/canonical
kyle.dent@example.com
```

数据库格式

不同类型的Unix数据库文件，有不同的内部格式。你的系统支持哪些类型的数据库，要看你的系统上安装了哪些数据库的函数库。通常， Postfix至少支持btree、dbm 与hash这三种类型中的一种，不过，你的系统实际支持的类型可能更多。搞清楚你的系统支持哪些类型的数据库是非常重要的。 postconf的“ -m”选项，可帮助你搞清楚你的Postfix支持哪些类型的查询表：

```
$ postconf -m
static
pcre
nis
regexp
environ
proxy
btree
unix
hash
```

此命令会显现出你的Postfix支持的所有查询表类型以及各种形式的信息变量（例如，代表环境变量的environ）。但无论如何，你应该至少能找到btree、dbm或hash这三种数据库类型中的一种。

Postfix默认的数据库类型，定义在default_database_type 参数：

```
$ postconf default_database_type
default_database_type = hash
```

注意：若没有特别声明，本书所有范例都使用 hash类型。如果你的系统使用其他类型，当你实验我们的范例时，请自行调整。

使用 postmap 时，如果没特别指定某一数据库类型，它会将查询表（文本文件）转换成default_database_type参数所指定的数据库类型。然而，当你将查询表赋值给相关参数时，你必须指出查询表的数据库类型。

当你将一个查询表赋值给某参数时，必须同时注明查询表的“ 类型 ”与“ 名称 ”。指定查询表的语法如下：

```
parameter = type:name
```

这里的 *type* 是变量的访问方法（postconf “-m” 所列出的项目之一），而 *name* 是含有“key-value ”对应关系的资源的名称。对于索引式数据库的查询表，*name* 等于查询表的完整路径名。以 canonical_maps 为例：

```
canonical_maps = hash:/etc/postfix/canonical
```

你可以同时指定多个查询表给同一个参数。 Postfix 会依照你给定的顺序来搜寻查询表，并且在找到符合条件的项目时立刻停止。某些参数支持“递归式查询表”（recursive lookup table），如 canonical_maps 参数。使用递归查询时，系统一旦找到一个符合条件的对应值，Postfix会尝试将该值与所有的索引键再比对一次，直到找出一个相符的索引键。举例来说，若 canonical_maps 参数所指的查询表内容如下：

```
# recursive canonical maps
kdent@example.comkyle.dent@example.com
kyle.dent@example.com     dent.kyle@example.com
```

那么，当 Postfix 查询“ kdent@example.com ”时，最后得到的对应值是“dent.kyle@example.com”。

你或许已经注意到，使用 postmap 编制索引文件之后，它会创建额外的文件。你可能会发现系统上多了一个 *.db* 文件，或是多了一对 *.dir* 与 *.pag* 文件（随你所用的数据库类型而定）。注意，指定查询表给参数时，只需指出查询表的类型以及文本查询表的完整路径名称即可 —— 不需包含扩展名，因为扩展名是由数据库系统决定。

比对顺序

有许多参数的查询表，其索引键为邮件地址 —— 有可能是完整地址，也可能只有人名部分，或是只有网域部分。例如 canonical_maps参数的查询表，其索引键只有人名部分有意义；而 transport_maps参数的查询表，其索引键只有网域部分有意义。对于不同类型的查询表，Postfix采取的比对顺序也略有不同。如果你想知道特定参数的查询表的比对顺序，请参阅相关查询表的在线说明。

原则上，对于 canonical_maps、relocated_maps 与 virtual_alias_maps这种只有人名部分有意义的参数而言，其查询表的比对顺序如下：

1. 完整的地址，例如 *kdent@example.com*。
2. 只含人名部分的地址，例如 *kdent*。
3. 以 @ 开头的网域部分，例如 *@example.com*。

另一方面，对于只有“网域部分”才有意义的参数，例如 transport_maps，Postfix 采用以下比对次序：

1. 完整的地址，例如 *kdent@example.com*。
2. 只有网域本身的地址，例如 example.com。

3. 以“ .”字符开头的网域地址，例如 .example.com。这表示任何子网域都符合条件。

如果你希望网域名称涵盖其下的所有子网域（比方说，“example.com”代表example.com与其所有子网域），你可以利用 parent_domain_matches_subdomains 参数来简化查询表的内容。此参数的默认值包含许多查询表。举例来说，如果你希望可以在 transport_maps 查询表使用这种简化语法，方法如下：

```
parent_domain_matches_subdomains =
   debug_peer_list
   fast_flush_domains
   mynetworks
   permit_mx_backup_networks
   qmqpd_authorized_clients
   relay_domains
   smtpd_access_maps
   transport_maps
transport_maps = hash:/etc/postfix/transport
```

现在，*/etc/postfix/transport* 查询表里的网域地址部分，会自动匹配网域本身与其所有子网域，用不着另外增加“代表所有子网域”的键一值对。

查询表与简易列表

某些参数通常只需要一份简易列表，但是也容许你指定一个查询表，例如 mydestination。使用简易列表时，只需要列出索引键即可；如果使用查询表，则必须为每一个索引键都配对一个（无意义的）对应值。通常，只有在你想写给自己看的注释时，或者列表里有上千个项目的时候，你才要这样做。将长串列表独立到一个简单的文字文件，不仅可简化配置文件，在效率上也比较理想。举例来说，如果你的 Postfix 要服务的网络客户（定义在 mynetworks 参数）不是集中在某个子网络（不能用简单的 network/mask语法来表示），而必须分别列出个别网络客户的 IP 地址时，查询表可能会是比较理想的选择。你如想知道哪些参数可同时支持简易列表与查询表，请参阅 Postfix 的说明文件。

模式表

除了支持与 key-value配对关系的查询表之外，Postfix还支持一种更有灵活性的特殊表，让你可以用正则表达式（regular expressions）来描述一组共同特征。正则表达式又称为“模式”（pattern），许多 Unix 工具应用它们来比对“具有描述的共同特征”的字符串。Postfix 支持两种风格的正则表达式，随你的系统所安装的函数库而定。

默认情况下，Postfix 使用“ POSIX扩展正则表达式 ”,我们简称为“ regexp”。POSIX的全名是Portable Operating System Interface，这是为了促进操作系统兼容的标准，regexp

是其中关于正则表达式语法的标准规范。 Postfix 也支持“ Perl 兼容正则表达式 ”（Perl-compatible regular expression），简称为“ pcre”。如果你习惯使用 Perl 风格的正则表达式，你会发现它与 regexp的语法有点小差异。如果你希望 Postfix支持 pcre ,在编译Postfix 时，必须链接 pcre 函数库才行。pcre 不仅在语法功能上不同于 regexp，在效率上也略胜 regexp 一畴。如果你不确定自己系统上的 Postfix 是否支持 pcre ,请使用 `postconf` 的“`-m`”选项来检查。

倘若 pcre 出现在 postconf “`-m` ”显现出的支持类型中，则你的模式表可以使用 Perl 风格的正则表达式。但如果你的系统不支持 pcre ，那也不必急着重新编译 Postfix,因为默认的 regexp已经足以应付管理员在正则表达式方面的需求，除非你确实需要 Perl正则表达式特有的功能。

对 POSIX 与 Perl两种风格的正则表达式，都可以找到很详尽的说明。任何关于 Perl的书籍，都应该会提到它的正则表达式；如果你的系统上已经安装了 Perl ,请参阅 `perlre(1)` 在线说明书。关于 regexp的说明，通常出现在 `re_format(7)` 在线说明书，如果你的系统上没有这份文件，请去 Internet上搜索。此外，Dale Dougherty 与 Arnold Robbins 合著的《sed & awk》（O'Reilly 出版）也提供了 POSIX 正则表达式的解说与范例。

要使用模式表，除了要指定文件名之外，还必须注明其表示法为 `regexp` 或 `pcre`：

```
body_checks = regexp:/etc/postfix/re_body_checks
```

模式表（此例中的 `re_body_checks`） 里的每一行，都分成“ 模式 ”与“ 值 ”两部分。模式是放在一对“ ／ ”符号之间的 regexp或 pcre 正则表达式，“ 值 ”是该正则表达式所对应的字符串。“ 模式 ”与“ 值 ”之间以若干空格隔开：

```
/pattern/   value
```

模式表在 Postfix 系统上最主要的应用，是用来过滤掉烦人的垃圾邮件：将垃圾邮件的标题与正文的特征写入两个样式表，并将 `header_checks` 与 `body_checks` 这两个参数分别指向这两个模式表。关于垃圾邮件与模式表的问题，请参阅第十一章。

其他格式

Postfix 也可以将其他后台系统当成查询表使用，例如 MySQL、LDAP、NIS等。当你使用这类外界资源时，应该先从最简单的数据库格式入手，例如 `dbm` 或 `hash`，等确定该设定方法确实有效之后，再改用外界资源，并比较前后效果是否相同。

`postmap` 提供了一个重要的选项：“ `-q`”，让你测试任何种类的查询表。例如，当你使用 MySQL数据库时，下面两个命令应该返回相同的值：

```
$ postmap -q hash:/etc/postfix/transport

$ postmap -q mysql:/etc/postfix/transport.cf
```

关于 Postfix 如何搭配外界资源，请参阅第十五章。

别名文件

别名文件（alias file）是 Postfix 查询表中的特例，因为它使用一种兼容于 Sendmail 的格式。传统上，别名文件的文件名为 *aliases*，而且其实际存放位置随系统平台类型而定，但通常是在 */etc/* 或 */etc/mail/* 这两个目录中的一个。 Postfix默认使用系统原有的 aliases 文件，以便原本使用 Sendmail 的系统，可以继续沿用原有的别名文件。

别名文件的位置

基于沿革因素，邮件系统只使用一个别名文件数据库。但是 Postfix 没有这样的限制，你可以使用多个别名文件来组织你的配置信息。一般而言，当你要设定多组邮件列表（mailing list）时，你会想要将它们分别设定在各自的别名文件。 `alias_maps` 参数显现出你有哪些别名文件。

如果你的系统支持 NIS （一种专用来记录用户信息的网络数据库），Postfix 的默认行为会搜寻 NIS 所提供的别名信息。因此，默认的 `alias_maps` 参数如下：

```
alias_maps = hash:/etc/aliases, nis:mail.aliases
```

倘若你的系统提供 NIS服务，但是你不想使用它，你可以修改 `alias_maps`参数，让它只包含别名文件：

```
alias_maps = hash:/etc/aliases
```

有人可能会想要将别名文件与 Postfix 的其他配置文件放在一起，以便管理。对于这种情况，只要将 `alias_maps` 指向新位置即可：

```
alias_maps = hash:/etc/postfix/aliases
```

别名文件的位置既然都改了，别名数据库（`newaliases` 所产生的数据库文件）的位置也要修改：

```
alias_database = hash:/etc/postfix/aliases
```

重建别名数据库文件

由于别名文件的格式不同于其他 Postfix 查询表，所以不能使用 `postmap` 来重建别名数

据库文件。你应该使用 Postfix提供的另一个工具：postalias。此工具命令行的语法与postmap相同，你可用它将使用文字符号的别名文件转换成实际查询用的别名数据库：

```
# postalias /etc/aliases
```

另一个为了与 Sendmail兼容而设计的命令行工具是 newaliases，这个工具的作用其实与 postalias完全相同，但是其用法被刻意设计成兼容于 Sendmail 包里的同名工具。所以你可以直接下达 newaliases命令而不提供任何选项，也不必指定文本别名文件的位置，它就会自动产生别名数据库。

当你运行 newaliases 时，它是依据 alias_database 参数来决定源别名文件的位置。alias_maps 与alias_database 这两个参数的主要差异，在于前者可包含其他类型的别名信息来源（数据库文件、 NIS 等），而后者只能包含“ Unix数据库 ”这一种来源（必须使用 newaliases命令产生的文件)。注意，newaliases是依据default_database_type参数来决定 Unix 数据库的格式。

别名文件的格式

用于产生别名数据库的别名文件，其内容格式类似 Postfix 的查询表，唯一差别在于别名的定义。别名文件同样也可以有空白行与注释行，注释行同样是以 # 符号开头，也同样不能与别名定义摆在同一行。一个别名的定义可以跨越多行，以空格开头的行，被视为前一行的延续。

别名的定义格式，最左侧是别名本身，其后是一个冒号与若干空格，然后是一个或多个“目标”，目标之间以逗号隔开。别名与目标如果包含空格或任何特殊字符（# :、@ ），则必须以一对双引号界定其范围。简单来说，别名的定义格式如下：

```
alias: target1, target2, ...
```

冒号左侧的 alias一定是一个本地地址，所以不能包含网域部分。冒号右侧的 target通常是一个或多个邮件地址，有下列四种可能性：

邮件地址

任何符合 RFC 2822规范的地址都可以。这表示 target 可以是本地地址或是需要转递到另一台主机的外地地址。例如：

```
kyle.dent:        kdent, kdent@oreilly.com
```

文件名

本机文件的完整路径。新邮件会被添加到指定文件的末端，也就是将该文件当成本地邮箱使用。例如：

```
info:   /usr/local/mail/info_box
```

关于邮箱与邮箱格式的问题，请参阅第七章。

命令

一个竖线字符（|）与一个外部命令的完整路径。例如：

```
info:   "|/usr/local/bin/autoreply"
```

关于如何将邮件内容传递给外部命令，请参阅第十四章。

```
:include:file
```

引入一个含有额外目标的文件。原则上，该文件里的目标可以是前述的任何有效目标，但是在默认情况下，“文件名”与“命令”这两种目标是禁止的。例如：

```
info:   :include:/usr/local/mail/info_list
```

下一节会讨论哪些配置参数不在默认情况的限制范围内。

正常来说，当 Postfix投递本地邮件时，它会先使用收件人的身份，以获得将邮件写入收件人邮箱的权限；但如果收件地址是一个别名，则 Postfix 会使用别名文件拥有者的身份。不过，如果别名文件拥有者是特权用户（root），则 Postfix使用 `default_privs` 参数所指定的非特权身份。

别名目标限制

利用别名文件的“目标”可以做很多事。你可以用 `allow_mail_to_commands` 与 `allow_mail_to_files`来容许哪几种“目标”出现在别名文件里。有效的别名机制包括：alias（别名文件）、include（使用 include:file语法引入的外部文件）与 forward（第七章要讨论的 *.forward* 文件）。

基于安全考虑，这两个参数的默认值都是“alias , forward”，也就是说，只有别名文件与 *.forward*文件的目标可以设定为“文件”与“命令”，而引入文件的目标不可以是“文件”与“命令”。如果你想完全禁止邮件内容被递送到外部命令与文件，那就将这两个参数都设定为空白：

```
allow_mail_to_commands =
allow_mail_to_files =
```

相反，如果你希望所有别名机制都可以将邮件递送到外部文件与命令，则可以这样设定：

```
allow_mail_to_commands = alias, forward, include
allow_mail_to_files = alias, forward, include
```

这样的设定效果与 Sendmail 的默认行为相同。这样做可能使得邮件列表管理系统（mailing-list manager，MLM）成为凯觎目标，因为黑客提供的“文件名”或“命令”可能会被 MLM当成一般目标地址而被写入别名文件，进而使得系统安全出现隐患。如果你不需要额外的 include选项，那就接受 Postfix的默认值。

重要的别名

Postfix 包随附的别名文件模板，已经预先设定了一组系统别名。习惯上，这些系统别名最后都是指向 root账户。如果你想定期阅读root 的邮件，最好的办法不是直接以 root 身份登录系统，而是另外设置一个 root 别名，将其邮件转到另一个你平常用来收信的邮箱。

RFC 2142定义了一组所有网域都应该具备的别名，你可以按实际提供的 Internet的网络服务类型来决定要设置哪些别名。但无论如何，最起码要设置一个 *postmaster* 别名，好让其他网域的邮件管理员知道怎么联络你。请详细阅读 RFC 2142，决定你需要创建哪些别名。

重要的考虑事项

本章一开始示范了只要修改配置文件里的一两个参数，就可以顺利启动 Postfix。然而，在实际使用中，你需要多调整一些选项，使 Postfix 系统的运作方式符合你规划的用途。本节先讨论各种用途都必须考虑的问题，包括系统本身的标识以及非常重要的转发控制 (relay control)。

设定 MTA 的标识

不管你的 Postfix 要做何种用途，你都必须决定 MTA的标识，也就是其主机名称与网域名称。这方面的参数共计有四个，分别是 myhostname、mydomain、myorigin 以及 mydestination。

myhostname 与 mydomain

本章一开始就曾经讨论过 myhostname 参数的用途与重要性。如果你没指定 myhostname，Postfix 使用 gethostname() 函数来查询系统的主机名称。如果你的系统已经事先设定好正确的主机名称，则 gethostname() 能正确返回完整的主机名称，倘若该名称确实就是你想要的，则你不一定要设定 myhostname 参数的值。但如果 gethostname() 函数返回的名称不完整或者不是你想要的名称，则你应该明确赋予 myhostname 参数一个完整主机名称，或是将 mydomain参数设定成系统所属的网域的完整名称。当 mydomain参数的值为手工设定时， Postfix会依据 gethostname() 回复的本地主机名称以及管理员指定的 mydomain 参数值，自动推算出 myhostname 参数的值。

如果你将 myhostname 设定成系统的完整主机名称，但是省略了 mydomain，则 Postfix

扣掉 myhostname 的值的第一个点之前的部分，借此推算出 mydomain 的值。即若 myhostname被设定成 *mail.example.com*，则 Postfix会自动推算出 mydomain的值为 *example.com* —— 除非你自己刻意将 mydomain 设定成其他网域名称。类似地，如果主机名称是 *mail.ny.example.com*，则推算出来的网域名称为 *ny.example.com*。如果你的系统不能正确回复它自己的完整网域名称，而你也没设定 mydomain 和 myhostname参数值，则当你启动 Postfix 时，它会将问题记录在你的日志文件里。关于日志文件，请参阅本章的“日志记录”。

myorigin

当用户通过 Postfix 系统发出邮件，但是其信封与标头里的邮件地址都没注明网域名称时，Postfix会自动使用 myorigin参数定义的网域名称来补齐缺少的信息。 mydomain 的默认值等于 myhostname 的值。举例来说，若运行 Postfix 的主机的名称为 *mail.example.com*，当用户通过这台主机寄出一封“ From: kdent’邮件时，Postfix 在实际寄出邮件之前，会将它改写成“ From: *kdent@mail.example.com*”。由于大多数用户只希望出现网域名称（例如 *kdent@example.com*)而不要出现主机名称,所以 myorigin 参数通常设定成 $mydomain：

```
myorigin = $mydomain
```

mydestination

mydestination 参数列出了 Postfix应该将其视为“本地网域”的所有网域名称。在默认情况下，Postfix只会收下送给 $myhostname与 localhost.$mydomain 的邮件，也就是说，Postfix 的默认行为只接受送给系统主机的邮件。如果你的 Postfix要代整个网域上的所有主机收信，则应该将 $mydomain加入 mydestination的参数行：

```
mydestination = $myhostname, localhost.$mydomain, $mydomain
```

现在，你的邮件服务器可以成为整个网域的网关器，代该网域的所有主机收下邮件。

转发控制

除了为本地用户服务之外， Postfix也可以转发（ relay）其他系统的邮件。规定哪些对象可以使用你的系统的转发服务是非常重要的事。一般而言，局域网上的主机或用户需要你有能寄信到任何地方的能力，但是你不会愿意将同样的服务提供给局域网之外的网络。在邮件管理上，转发控制（relay control）是非常重要的功能，因为这是抑止垃圾邮件的首要控制手段。Spammer （滥发垃圾邮件的人，后文简称他们为“垃圾邮件发送者”）最常用的一种手段，就是找出有固定连接而且能为他们转发垃圾邮件的服务器（此类系

统称为“ open relay”），通过这些无辜的服务器来滥发垃圾邮件，借此隐藏自己真正的行踪。你应该禁止非授权对象使用你的邮件系统来转发邮件。如果你将系统设定成 open relay，这不仅只是助长垃圾邮件滥发问题而已，而是终有一天，你会发现自己的系统被列入黑名单，被其他邮件系统当成垃圾邮件的来源地，而成为他们的拒绝往来者，到最后，你甚至无法寄出自己的正常邮件。

关于如何阻挡垃圾邮件的课题，请参阅第十一章，本节先说明如何避免自己成为发送垃圾邮件的跳板。

限制转发访问

Postfix的默认配置不是 open relay。mynetworks_style与mynetworks 这两个参数决定客户端必须具备怎样的资格，才可以通过 Postfix 来寄出邮件。默认配置仅容许相同 IP 子网络上的其他主机使用转发服务。你可以使用 mynetworks_style参数来紧缩或放宽服务的地址范围：如果你只愿意让服务器自己能使用转发服务（用户必须登录服务器，使用服务器主机提供的 MUA软件来寄信），则可将 mynetworks_style设定成 host；你也可以将 mynetworks_style 设定成 class，让任何与服务器位于同一级 IP 网络（A、B、C 三级中的任一个）的主机都可以使用转发服务。不过，对于大部分网络而言，使用 class 配置等于将服务器开放给许多不相干客户端。如果你不熟悉 IP地址的分级方式，建议你使用默认的 subnet，或是更严格的 host。

或者，你也可以使用 mynetworks 参数明确指出哪些主机可以享受转发服务。mynetworks参数的效力高于 mynetworks_style ，如果你设定了前者，后者自动失效。你可以逐一列出个别的 IP 地址，或是使用 network/mask 表示法（例如 192.168.100.0/28）来指定一段子网络。mynetworks 的方便之处，在于你可以提供转发服务给一组特定的 IP地址群，而不管它们是否与服务器位于同一个子网络。例如，假设你是总公司的网管，你想提供转发服务给分公司办公室里的同事，而分公司位于另一个子网络，有固定的 IP地址。在这种情况下，你可将分公司网络的 IP 地址段纳入 mynetworks，让他们可使用你的 Postfix server来寄信。不过，要是你想服务的对象没有固定 IP 地址，则你必须采取某种 SMTP 身份验证机制。

SMTP 身份验证

所有能配合SMTP协议进行身份验证的技术都比较复杂。在你挑选任何一种验证技术之前，最好再想想看，还有没有比较简单的选择。比方说，有没有可能让远程用户得到固定的 IP地址？远程用户是否可改用另一台 SMTP server？或许远程用户的 ISP正好就提供 SMTP 代转服务。

有人可能会想到，为何不利用控制垃圾邮件的技术，设置一条“寄件人地址（信封地址）是否为本地网域”的检查规则，借此决定是否可代为转发邮件？答案是绝不要这样做！因为寄件人地址太容易假造了，而垃圾邮件发送者绝对懂得如何伪装寄件人地址——这简直是他们的专长。如果你真的使用了这种管制技术，你将会很快发现，你的邮件服务器已经成为 open relay，你不仅会收到“看起来像自家人寄出的垃圾邮件”，别人也会收到“看起来像是从你家寄出的垃圾邮件”。

因应动态 IP 地址的解决方案

第十二章会讨论如何使用 SASL 来验证 SMTP 通信双方的身份。SASL是规范 server 与 client 之间如何交换身份证书的一种通用协议。你的 SMTP server 必须链接额外的函数库才能使用 SASL。除了 SASL之外，还有三种类似的解决方案，分别是 *pop-before-smtp*、*DRAC*（Dynamic Relay Authorization Control）以及 *WHOSON*。这些方案都是针对没有固定 IP地址的客户端而设计的。这些技术要求用户必须先登录POP/IMAP server，借它提供客户端当前的 IP 地址给你的系统。SMTP server 得到客户端的 IP 地址之后，便批准该客户端可以使用转发服务一段时间（时间长短可由你事先调整）。这项技术的优点在于用户几乎感受不到有何不同，唯一差别是他们必须先检查新邮件（登录 POP/IMAP server），然后才能寄出新邮件。

DRAC 和 pop-before-smtp 两者与 Postfix 的合作方式，都是借由动态修改 Postfix 查询表：当用户成功登录 POP/IMAP server 之后，将客户端当时的地址填入查询表，在超过预定期限之后，删除先前填写的地址。Postfix不需要增加任何额外的函数库，唯一需要设定的是依据 POP/IMAP server 修改的那个查询表来决定是否提供转发服务，但 POP/IMAP server本身则需要修改、重新编译。DRAC 不同于 pop-before-smtp 之处，在于它可以通过网络来访问 SMTP server，而 pop-before-smtp 则要求 POP/IMAP server 必须与 SMTP server 安装在同一台系统上。

WHOSON 实际上是一种让 POP/IMAP与 SMTP两种服务器可以交互的接口协议。你可以在网络上架设一台 WHOSON server，然后取得一个修补文件（patch）让 Postfix 支持一种新型的查询表。在修补源程序并重新编译好 Postfix之后，它就可以与 WHOSON server 通信，借此判断特定客户端 IP 地址是否有权使用转发服务。

身份证书

另一种可以考虑的方案，是采用“客户端身份证书”（关于证书与 TLS，请参阅第十三章）。我们通常将证书当成保密通信的一种手段，其实证书也是最理想的身份验证法。然而，证书的审核与管理需要不少工夫，而且你的 Postfix 必须支持 TLS 协议才行。

这些额外的支持机制都不是理想方案，因为你可能要重新编译、链接现有的服务器程序，并开放某些系统文件的写入权限给这些服务器程序。此外，它们也加重了系统管理员的工作负担。如果你不能使用先前所说的非验证式技术，或是你的 SMTP server 必须满足“每一位’用户的转发要求 —— 不管他们到底是在Internet 上的哪个角落，那么，SASL 或许是最可靠、也最有灵活性的身份验证方案。

管理

维护邮件服务器是一件长期的工作。你不能只是启动 Postfix ,之后就不再管理。有一些例行性的管理工作必须靠你完成，而你也应该经常检查自己的系统是否有任何问题。本节讨论一般性的管理工作，并示范如何在 Postfix 完成这些工作。

Postfix 的 `postfix`工具提供了一个 check 命令，可帮助你检查当前的 Postfix配置是否有问题、文件与目录的拥有权是否正确，甚至帮你创建任何遗失的目录。运行方式如下：

```
# postfix check
```

这个检查程序秉持“没有消息就是好消息”的 Unix 优良传统。如果你的系统一切无误，它不会出现任何信息；否则，它会将查出来的问题显示在屏幕上，并同时记录在日志文件里。

日志记录

Postfix是一个长期运行的程序，你应该定期检查日志文件，看看有没有什么警告或异常的信息。有可能改变系统的事件，都可能会连带影响 Postfix ，因此，几乎所有 Postfix 的活动 —— 不管成功与否，都会被记录在日志文件。每次你启动或重新加载 Postfix时，最好检查一下日志文件，确定 Postfix 已经完成你要求的活动。

Postfix 的日志记录是通过系统的*syslog* daemon来代为完成。不管是哪一种版本的Unix，系统日志都是管理工作上非常重要的一环。如果你的服务器有专职的管理员，你应该向他（们）讨教关于 Postfix 日志记录的事宜。

一般而言，*syslog* daemon（`syslogd`）从各种系统进程中接收信息，并将信息写到它们的最终目的地（屏幕或日志文件）。syslogd对于日志信息的分类方式，主要是依据信息本身的重要性（严重程度）以及产生信息的应用程序或机制。 */etc/syslog.conf*配置文件决定 `syslogd`应该将各种类型的信息写到何处。 Postfix所使用的记录机制是“ mail”。如果你不知道Postfix产生的信息到底是记录在哪一个日志文件， */etc/syslog.conf*能提供正确的线索。某些操作系统是将所有信息都记录在同一个日志文件中，像 */var/log/*

syslog；而有些操作系统采取比较细腻的分类方式，它们可依据应用程序或服务类型，将信息分别记录在专属的日志文件中，对于这种系统， Postfix 日志信息可能会出现在*/var/log/maillog* 文件 —— 如果该系统的 */etc/syslog.conf* 是像下面这样设定的话：

```
mail.*          -/var/log/maillog
```

顺利找出邮件日志文件的位置之后，请务必定期检查它。你应该依据服务器的邮件量以及现行的日志轮替模式，决定至少每隔多久要检查一次。当然，决定权在你，多检查几次总是好事，重要的是要持之以恒，养成习惯。

要如何检查日志文件？这是一项需要练习的技巧。举例来说，如果想查出 Postfix 曾经发生哪些值得注意的事，你可以这样做：

```
$ egrep '(reject|warning|error|fatal|panic):' /var/log/maillog
```

这个命令假设日志文件位于*/var/log/maillog* ，如果不是，请换成实际使用的日志文件路径。

Postfix 的启动、关闭与重新加载

我们已经介绍过如何使用 postfix 命令来启动 Postfix：

```
# postfix start
```

在 Postfix 持续运行的过程中，如果你修改了 *main.cf* 或 *master.cf*，则 Postfix 必须重新加载 （reload）配置参数：

```
# postfix reload
```

此命令会等待运行中的进程完成工作后才结束它们，然后重新读取配置参数，继续运行。当你下达此命令时，如果 Postfix 正好在收信，发信方不会感受到任何异样或停顿。

在 start 或 reload 之后，最重要的事是检查日志文件，看看 Postfix 是否回复任何错误或警告信息。

如果要关闭 Postfix，请使用 postfix 的 stop 命令：

```
# postfix stop
```

同样，postfix 会等待进程完成未完的工作，然后才结束它们。对于 reload 就足够应付的事，就不应该先 stop 然后立刻 start。此外，尽量避免频繁的 start、stop、reload，因为这些动作都会严重影响服务器的运行效率。

开机时自动启动 Postfix

开机时会自动启动 Sendmail 的系统，在你安装了 Postfix 之后，下次开机时就会自动启动 Postfix 来取代 Sendmail。这是怎么办到的？答案在Postfix 提供的 sendmail兼容程序（与 Sendmail包的主程序同名）。通常，在开机期间，安装了 Sendmail 包的系统，其启动脚本（ startup script）是以类似下面的命令来启动 Sendmail：

```
sendmail -bd -q15m
```

所以，只要启动脚本调用的是 Postfix版的 `sendmail`,就可以顺利启动 Postfix。Postfix 的 sendmail 程序知晓绝大部分的 Sendmail 命令行选项，并运行 Postfix 的等效动作。当然，Postfix 不一定能了解所有 Sendmail 选项，“`-q`”选项就是其中一个例子。此选项决定 Sendmail应该每隔多久就要扫描一次队列，但是 Postfix 的等效机制是定义在 *main.cf* 配置文件的 `queue_run_delay`参数，其默认值为 1000 秒（约 16 分钟半）。

你的系统可能会提供一个配置选项，让你决定是否在开机期间自动启动Sendmail。安装了 Postfix 之后，打开这个配置选项可以让 Postfix在系统开机时自动启动。不同版本的 Unix 系统，决定开机启动程序的机制也不尽相同。如果你的系统的 Sendmail 启动脚本无效，或是你愿使用专为 Postfix 设计的启动脚本，那就自己写一个吧！

自助法

启动脚本的需求与惯例，随 Unix 系统的版本而异。所以，你应该先参考系统的说明文件，搞清楚启动脚本的存放位置，以及如何让系统开机程序运行你的脚本。在 System V 兼容的系统上，你可以参考例 4-1写一个自己的 Postfix 启动脚本：

例 4-1：一个 Sys V 风格的启动脚本

```
#!/sbin/sh
#
# 使用你自己的 logger 和 postfix 命令
# For Linux :
#  LOGGER="/sbin/syslogd"
#  POSTFIX="/usr/sbin/postfix"
#
LOGGER="/usr/bin/logger"
POSTFIX="/usr/sbin/postfix"
rc=0

if [ ! -f $POSTFIX ] ; then
    $LOGGER -t $0 -s -p mail.err "Unable to locate Postfix"
    exit(1)
fl
if [ ! -f /etc/postfix/main.cf ] ; then
    $LOGGER -t $0 -s -p mail.err "Unable to locate Postfix configuration
    exit(1)
fl
```

```
case "$1" in
    start)
        echo -n "Starting Postfix"
        $POSTFIX start
        rc=$?
        echo "."
        ;;

    stop)
        echo -n "Stopping Postfix"
        $POSTFIX stop
        rc=$?
        echo "."
        ;;

    restart)
        echo -n "Restarting Postfix"
        $POSTFIX reload
        rc=$?
        echo "."
        ;;

    *)
        echo "Usage: $0 {start|stop|restart}"
        rc=1

esac
exit $rc
```

例 4-1 只是一个范本，你应依据你的实际系统环境来修改它，甚至增加一些额外的事前或事后检查。完成修改之后，还必须将它安装在正确目录下才有效。通常， Unix 系统的启动脚本都是集中放在*/etc/init.d/* 目录下，但是偶尔也有例外，例如，HP-UX使用 */sbin/init.d/* 目录。除了要放在正确目录下，你还必须在适当的 run level 目录（可能是 */etc/rc2.d/*、*/etc/init.d/rc3.d/* ，随系统平台而定）建立一个符号链接（ symlink)。举例来说，如果你将上述脚本命名为 *postfix* ，则在 Linux 系统上，你应该将 postfix script 放在 */etc/init.d/* 目录下，然后以下列命令在 */etc/init.d/rc3.d/* 目录下制作一个符号链接：

```
# ln -s /etc/init.d/postfix /etc/init.d/rc3.d/S95postfix
```

由于各个 Unix 系统的初始启动机制都不太一样，实际细节请参阅你的系统说明书。

队列管理

Postfix 的队列，是邮件系统管理中非常重要的一环，所以我们将另辟专章讨论。关于 Postfix 的队列管理，请参阅第五章。

master.cf

Postfix 的所有服务器程序，都是由 master daemon 在必要时才予以启动的。这些服务器程序的运行参数，全部都定义在 *master.cf* 配置文件。

master.cf 的基本格式与 Postfix 的其他配置文件一样：# 符号代表注释，空白行与注释行没有作用，开头为空格的文字行被视为前一列的延续。

例 4-2 是一个 *master.cf* 样本。除了注释与空白之外的每一行，各描述一种服务的工作参数。参数行的每一栏，代表一个配置选项。“ - ”符号代表该栏为默认值。某些默认值是由 *main.cf* 配置文件里的参数决定的。

例 4-2：一个 master.cf 配置文件样本

```
#==========================================================================
# service   type  private unpriv  chroot  wakeup  maxproc cmd + args
#  name           (yes)   (yes)   (yes)   (never) (50)
#==========================================================================
smtp        inet  n       -       y       -       -       smtpd
smtp        unix  -       -       y       -       -       smtp
smtps       inet  n       -       n       -       -       smtpd
     -o smtpd_tls_wrappermode=yes -o smtpd_sasl_auth_enable=yes
submission  inet  n       -       n       -       -       smtpd
      -o smtpd_enforce_tls=yes -o smtpd_sasl_auth_enable=yes
pickup      fifo  n       -       y       60      1       pickup
cleanup     unix  n       -       y       -       0       cleanup
qmgr        fifo  n       -       n       300     1       qmgr
qmgr        fifo  n       -       y       300     1       nqmgr
#tlsmgr     fifo  -       -       n       300     1       tlsmgr
rewrite     unix  -       -       y       -       -       trivial-rewrite
bounce      unix  -       -       y       -       0       bounce
defer       unix  -       -       y       -       0       bounce
flush       unix  n       -       y       1000?   0       flush
showq       unix  n       -       y       -       -       showq
error       unix  -       -       y       -       -       error
local       unix  -       n       n       -       -       local
virtual     unix  -       n       y       -       -       virtual
lmtp        unix  -       -       y       -       -       lmtp
#
# 非 Postfix 软件的接口
#
cyrus       unix  -       n       n       -       -       pipe
  flags=R user=cyrus argv=/cyrus/bin/deliver -e -m ${extension} ${user}
relay       unix  -       -       n       -       -       smtp
proxymap    unix  -       -       n       -       -       proxymap
```

以下按顺序分别说明各栏的意义以及它们的默认值。

服务名称（service name）

服务器组件的名称。实际的命名规则，随该服务的传送类型（第二栏）而定。

传送方式（transport type）

传送服务所用的通信方法。有效的传送方式包括 inet、unix 与 fifo。

inet 方法表示服务可通过“网络套接字”（network socket）来访问，这类服务的对象可以是同系统上的其他进程，或是网络上其他主机的客户端进程。网络套接字服务的名称（第一栏），是用服务方的“IP 地址”（主机名称也可以）与“通信端口”（数值或 */etc/service* 定义的端口的符号名称）的组合来表示，例如：192.168.1.2:25、localhost:smtp 。如果服务方恰好位于本地主机上，则“IP地址”与冒号都可以省略。

unix代表“Unix domain socket”，而 fifo代表“命名管道”(named pipe)。两者都是同机器不同进程之间的通信机制，而且同样使用特殊文件为通信中介。unix与fifo服务的名称与Unix标准文件名的命名规则相同，但是不包含目录路径的部分。Postfix使用服务名称来创建通信中介用的特殊文件。 Unix domain socket 与命名管道两者都是 Unix的标准“进程间通信机制”(interprocess communications，通常简称为 IPC)。更详尽的信息，请参阅有关 Unix 程序设计的书籍。

表 4-1列举了各种服务类型的有效服务名称。

表 4-1 各种服务的可能名称范例

服务名称	传送方式	说明
smtp	inet	smtpd daemon 的服务名称。此为 */etc/service* 定义的SMTP通信端口代表的名称。
127.0.0.1:10025	inet	位于 loopback接口的 10025通信端口的服务器组件。
465	inet	位于本地主机（所有接口）的 465通信端口的服务器组件。
maildrop	unix	一个必须透过Postfix pipe daemon 才能访问的服务器组件。
pickup	fifo	一个必须透过 FIFO 机制才能访问的 Postfix 组件。

私有的(private)

某些服务组件仅供 Postfix 系统自己使用，不开放给 Postfix之外的其他软件使用。如果本栏标示为 y，表示私有访问（默认值）；n代表开放公共访问。inet类型的组件必须标示为 n，否则外界就无法访问该服务，毕竟网络套接字本身的用意，就是要开放给其他进程访问。

非特权的（unpriv）

是否使用非特权账户。默认值为 y，表示服务组件运行时，只需使用 mail_owner 参数指定的非特权账户（默认值为 postfix），即以完成任务所需的最低限度权限来提供服务。大部分 Postfix 组件都可以使用非特权账户。对于需要 root 特权的服务组件，此栏必须设定为 n。

改变根目录（chroot）

是否要改变组件的工作根目录，借此提升额外的安全性。工作根目录的位置由 *main.cf* 的 queue_directory 参数决定。此栏的默认值为 y（表示要改变工作根目录），大部分的 Postfix 组件也都可以在 chroot 环境下运作。不过，标准的安装方式是让所有组件都在正常环境下运行。将服务组件放在 chroot 环境下，添加了许多额外的复杂事情，你应该先通盘了解 chroot 所带来的保障，然后再决定这样的额外安全性是否值得你多费一番设定与维护的工夫。关于 chroot 的问题，请参阅本章关于“chroot”的讨论。

唤醒间隔（wakeup）

某些组件必须每隔一段时间被唤醒一次，定期执行它们的任务。pickup daemon 就是这样的一个例子。其默认休眠间隔是 60 秒，master daemon 每隔一分钟就唤醒 pickup 一次，要求它检查 maildrop 队列是否收到新邮件。qmgr 和 flush daemon 也是需要被定期唤醒的服务组件。在时间值之后尾随一个问号（?），表示只有在需要该组件时才予以唤醒；0 表示不必唤醒。此栏的默认值为 0，因为目前只有三个组件需要被定期唤醒。Postfix 包预先为这三个组件设定的唤醒间隔时间，应该足以应付大部分情况，其他服务组件都不需要 master 的定期唤醒。

进程数上限（maxproc）

可以同时运行的进程个数的上限。如果没指定，则以 *main.cf* 的 default_process_limit 参数为准（其默认值为 100）。如果设定为 0，表示没有任何限制。如果服务器系统的资源有限，或是想让系统在某方面的表现特别好，你可以调整 maxproc 的值。

命 令（command）

最后一栏是运行服务的实际命令。命令中的“程序文件名”部分不必包含路径信息，因为 master daemon 假设所有程序文件都放在 daemon_directory 参数所指定的目录下（默认目录为 */usr/libexec/postfix/*）。Postfix 的所有程序皆提供“-v”选项，可用来提高日志信息的详细程度，当我们需要解决问题时，经常利用这种方法来获得更多、更有用的调试信息。此外，你可以使用 -D 选项，让 Postfix 程序产生调试信息给调试程序。如果你需要知道更多关于调试的技术，请参考 Postfix 包随附的 *DEBUG_README* 文件。

每一个 Postfix daemon 都有自己的命令行选项。关于各个服务组件的选项，请参阅它们的在线说明书。请注意，只有 Postfix提供的服务器程序才可以放在命令栏，如果你想要运行自己的命令，请使用 Postfix 提供的 `pipe` daemon。关 于 `pipe`的用

时间单位

Postfix 有一些与时间 相关的参数，为了方便描述其值， Postfix提供了一组简写代号来表示时间 单位：s（秒）、 m（分）、h（时）、d（天）、w （周）。如果没明确注明时间单位，各时间参数以自己的默认时间单位来解读你给的值。虽然从在线说明书可查到所有时间参数的默认单位，但是谨慎的管理员不应该贸然留下模糊的解释空间，而应该明确标示给定时间值的单位。

某些服务器组件会参考 *main.cf*提供的参数值，但同时也提供了“ `-o` ”命令行选项，让你可以在 *master.cf* 中强制设定你要的参数值。举例来说，若要创建一个特殊的 `smtp`服务，你可将下面内容加入 *master.cf*配置文件：

```
smtp-quick  unix  -    -     n    -    -    smtp
   -o smtp_connect_timeout=5s
```

参数名称、等号与设定值可以紧接在一起，不必留空格。在加入本例这样的设定之后，你的系统就多了一个特殊的 `smtp-quick` 服务，当它寄信时，如果对方服务器五秒内没有响应，就会自动断线。但是，遵照 *main.cf* 设定值的那个 `smtp` 服务，则使用不同的 `smtp_connect_timeout`参数值。

收信限制

`smtpd` daemon 可以对外来邮件强加一些限制，包括邮件的大小、同一封邮件的收件人数、邮件内容每一行的长度上限，甚至是同一个客户端可以容许在发生多少次错误后予以断线。这些限制决定于 *main.cf*配置文件的几个参数。

`smtpd_recipient_limit` 参数决定一封邮件最多可以有几位收件人，默认值是 1 000。就一般的邮件系统而言，这是很宽松的限制。

`message_size_limit` 参数限制单封邮件的容量上限。默认值为 10 MB。如果想节约磁盘空间或内存，建议你降低此值。但如果你的用户经常夹带大的图像文件，则应该考虑适度扩大此值。

如果同一个客户端频繁出错，那通常是有问题或被攻击的征兆。 Postfix能累计客户端曾

经发生错误的次数，对于可疑的客户端， Postfix 会主动延迟响应的时间，而且错误次数越多，延迟时间就越长。这样的延迟效果可保护你的系统，以免被错乱的客户端搞垮（不管对方是无心或恶意）。初次的延迟时间由 smtpd_error_sleep_time 参数决定（默认值为 1秒钟），当客户端累积了 smtpd_soft_error_limit次错误之后，往后每发生一次错误，Postfix就多延迟 1秒钟，当错误次数超过 smtpd_hard_error_limit 时，Postfix就放弃该客户端，并主动断线。

举例来说，若网络远程有一个恶意程序连接到你的邮件服务器，试图发出垃圾命令来搞垮你的服务器。则这种垃圾命令会被Postfix视为一种错误，并会开始怀疑客户端是否心怀不轨。假设你设定了下列延迟参数：

```
smtpd_error_sleep_time = 1s
smtpd_soft_error_limit = 10
smtpd_hard_error_limit = 20
```

则一开始，每次客户端出错之后， Postfix都只会延迟 1 秒钟（ smtpd_error_sleep_time），在历经十次试探之后（ smtpd_soft_error_limit） Postfix开始延长每次延迟的时间。在第十一次错误时， Postfix 等待 11秒；第十二次则等待 12秒，依此类推。当总错误次数到达二十次（ smtpd_hard_error_limit），Postfix便有充分理由认定对方另有所图，所以会立刻切断连接。如果对方再次连接，而且试图制造同样麻烦，则他势必再经历一回漫长的等待。

改写地址格式

Postfix会尝试修正邮件标头里的邮件地址，使其符合 RFC 2822标准的格式。有两种显而易见的错误格式，Postfix 会自动予以修正。

第一种情况是只有人名而没有网域部分（例如： kdent），另一种情况是网域部分只有主机简名而没有域名（例如：*kdent@host*）。对于前者，Postfix 会将 myorigin 的值附加到人名之后；对于后者，则是将 mydomain 的值附加到主机简名之后（例如： *kdent@host.example.com*）。

规范地址

除了自动补齐不完整的邮件地址之外， Postfix还提供另一种改写地址的功能，让你可将毫无共通点的地址，替换成整齐划一的格式。 canonical_maps 参数（译注 1）指向一

译注 1： 虽然“canonical”有许多含义，但是在计算机专业术语中，它意味着“平常”、“标准”或“正常”。

关闭地址自动补齐

Postfix 自动补齐不完整邮件地址的行为，有时候会造成用户的混淆。假设你的 Postfix 系统代理 *example.com* 网域的电子邮件，有一天，系统从外界收到了一封邮件，但是其 From：字段所含的地址并不完整，如下：

```
From: Marketing
To: kdent@example.com
```

Postfix 运行它的例行修正工作，所以上述字段现在变成这样：

```
From: Marketing@example.com
To: kdent@example.com
```

类似本例这种不完整的地址，其实是发送垃圾邮件者惯用的伎俩之一。天真的用户看到调整后的地址，往往误以为垃圾邮件源自发送（无辜的）服务器。你当然可以设定 Postfix，要求它不要主动附加网域名称到不完整的地址，然而，除非你的服务器纯粹为“邮件网关”（mail gateway），没有任何邮件是从机器本身的 MUA 产生，才可以放心地关掉 Postfix 的自动补齐功能。有许多应用程序都要求邮件地址必须符合 RFC 2822 标准格式，如果邮件中所含的地址格式不合规定，可能会遇到不少麻烦。

要避免 Postfix 擅自将 myorigin 或 mydomain 附加到不完整的邮件地址，你可以修改 append_at_myorigin 和 append_dot_mydomain 这两个参数：

```
append_at_myorigin = no
append_dot_mydomain = no
```

在大部分情况下，你不会想要这样做。因为 Postfix 自己假设邮件地址总是以正确格式出现，而许多处理邮件信息的应用程序也是如此认为。比较妥当的解决办法，是拒绝接受邮件地址不完整的邮件。关于如何阻挡问题邮件，请参阅第十一章。

个定义地址对应关系的查询表，称为规范表（canonical map）。如果你的网络上有多个不同的邮件系统，分别产生了不同样式的地址，你可以将所有邮件都导到一个 Postfix 网关，由它将各式各样的地址转换成标准格式。规范表通常用于将邮件地址从（凌乱的）内部格式转换成（统一的）公共格式，像这样：

```
#
# /etc/postfix/canonical
#
pabelard@example.com    peter.abelard@example.com
hfulbert@example.com    heloise.fulbert@example.com
```

甚至也可以连同网域部分一起转换：

```
#
# /etc/postfix/canonical
#
pabelard@example.com    abelard@oreilly.com
hfulbert@example.com    heloise@oreilly.com
```

在 *main.cf*，将 canonical_maps 参数指向规范表：

```
canonical_maps = hash:/etc/postfix/canonical
```

让你的规范表运行一次 postmap，并在修改 *main.cf*之后重新加载 Postfix：

```
# postmap /etc/postfix/canonical
# postfix reload
```

canonical_maps 参数会影响所有地址，包括信封与标头里的地址。每当 Postfix 发现邮件地址符合规范表的某索引键，便会将它改写成对应值。如果你只想修改收件人或发信者的地址，Postfix 提供了额外的参数：sender_canonical_maps 与 recipient_canonical_maps 。这两个参数的工作方式与 canonical_maps相同，但是只会影响相关的地址。如果你同时设定了这三个参数， Postfix 会依照下列顺序来改写地址：先是 sender_canonical_maps ，然后是 recipient_canonical_maps，最后才是 canonical_maps。

伪装主机名称

地址伪装（address masquerading）的用意，在于隐藏内部主机的名称，让邮件看起来就像是从网关系统发出去的一样。假设你的 Postfix server 需要转发内部网络主机寄出的邮件，当内部主机送出邮件时，发信者地址含有完整的主机名称（例如 *me@host.domain.com*），但是你希望只出现网域名称（例如 *me@domain.com*）。masquerade_domains参数用来将主机名称转换成较简单的网域名称，你只要将一个网域名称赋值给它：

```
masquerade_domains = example.com
```

如此一来，类似 *heloise@server1.example.com*和 *frank@server2.example.com* 这样的地址，会被分别转换成 *heloise@example.com* 和 *frank@example.com*。

masquerade_domains 参数可以接受多个网域名称，甚至是子网域名称。 Postfix 会按照你给定的顺序来比对邮件中的地址。想象一个含有两个子网域的企业网络：*acct.example.com*和 *hr.example.com* ，你希望来自这两个子网域的邮件只出现它们各自的子网域名称，而来自任何其他网域或网络主机的邮件，都只出现上层网域名称（此例中的 *example.com*），则你可以这样设定 *masquerade_domains*：

```
masquerade_domains = acct.example.com hr.example.com example.com
```

在这种情况下，由于 *heloise@sys3.acct.example.com*这个地址属于*acct.example.com*子网域，所以会被伪装成*heloise@acct.example.com*；而另一个地址*frank@db.hr.example.com* 属于 *hr.example.com* 子网域，所以会被改成 *frank@hr.example.com*；最后，*helene@server1.example.com*地址属于 *example.com*网域，所以会变成*helene@example.com*。

如果你想保留某个原本会被换掉的网域名称，你可以在该网域名称之前注明一个感叹号（!）：

```
masquerade_domains = !it.example.com, example.com
```

这表示来自 *it.example.com* 网域的邮件地址不会被改写，换言之， *kdent@it.example.com* 会被保留原样。

Postfix 甚至容许我们排除特定的账户。举例来说，假设我们希望 *root@db.example.com* 与 *admin@labs.example.com* 这样的地址保留原样（好让我们得以追查来源），则我们可将这些账户名称列入 masquerade_exceptions参数：

```
masquerade_exceptions = admin, root
```

使用地址伪装功能时，标头与信封上的所有地址都会受到影响，唯一例外是信封上的收件人地址。这使得写给特定主机的邮件，仍可通过邮件网关器送到该主机，而同时又能伪装成由该主机寄出的邮件。如果你希望所有类型的地址都被伪装，那就将 Postfix 能认出的所有地址类型全部都列在 masquerade_classes参数中：

```
masquerade_classes = envelope_recipient, envelope_sender,
                     header_sender, header_recipient
```

请注意，如果你像上面这样设定 masquerade_classes，则邮件网关系统将不知道如何投递原本应该送到 *kdent@server1.example.com* 的邮件，因为该地址已经被改成 *kdent@example.com*。

改变投递地址

当用户更改邮件地址（邮箱），而你不愿意服务器继续帮他们代收邮件时，你可以用一个查询表定义这些用户的新旧地址对应关系，并让 relocated_maps 参数指向此查询表：

```
relocated_maps = hash:/etc/postfix/relocated
```

查询表的每一行各记录一个“旧地址一新地址”的对应关系。每当收到要送给旧地址的

邮件或是旧网域的任何人发的邮件， Postfix会拒收，并将对应的新地址发给寄信方。举例来说，若 */etc/postfix/relocated* 查询表的内容是这样：

```
kdent@ora.com       kdent@oreilly.com
heloise@ora.com     hfulbert@oreilly.com
```

那么，每当有 SMTP client 要求 Postfix 将邮件送到 *kdent@ora.com* 或 *heloise@ora.com* 时，Postfix 皆会予以拒收，而 SMTP client 可从响应的错误信息中知道新地址。

除了个人地址变迁之外，Postfix 也能应付整个网域搬迁的情况。如果将下列对应关系加入 */etc/postfix/relocated* 查询表：

```
@example.com        oreilly.com
```

则每当 Postfix 遇到要寄给 *example.com* 的邮件，不管收件人是谁，都一律予以拒收，并告知对方，邮件应该送到 *oreilly.com* 才对。

不明用户

对十应该收下的本地邮件，如果收件地址的人名部分在任何查询表、别名表、系统账户都查不出来，这个人即被视为“不明用户”（unknown user）。正常情况下，Postfix 会拒收写给不明用户的信；如果你想收这类邮件，你可以将 `local_recipient_maps` 参数设成空，避免 Postfix 拒收不明用户的邮件，然后使用 `luser_relay` 参数将这类邮件集中到特定邮箱：

```
local_recipient_maps =
luser_relay = catchall
```

这里假设 `catchall` 是系统上的一个有效地址（别名或实际账户），可用于接收寄给不明用户的邮件。请注意，使用 `luser_relay` 有潜在的风险。首先，你会因此而收到大量垃圾邮件。因为发送垃圾邮件者经常采用“字典攻击”（dictionary attack），也就是将常见的人名与你的网域名称结合在一起，企图蒙混过关。当你设定了 `luser_relay` 之后，原本会被 Postfix 挡掉的邮件，会全部都集中在你指定的邮箱中。另一个问题，如果不明邮件是因为寄件人打错字所造成的，对方会误以为邮件已经顺利寄出，而没有发现错误的机会。这样你必须天天检查这些被拦截下来的邮件（包括大量烦人的垃圾邮件），并承担潜在的道德风险（私看别人的邮件不是合法行为，除非得到授权）。

改变根目录（chroot）

Postfix 提供多层次的安全防护措施，其中一个层次是让你可将大部分的 Postfix 服务局限在 *chroot* 环境下运作。chroot 是 Unix 操作系统特有的一种安全措施，其作用是将进

程环境（process context） 的根目录，从平常的“／”改成文件系统上的另一个子目录，将进程的文件系统限制在该子目录之下；也就是说，被chroot的进程，无法访问该子目录之外的文件系统。

对于必须与外界通信的服务器进程而言， chroot 特别有用，因为今日的Internet到处充斥着心怀不轨的人。这样，即使万一有人侵入了 `smtpd` daemon，他顶多只能访问一小部分的文件系统，能够造成的损害非常有限，甚至无法从中获得任何好处。

不过，为Postfix规划一个chroot环境是一件颇为复杂的工程，复杂到系统管理员通常不愿意面对。一般而言，chroot不是必要的，除非使用Postfix的主机位于需要高度安全的环境中，或是位于必须暴露在外的服务器上，例如专用的防火墙与堡垒主机。

所有支持chroot的Postfix进程，都可将它们的根目录改成 `queue_directory` 参数指定的子目录，这通常是 */var/spool/postfix/*。对于处于chroot环境下的进程，在该进程环境下必须以 */pid/* 路径来访问文件系统上的 */var/spool/postfix/pid/* 目录，而且不能访问新根目录之外的文件系统。

除了 `pipe`、`virtual`、`local`和 `proxymap` 之外的其他Postfix服务器组件基本上都可以被chroot，只要将该组件在 *master.cf* 配置文件里的第五栏改成 `y` 即可，具体方法请参阅例4-2。

既然chroot改变了服务器进程的文件系统的可视范围，我们就必须将进程所需的全部资源都放在新的根目录之下，这样被chroot的服务器进程才能顺利运行。但是Postfix daemons所需的资源因平台而异，这里很难逐一列出来说明。大体上， Postfix需要能够访问账号信息（*/etc/passwd*）、DNS解析配置（*nsswitch.conf* 或 *resolv.conf*）、时区信息或提供时间信息的函数库。某些平台还需要特定的设备文件。 Postfix包里随附了专为不同平台而设计的脚本，在解压包的 *examples/chroot-setup/* 子目录下可以找到它们。

运行正确的脚本，应该足以为你的系统设置好chroot环境。如果没有适合你的平台的脚本，你可能要做一些小实验，才能备齐所有必要的资源。考虑先前提到的那些资源，或是参考其他平台的脚本，应该对你有所帮助。实验时请留心观察日志文件，看看被你禁锢的进程发出了怎样的错误信息。比方说，如果你看到类似这样的记录：

```
postfix/smtp[1575]: fatal: unknown service: smtp/tcp
```

这表示 Postfix无法判断 `smtp` 服务应该使用哪一个通信端口，而问题可能是其无法在chroot环境下访问 */etc/services* 文件所致，所以，只要将该文件复制一份放在 */var/spool/postfix/etc/services* ，问题或许就解决了。从错误尝试中学习看出症状背后所隐藏的病因，也有助于累积实际管理经验。

如果平常的 Postfix 日志提供的线索还不足以找出毛病，不妨提高服务器进程的“详细程度”（将 `-v` 加到服务器进程在 *master.cf* 里的命令行选项），或者利用程序员常用的各种调试工具（例如 `truss`、`strace` 或 `tusc` 等）检查受测程序到底在哪里停了下来。如果你发现失败原因是因为漏掉了某个组件，那就将该组件复制到 chroot 环境下。关于如何利用调试工具来追踪 Postfix 的运行情况，请参阅 *DEBUG_README* 文件。

在 chroot 环境下运行的 Postfix 系统，需要你多花一点心思维护，以保持平常环境的系统文件与 chroot 环境下的版本一致。举例来说，如果你的 chroot 服务器进程需要 */etc/passwd* 文件，那么，每当系统的 */etc/passwd* 被改变，你就必须同步更新 */var/spool/postfix/etc/* 目录下的副本。创建文件链接的办法是行不通的，因为符号链接（symlink）无法进入 chroot 系统（如果可以，那也不必 chroot 了），而硬链接（hard link）无法进入文件系统。

在线说明书

原版 Postfix 包随附了许多说明文件。如果你使用别人预先编译好的安装包（*.rpm*、*.deb*、*.pkg* 等），这些文件不一定齐全，但是最基本的 manpage 和配置文件样本应该都有。样本文件放在 `sample_directory` 参数指定的目录下（通常与 *main.cf* 位于同一个目录），Postfix 的所有参数都可以在这些样本文件里找到详细说明。

安装好 Postfix 之后，在线说明书（manpages）应该会安装在 man 能够自动找到的地方。要检查说明书的安装位置是否正确，只要直接下达 man 命令看看是否有效就可以了，例如：

```
$ man postfix
```

如果你的系统没有显示说明书，而是出现类似这样的错误信息：

```
$ man postfix
No manual entry found for postfix.
```

那表示你的说明书不是放错目录，就是当初根本没有安装它们。各个 Unix 系统放置在线说明书的位置都不太一样，但是 `MANPATH` 环境变量应该可以给你一些线索。

Postfix 的各种命令行工具、查询表、daemon 都有对应的在线说明，而且所有文件都有 HTML 版本。如果你的系统没安装 HTML 文件，在 Postfix 的网站（*http://www.postfix.org/*）可以找到它们。在线说明文件永远能反映当前 Postfix 版本的实况，如果你发现本书提供的信息不符合实际现况，记住， man 就在你身边。

第五章

队列管理

队列管理单元的服务器程序 —— qmgr，是整个 Postfix 系统的中心枢纽（注 1）。所有邮件，包括等待送出与从外界收进来的，都必须通过队列。了解队列的运行原理以及 Postfix 如何处理队列，有助于你解决问题。

队列管理器总共设置了五个做不同用途的队列，包括：输入（incoming）、活动（active）、等待（deferred）、故障（corrupt）、保留（hold）。每一个队列在 queue_directory 参数指定的路径下各有一个专属的子目录。默认的队列目录是 */var/spool/postfix/*，所以，你的 Postfix系统上可能有一个类似下面的目录结构：

```
/var/spool/postfix/active/
/var/spool/postfix/bounce/
/var/spool/postfix/corrupt/
/var/spool/postfix/defer/
/var/spool/postfix/deferred/
/var/spool/postfix/hold/
```

于后台运作的 qmgr daemon 能自动处理大部分的队列管理工作，必要时，管理员可使用 postsuper 和 postqueue 自己动手管理。本章介绍 qmgr 与相关命令行工具的运行原理，以及能影响队列的 Postfix 参数。

注 1：到 2.0.18 版为止，队列管理服务器程序的真实名称都是 nqmgr（这个 n 是 new的意思）。qmgr 是 Postfix 初版的队列管理服务器程序，后来为了使用比较好的调度算法，又为了与 qmgr 共存，所以才暂时使用 nqmgr这个名称。在将来，一旦证实 nqmgr 够稳定之后，它会被命名为 qmgr，正式取代旧版本的地位。既然 nqmgr注定只是个过渡期的名称，本书于是提早帮它正名。

qmgr 的运行原理

图 5-1 描绘了邮件如何通过各个队列。邮件进入 Postfix 系统的第一站是“输入队列”。Postfix以 queue_minfree参数来保护队列文件系统，此参数的默认值为 0，表示 qmgr 可以无限制地使用队列磁盘空间。如果你不想队列耗尽服务器的磁盘空间，建议你设定一个合理的上限值。

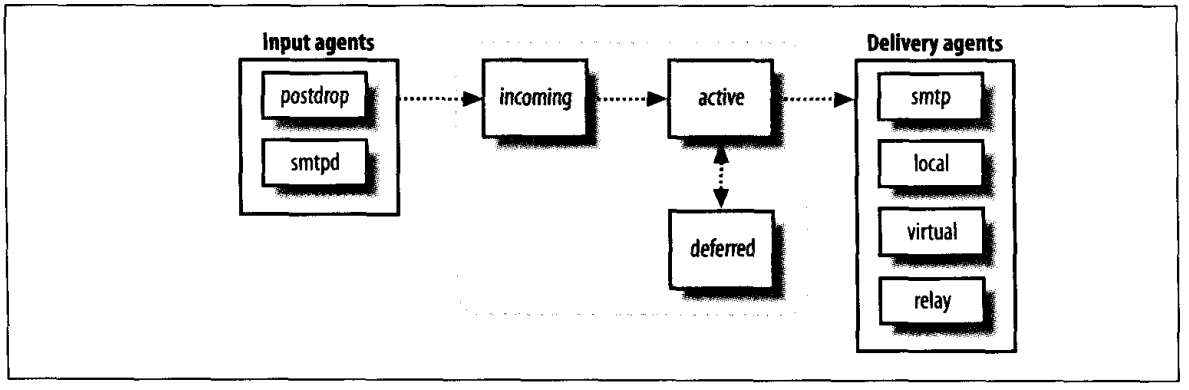

图 5-1：邮件经过各队列的流程

每当有新信进入收件队列， qmgr 便会将邮件移到活动队列，并调用适当的 MDA 来处理。只要投递过程没遇到意外，这段流程通常是相当迅速的，快到你没机会见到邮件停留在活动队列里，除非 Postfix正在将邮件送到一台非常慢的远程 SMTP server。如果无法在 30 秒内连接到远程系统，Postfix 便会认为该系统目前无法到达。

暂时无法送出的邮件会被放在等待队列（deferred queue）。可能发生的临时性意外包括：DNS 系统暂时离线、远程服务器主机临时故障、网络不稳定、收件人的邮箱爆满等。如果遇到永久性的问题或是被远程拒收，则邮件会连同错误报告（或退信通知）立刻被退回给原寄件人，不会留在队列里。

等待邮件

留在等待队列里的邮件，会待到它们被成功投递出去，或因为超过时限而被退回给原寄件人为止。退信通知函与遭退邮件的总和大小不得超过 bounce_size_limit（其默认值为 50 000bytes），如果超过此限制，则原寄件人只会收到退信通知函，而不会收到遭退邮件本身。

如果邮件一时无法投递， Postfix会给邮件标上一个时间，这个时间代表下次尝试递送的时刻。对于一时无法联系的远程主机，Postfix 会将它们另外记录在自己内部的一份有时效性的名单上，借此避免不必要的递送尝试。如果有任何等待邮件要被安排重新投递，

而且活动队列里有足够空间，则 qmgr 会交替在等待队列与收件队列之间取信。如此一来，新进邮件就不至于因为 Postfix正在忙着处理大量等待邮件而等待太久。

队列调度

Postfix 会定期扫描队列，检查每一封等待邮件的时间，看看是否有任何等待邮件已经到了下一回合应该尝试投递的时间。每一次投递失败，都会导致延迟间隔时间加长，所以，失败次数越多，就必须等待更长的时间才有再次被投递的机会。延迟时间上限由 maximal_queue_lifetime 参数决定。如果邮件在等待队列里的时间超过此上限，Postfix便会放弃投递，并退信给原寄件人。 maximal_queue_lifetime 的默认值为 5 天，你可以将它改成任何时间长度；如果你希望 Postfix 在第一次递送失败时就立刻退信，你可将此参数的值改为 0。

Postfix 内部的队列扫描间隔决定于 queue_run_delay参数，其默认值为 1000秒。也就是说，大约每隔 1000秒，Postfix就会检查一次等待队列，看看是否有任何应该再次尝试投递的邮件。

Postfix 重新投递等待邮件的频繁程度的上下限，分别由 minimal_backoff_time与 maximal_backoff_time这两个参数决定。每次邮件被延迟， qmgr便会加长下次投递机会的时间间隔，但是累增后的间隔时间不得超过maximal_backoff_time(默认值为 4000秒),也不得低于 minimal_backoff_time （默认值为 1000秒)。如果你发现经常有寄不出去的等待邮件，或许你应该考虑适度加大maximal_backoff_time ,让Postfix 不要浪费太多资源于等待邮件。

投递操作

每当需要投递邮件时， qmgr便会调用适当的 MDA 来运行投递程序。Postfix会很谨慎地避免干扰收信系统，并提供几个参数来控制外寄邮件的可用资源。这些参数的默认值应该足以应付大部分的情况，但是如果你的操作环境特殊，需要调整资源分配或是想要改善投递操作，你应要实际地测试 qmgr或 MDA的操作参数，并依据实测效果来加以调整。

列在 *master.cf*配置文件里的各个 MDA，都有可能被用来投递外寄邮件。*master.cf*配置文件的 maxproc字段（第七栏）限定了 MDA的进程数上限。如果该字段没有明确的设定值，则 Postfix 使用 default_process_limit 参数的值为默认值。

大部分的 MDA 都有能力同时投递多封邮件到同一个目的地，但是并非所有收信系统都时时刻刻有能力同时收下许多邮件。为了兼顾投递效率与接收系统的工作负荷， Postfix

使用一种特殊的投递调度算法：如果有多封邮件要送到同一个目的地， MDA 第一次最多只能同时送出 initial_destination_concurrency封邮件（默认值为 5），如果能成功完成初次投递，则 Postfix 会试着同时送出更多邮件（如果队列里还有要寄到同目的地的邮件），直到发现目的地系统不能承担当前的工作负荷，或是同时投递的邮件数即将超过 default_destination_concurrency_limit（其默认值为 20）。

你可以适度提升 initial_destination_concurrency 参数的值，但是不能超过该 MDA 在 *master.cf* 配置文件的 maxproc 字段值（或 *main.cf* 配置文件的 default_process_limit 参数值）。一般而言，你不应该提升 default_destination_concurrency_limit，因为这有使收信系统瘫痪的风险。

Postfix随附的每一个MDA ,都有一个对应的mda_destination_concurrency_limit 参数，它们的效力高于 default_destination_concurrency_limit 参数。因此，如果想提升本地邮件的同时投递量，而不想影响其他 MDA，你可以修改 local_destination_concurrency_limit 参数的值。同理，如果只想降低远程邮件的同时投递量，而不影响其他 MDA，也可借由修改 smtp_destination_concurrency_limit 参数来达成目的。

同理，Postfix 内置的所有 MDA 都有自己的 mda_destination_recipient_limit 参数（它们的默认值全部都是 default_destination_recipient_limit） ，它们限制 Postfix 将邮件交给 MDA时，每次最多可以指定多少位收件人（译注 1）。如果同一封邮件的收件人数超过 MDA 每次能够承受的人数， Postfix 会将收件人分成多个小组，分批投递。

损毁邮件

故障队列（ corrupt queue ）纯粹用于存放受损或无法解读的邮件。如果邮件已经损坏到无法进行任何处理的程度，Postfix 便将它们放在此处，供你调查故障原因。管理员可使用“ 队列管理工具 ”提到的工具来查看故障邮件。一般而言，故障邮件非常少见，如果不幸遇到，通常是操作系统或硬件故障的征兆。

错误通知函

Postfix 可发出“ 错误通知函 ”给管理员，让管理员知道发生了什么类型的错误。 Postfix 将错误通知分成七大类，如表 5-1所示。管理员可设定 *main.cf* 里的 notify_classes 参

译注 1： 注意， mda_destination_recipient_limit 与邮件本身最多可以寄给多少人无关。

数来决定要收哪些类型的错误通知。在默认情况下，管理员只会收到 resource（资源）与 software （软件）两种错误类型的通知函。

对于每一种类型的错误通知函，都有一个对应的 *class*_notice_recipient 参数，这些参数代表该类通知函的收件人地址，它们的默认值都是 *postmaster* 。表 5-1列出 Postfix 定义的七种错误类型，以及对应的通知函收件人参数。

表5-1 :错误通知

分类	通知函内容	通知函收件人
bounce	遭退邮件本身的标头	bounce_notice_recipient
2bounce	无法投递的退信通知函	2bounce_notice_recipient
delay	被延迟邮件的标头	delay_notice_recipient
policy	SMTP 对话过程（因违反垃圾邮件过滤条件而予以拒收的邮件）。	error_notice_recipient
protocol	SMTP 对话过程（曾经在 SMTP对话过程出现错误者）	error_notice_recipient
resource	因为系统资源问题而投递失败的通知	error_notice_recipient
software	因为软件问题而投递失败的通知	error_notice_recipient

如果想收到所有问题的通知函，请照下面这样设定 notify_classes 参数：

```
notify_classes = bounce, 2bounce, delay, policy,
   protocol,resource, software
```

如果你的站点有专人研究垃圾邮件，你可以这样设定：

```
notify_classes = policy, protocol, resource, software
policy_notice_recipient = antispamexpert@example.com
```

队列管理工具

Postfix 提供了一组命令行工具来检查、控制、管理队列里的邮件，其中以 postsuper和 postqueue这两个工具最为重要。你可以对队列中的邮件进行下列操作：

显示邮件列表

删除邮件

保留邮件

重新排队

- 显示邮件内容
- 清空邮件

每一种操作都有对应的命令，以下几个小节分别解释如何完成各种操作。

显示邮件列表

邮件列表的显示项目，包括邮件在队列里的标识符（Queue ID）、大小（Size）、到达时间（Arrival Time）、寄件人地址（Sender）、收件人地址（Recipient）。对于等待邮件，还会另外显示等待原因。如果邮件是在活动队列里，其 Queue ID 栏会加注一个星号（除非你的服务器很慢或是很负荷沉重，否则应该没有机会见到星号）。在保留队列里的邮件，其 Queue ID 栏会加注一个感叹号。等待的邮件不加注任何符号。

使用 `postqueue -p` 命令可列出队列里的所有邮件，其效果如同 Sendmail 包的 `mailq`。当安装 Postfix 时，安装脚本会以 Postfix 版的 `mailq` 程序（它其实只是 `postqueue -p` 的符号链接而已）取代 Sendmail 包原有的同名程序，借此维持与 Sendmail 包的兼容性。

典型的邮件列表如下：

```
$ postqueue -p
-Queue ID- --Size-- ----Arrival Time---- -Sender/Recipient-------
DBA3F1A9      553 Mon May 5 14:42:15  kdent@example.com
       (connect to mail.ora.com[192.168.155.63]: Connection refused)
                                       kdent@ora.com
-- 1 Kbytes in 1 Request.
```

如上所示，这封邮件的 Queue ID 没有星号也没有感叹号，所以可确定它被放在等待队列。

删除邮件

使用 `postsuper` 命令的 `-d` 选项（代表 `delete`），可移除队列里的邮件。邮件是以它们在队列里的标识符表示（`postqueue -p` 显示的 Queue ID 栏），例如：

```
# postsuper -d DBA3F1A9
postsuper: DBA3F1A9: removed
postsuper: Deleted: 1 message
```

如果要删除所有邮件，请把 Queue ID 换成 `ALL` 这个关键字：

```
# postsuper -d ALL
postsuper: Deleted: 23 messages
```

请注意，由于删除所有邮件是相当危险的操作（有可能误删不该删的信），所以 `ALL` 关键字必须全以大写字母表示才有效，其目的是希望你三思而后行。

不管是一次全删或一次只删一封邮件，都是过尤不及的做法。实际上，我们时常需要只删掉寄给某人（或来自某人）的邮件。例 5-1是一个方便好用的 Perl程序，让你通过邮件地址来指定要被删除的邮件。

例 5-1：让你能通过邮件地址来删除邮件的 Perl 程序

```
#!/usr/bin/perl -w
#
# pfdel —— 从 Postfix 队列删除含有特定邮件地址的邮件
#   （不管是收件人还是寄件人的邮件地址）
#
# 用法：  pfdel <E-mail_address>
#

use strict;

# 如果必要的话，请依据你的系统现况修改下列路径：
my $LISTQ = "/usr/sbin/postqueue -p";
my $POSTSUPER = "/usr/sbin/postsuper";

my $E-mail_addr = "";
my $qid = "";
my $euid = $>;

if ( @ARGV !=  1 ) {
        die "Usage: pfdel <E-mail_address>\n";
} else {
        $E-mail_addr = $ARGV[0];
}
if ( $euid != 0 ) {
        die "You must be root to delete queue files.\n";
}

open(QUEUE, "$LISTQ |") ||
  die "Can't get pipe to $LISTQ: $!\n";

my $entry = <QUEUE>;    # 跳过第一行（译注 2）
$/ = "";                # 其余数据分散在多行
while ( $entry = <QUEUE> ) {
        if ( $entry =~ / $E-mail_addr$/m ) {
                ($qid) = split(/\s+/, $entry, 2);
                $qid =~ s/[\*\!]//;
                next unless ($qid);

                #
                # 运行 postsuper -d $qid
                # 直接将 postsuper 的输出信息传送给用户
                #
                if ( system($POSTSUPER, "-d", $qid) != 0 ) {
                        # 如果 postsuper 遇到问题，则直接放弃
                        die "Error executing $POSTSUPER: error " .
```

译注 2：　因为第一行若不是“ -Queue ID- --Size-- ...”，就肯定是“ Mail queue is empty”。

```
                        "code " . ($?/256) . "\n";
                }
        }
}
close(QUEUE);

if (! $qid ) {
        die "No message with the address <$E-mail_addr> " .
            "found in queue.\n";
}

exit 0;
```

保留邮件

当你想将邮件无限期留在队列系统里，保留队列（hold queue）就成了容纳这些邮件的场所。图 5-2描绘了保留队列与其他队列的交互关系。从该图可看出，不管邮件目前已经传到哪一个队列，你都可以将它们移出原来的队列，转移到保留队列。

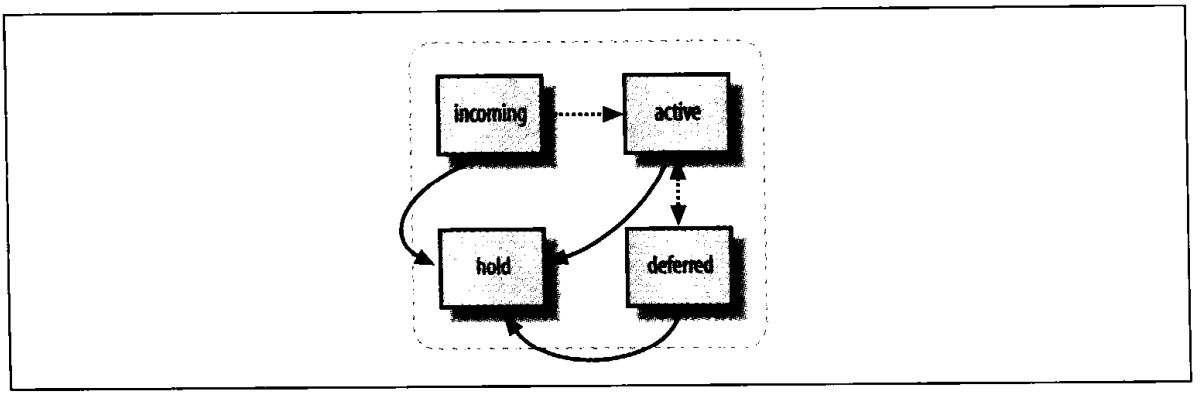

图 5-2 保留邮件

假设你已知道要保留的邮件的标识符，使用 postsuper工具的 -h选项，就可将指定的邮件搬到保留队列：

```
# postsuper -h DBA3F1A9
```

在这之后，如果观察邮件列表，会发现该邮件的 Queue ID 多了一个感叹号：

```
$ postqueue -p
-Queue ID- --Size-- ----Arrival Time---- -Sender/Recipient-------
DBA3F1A9     553 Mon May 5 14:42:15  kdent@example.com
       (connect to mail.ora.com[192.168.155.63]: Connection refused)
                                       kdent@ora.com
-- 1 Kbytes in 1 Request.
```

要将邮件移回原来的队列，继续其未竟之进程，可使用同一个 postsuper工具，只要将将原来的 -h选项改成大写的 -H选项：

```
# postsuper -H DBA3F1A9
```

在邮件被移回原来的队列之后， qmgr依照平常的调度原则来决定其下次投递时间。或者，你也可以执行清空（flush）命令，立刻将邮件递送出去（参阅“清空邮件”）。

重新排队

如果因为配置问题而耽搁了任何邮件，在问题解决之后，你可能希望被耽搁的邮件重新走一遍队列处理流程，以便能够成功完成递送。因为配置问题可能使得Postfix在邮件里存储了错误的投递处理信息，或是使用了错误的地址改写法则。重新排队（requeue）会使得 Postfix依据你的新配置来修正错误信息。使用 postsuper工具的 -r选项，可以重新排队某一特定邮件（如果你只提供一个 Queue ID的话），或是要求所有邮件(ALL)全部都重新排队：

```
# postsuper -r ALL
```

被重新排队的邮件，会得到新的 Queue ID，而且其标题上会多一个额外的 Received: 字段。

显示邮件内容

使用 postcat 工具的 -q 选项，可以查看一个队列文件的内容：

```
# postcat -q DBA3F1A9
```

早期版本的 postcat 并未提供 -q 选项，而是要求你提供队列文件的完整路径。然而，由于邮件可能在各个队列目录（*maildrop/*、*incoming/*、*active/*、*deferred/*、*hold/* ）之间游移不定，而且这些队列目录还有自己的子目录，所以管理员很难一眼看出队列文件的完整路径。倘若你还在使用不支持 -q 选项的旧版 postcat，你可以写一个类似例 5-2 的 shell script来模拟 postcat -q 的效果。这个脚本需要一个 Queue ID 自变量（所以一次只能显示一个队列文件），以检查所有队列目录来找出这个文件，然后以得到的完整路径来调用 postcat，借此显示队列文件的内容。

例 5-2：旧版 postcat 的封装程序

```
#!/bin/sh

PATH=/usr/bin:/usr/sbin
QS="deferred active incoming maildrop hold"
QPATH=`postconf -h queue_directory`

if [ $# -ne 1 ]; then
        echo "Usage: pfcat <queue id>"
        exit 1
fl
```

```
if [ `whoami` != "root" ]; then
        echo "You must be root to view queue files."
        exit 1
fi

if [ ! -d $QPATH ]; then
        echo "Cannot locate queue directory $QPATH."
        exit 1
fi

for q in $QS
do
        FILE=`find $QPATH/$q -type f -name $1`
        if [ -n "$FILE" ]; then
                postcat $FILE
                exit 0
        fi
done

if [ -z $FILE ]; then
        echo "No such queue file $1"
        exit 1
fi
```

清空邮件

要求Postfix立刻投递滞留在队列里的邮件的操作称为清空（flush），执行清空动作的命令是 `postqueue -f`。不过，除非你有理由确定邮件一定能成功投递出去，否则，最好还是让`qmgr`自己决定重新投递的时机。不断地反复要求清空，会严重影响邮件服务器的效率。

使用 `-s` 选项可清空寄到特定站点的邮件，而且收信站点必须要有接受快速清空（fast flush）的“资格”才有效。要使得某站点具备此资格，你必须将该站点的主机名称或网域名称列在 `fast_flush_domains`参数中。此参数的默认值只包含 `relay_domains` 所列的所有网域，但是你可以视情况增加额外的站点：

```
fast_flush_domains = $relay_domains example.com
```

如果你有一个间歇性的邮件交换器，你可以在该交换器上线的时间内，使用 `postqueue -s`清空先前无法送达到该交换器的所有邮件：

```
# postqueue -s example.com
```

关于快速清空与SMTP ETRN命令，请参阅第九章。

E-mail 与DNS

域名系统（Domain Name System，DNS）是一个世界级的分布式数据库，其主要任务是将“主机名称”对应到“IP地址”上；此外，对于邮件路由（E-mail routing）的决定，DNS也扮演了重要角色。本章介绍MTA与DNS之间的合作关系以及与DNS相关的Postfix配置参数。当你架设邮件服务器时，要记住DNS在以下两方面的影响：

在寄信功能方面，运行Postfix的系统必须要能够访问一部可靠的DNS server，供其查找主机名称、查询邮件路由信息。

在收信功能方面，你的网域的DNS server必须要能够让外界找到你的邮件服务器。

架设邮件服务器时，最常见的问题的原因就是DNS server没设定好。

DNS概论

在Internet设立初期，DNS还没发明之前，主机名称与IP地址的对应关系是记录在一个超大的文本文件中，这个文件放在一个中央站点，供Internet上的每个主机自由访问。每个主机都必须定期从中央站点下载最新版本，才能获得最新的主机名称信息。可想而知，这样的方法过不了多久，便会因为数据文件过于庞大，而使得名称地址信息难以散布。DNS就是在这种环境下所提出来的构想。 1983年颁布的RFC 882正式定义了DNS的关键功能与实施细节，其中最重要的两项观念是：数据是分散的和主机名称采取层次结构式命名法则。将数据打散，意味着各站点有权更新自己的信息，并随时反映实际情况。层次结构式的命名法则可避免主机名称冲突，并形成今日我们非常熟悉的网域名称系统每个机构（网站）至少有一个网域名称，附属于同一网域的所有主机，其完整名称（FQDN）由自己的简名与网域名称组成。例如，若某机构的网域名称是`example.com`，

该机构的所有主机都可以使用任何类似 server1.example.com、hp4100.example.com 或 www.example.com这样的名称。

每一个网域至少要有两部权威域名服务器（ authoritative name server)。能够直接访问到第一手网域信息的域名服务器，称为该网域的权威域名服务器。

网域数据由各种类型的资源记录（resource record）所组成。不同类型的资源记录，代表不同种类的信息，诸如 IP地址、域名服务器、主机别名、邮件交换器等。对邮件系统而言，我们只关心下列四种资源记录：

A

A 记录代表“ 主机名称 ”与“ IP 地址 ”的对应关系，其作用是将名称转换成 IP 地址。DNS 使用 A记录回答“ 某主机名称所对应的 IP地址为何？ ”这样的问题。人类习惯使用容易联想的名称，不擅处理无意义的数字，但是计算机硬件则刚好相反。主机名称必须靠 A 记录转译成 IP地址，网络层才知道如何选择路由，将邮件包送到目的地。

CNAME

某些名称并没有对应的 IP地址，而只是一个主机名称的别名。 CNAME记录代表“ 别名 ”与“ 规范主机名称 ”(canonical hostname ）之间的对应关系。管理员常利用 CNAME 将 HTTP或 POP之类的公共网络服务的 request导向实际提供服务但是名称不同于公共所知的服务器。举例来说，管理员可能公告他们网站的主机名称为 *www.example.com*，但其实该名称在大部分时间只是一个指向 *server1.example.com*的 CNAME记录而已。在 *server1.example.com*的例行维护期间，管理员可暂时将 *www.example.com*指向备用的 *server2.example.com* 服务器。

MX

MX记录提供邮件路由信息。这类记录提供网域的“ 邮件交换器 ”(Mail Exchanger)的主机名称以及相对的优先值。当 MTA要将邮件送到某个网域时，会优先将邮件交给该网域的 MX主机。同一个网域可能有多个邮件交换器，所以每一个 MX记录都有一个优先值，供 MTA 作为选择 MX 主机的依据。

PTR

PTR记录代表‘ IP地址 ”与“ 主机名称 ”的对应关系，其作用刚好与 A 记录相反。DNS 系统使用 PTR记录来回答‘ 某IP地址所对应的主机名称为何？ ”这样的问题。按照 RFC 882 的构想，PTR记录应该要能够与 A记录互补匹配，也就是说，从主机名称顺查出来的 IP 地址，再经过反查（从 IP 地址查出主机名称）之后，应该能得到原本的主机名称。不过，如果同时有多个主机名称对应到同一个 IP地址，则 PTR记录应该指回该 IP地址的规范主机名称。某些网络应用 PTR记录来检验客户端的主机名称是否可信。

决定邮件路由

MTA如何在DNS的辅助下将邮件送到正确目的地？这个内容与其直接描述算法，不如直接举例说明。假设example.com网域有一位名为Horatio的用户，他的工作站主机名称为denmark。他可以直接使用*horatio@denmark.example.com*这个邮件地址来收信，因为外界的MTA可以直接查出*denmark.example.com*的IP地址，将邮件送给该工作站的*horatio*用户。不过，这表示Horatio的工作站必须永远保持开机，而且还必须运行一个MTA,随时等待接收来自Internet的陌生MTA送来的邮件。倘若example.com网域里的每一台工作站都采用这种方法，则该网域的邮件管理人员大概会不堪重负，因为他们要管理几百个甚至几千个MTA，而且这些MTA全部都必须暴露在Internet上，安全堪虞。所以，与其让几百台计算机做同样的事，不如设置一台专门的邮件交换主机，统管整个网域的所有邮件，代替网域内的所有主机接收、寄发电子邮件。但问题来了，MTA如何知道特定网域的邮件交换主机是哪一台？答案就在DNS系统的MX资源记录。

邮件交换器的动作不外乎两种：投递它所收下的邮件，或是将邮件转寄到另一个邮件系统。为了保险起见，同一个网域可能同时设置多台MX系统，对于这样的网域，其DNS数据库也必须要提供同样多的MX记录。一般而言，在一个设置多台MX服务器的网域里，其中一个MX会被当成主要的邮件服务器，而其他MX服务器当成备用系统。MTA依据MX记录的优先值，选择适当的MX服务器来递交邮件。

BIND（译注1）是最常用的DNS服务器软件之一。以*example.com*网域为例，该网域的BIND数据库文件应该如下：

```
example.com. IN SOA ns.example.com. kdent.example.com. (
     1049310513
     10800
     3600
     604800
     900 )

;
; Nameservers
;
example.com.    IN NS ns.example.com.

;
; Host Addresses
;
example.com.           IN A 192.168.100.50
server1.example.com.   IN A 192.168.100.220
```

译注1：关于DNS系统与BIND软件的完整解释，请参阅Paul Albitz和Cricket Liu合著的《DNS & BIND》（O'Reilly出版）

```
ns.example.com.         IN A 192.168.100.5
mail1.example.com.      IN A 192.168.100.50
mail2.example.com.      IN A 192.168.100.54
mail3.example.com.      IN A 192.168.100.123

;
; Mail Exchangers
;
example.com.     IN MX 10 mail1.example.com.
example.com.     IN MX 20 mail2.example.com.
example.com.     IN MX 30 mail3.example.com.

;
; CNAME Records
;
pop.example.com.        IN CNAME mail1.example.com.
www.example.com.        IN CNAME server1.example.com.
```

就本章的讨论主题而言，我们只关心以下三个 MX资源记录：

```
example.com.     IN MX 10 mail1.example.com.
example.com.     IN MX 20 mail2.example.com.
example.com.     IN MX 30 mail3.example.com.
```

第一栏是网域名称，第二栏（IN）表示这些资源都在Internet 上，第三栏（MX）表示它们都是“邮件交换器”（Mail eXchanger）的资源记录，倒数第二栏是优先值，最后一栏是主机名称。这些字段之中，唯一需要解释的是优先值。优先值的有效范围在 0～65536 之间，绝对的优先值没有意义，只有在互相比较时才有意义；数值越低者，优先程度越高。习惯上，大多数管理员都选择 10 的倍数来做优先值，这让我们在临时需要改变优先级时，可以有“安插”的空间。

在上述的案例中，*mail1.example.com*是 *example.com* 最主要的邮件交换中心，它承接所有寄到该网域的邮件。假设某 MTA有一封邮件要寄给 `example.com` 网域的用户，它会先取得此网域的所有 MX 记录，并尝试将邮件递送给优先值最低者（优先程度最高）。假设 `mail1.example.com`在线，而且能够顺利收下邮件，则传信过程就到此结束。然而，如果 `mail1` 临时因为某种原因而无法收信，则 MTA 会尝试将邮件交给第二位 (`mail2.example.com`)；如果`mail2`能够收下邮件，后续的投递过程就由 `mail2`负责完成。一般而言，备用的MX 会等待主要的 MX恢复运作后，将先前代收的邮件全部交回给主要的 MX。

以上是收信网域能提供 MX资源记录的情况，但如果没有呢？在这种情况下， MTA会检查收信网域名称本身是否有一笔 A记录，如果有，则 MTA会尝试将邮件传给该 IP地址上的主机。

现在将谈谈同一网域的 MX主机之间如何交换邮件。先假设MX 上的 MTA 也使用前述

法则来选择邮件寄送对象，会发生什么事。假设位于mail2.example.com 的 MTA收到了一封写给 *ophelia@example.com* 的邮件（当然，这时候的mail1.example.com 是在离线状态下，否则 mail2 也就不会收到邮件了）。在 *mail2.example.com* 上的 MTA取得example.com网域的所有的MX记录后，决定将邮件传给 mail1.example.com，但因为 mail1 离线了，所以它选择第二顺位的 MX 主机，也就是它自己。很显然地，寄信给自己是没有道理的，所以它接着选择第三顺位的mail3.example.com 。在 mail2 将邮件交给 mail3 之后，mail3 将历经一次相同的抉择过程，而邮件又立刻传回到mail2，造成邮件循环（mail loop）（译注 2）。

为了避免邮件循环，必须稍微修正 MTA的MX挑选法则。当MTA发现自己被列在MX记录中时，它会放弃自己的那些记录以及所有优先值等于或大于自己（较低优先程度）的 MX记录。也就是说，真实情况是 mail2在查出 example.com网域的所有 MX记录后，它会先剔除自己与 mail3，只剩下 mail1 。由于 mail1 当时仍处于离线状态，所以mail2已经没有任何选择了，因此，邮件会留在mail2的等待队列里，直到mail1恢复连接时才寄出去。

知道了MTA如何使用 DNS来挑选收信主机之后，就可以开始讨论如何正确设定MX资源记录。总而言之，在规划 DNS网域数据库时，必须遵守下列原则：

所有 MX主机都必须要有合法的 A资源记录。MX记录所指的主机名称，必须要有一笔有效的 A记录。因为 MTA在选出收信主机之后，还必须查出其IP地址才能连接。

MX 记录不可以指向别名。MX记录所指的主机名称，不应该是别名（CNAME记录）。在正常情况下，当MTA 在检查 MX记录列表时，是依据自己的“规范名称”是否在列表中，而决定要不要排除优先程度低于自己的 MX主机。为了避免邮件循环，请确定列在MX 记录中的是规范名称，而非别名。就算MTA能接受 CNAME记录（如果 MTA能够查出并使用规范名称），也可能在递送邮件时造成问题。

MX 记录应该指向主机名称，而非IP地址。虽然目前将MX记录指向 IP地址还不会出状况，但是依据 RFC 974的声明，你应该使用主机名称才对。如果将来网络有所改变（比方说，升级到 IPv6），纯IP地址就可能会造成邮件递送问题。

指定明确的优先值。虽然 BIND容许你不指定 MX记录的优先值，但是这样做会引发非

译注 2: 这里描述的虚构情节与实际情况稍有出入。当 MTA在进行比较时，实际上是先查出主机名称的IP地址，因为 MX主机可能有多笔 A记录。Postfix将查出的IP地址与自己的inet_interfaces 和 proxy_interfaces 所列的地址群做比较，借此判断应该收下邮件还是转交出去。

必要的不确定性。问题严重程度要看寄信方的MTA与DNS server而定，最糟的情况甚至会导致你的MTA收不到信。

Postfix与DNS

寄出邮件时，Postfix使用系统的*resolver*（即DNS client，能向DNS server查询网域信息的函数库）来取得DNS信息。收信时，你的网域的DNS必须要能提供递送信息（MX或A记录）给外界查询，让其他MTA能够找到你的Postfix server。本节探讨收信程序与寄信程序所涉及的DNS课题。

DNS对于寄信程序的影响

Postfix的smtp MDA必须要能够取得MX与A记录，才能决定收信主机的名称与IP地址。也就是说，在寄信程序中，Postfix至少要进行两次DNS查询，第一次是为了取得MX主机名称，另一次是取得该主机名称的IP地址。由于Postfix使用操作系统的resolver来访问DNS server，因此，运行Postfix的系统至少要能够访问一台DNS server。虽然大部分运行Postfix的系统通常也会运行DNS server，但是这并非必要条件。

如果你怀疑自己的系统不能够正确地查询DNS信息，有三个命令行工具可以帮你找出问题：`nslookup`、`dig`以及`host`。绝大部分Unix系统都应该安装这三个工具，关于它们的详细用法，请参考相关的说明文件（先前提到的《DNS and BIND》是个不错的选择）。你可利用这些工具查出特定网域的所有类型的资源记录，包括让Postfix能将邮件送到正确主机的MX记录。

DNS问题可能源自于Postfix所在系统本身的设定错误，也可能是目标网域的DNS server没有设定好。当你寻找问题原因时，有非常重要的一点必须谨记在心：Postfix总是先查MX记录，然后再查A记录。即使你可以顺利查出网域名称的IP地址，但是如果该网域没有提供MX信息，Postfix不一定能够在第一次就顺利寄出邮件。

配置文件选项

投递邮件时，Postfix从DNS系统查出目标网域的所有MX资源记录，然后依据优先值来排序。在Postfix顺利连接到收信方的SMTP server之后，对方会以状态码来回答smtp MDA的要求。介于2xx范围之间的状态码代表答应要求；若是返回4xx范围内的状态码，表示对方遇到了暂时性的错误；如果是状态码在5xx范围内，表示发生永久性问题。关于SMTP状态码，请参阅第二章。

为了兼容于 Sendmail，在默认情况下，Postfix将回答 4xx与 5xx状态码的 SMTP server一律当成完全没有响应。如果你不愿意 Postfix 忽略 MX server 回复的错误状态码，而要其如实反应状态码的含义，请设定 smtp_skip_5xx_greeting 和 smtp_skip_4xx_greeting参数如下：

```
smtp_skip_4xx_greeting = no
smtp_skip_5xx_greeting = no
```

当 smtp_skip_4xx_greeting 被设定成 no，而 Postfix 在寄信到 MX server时遇到 4xx状态码，它将不会把邮件寄到目标网域的其他备用 MX serevr，而是放在自己的等待队列里，等待下次递送时机。

当 smtp_skip_5xx_greeting 被设定成 no，而 Postfix在寄信到 MX server时遇到 5xx状态码，它将不会把邮件寄到目标网域的其他备用 MX serevr，而直接将邮件退回给原寄件人。

某些网域的 MX 记录被设定成具有相同优先值。在默认情况下， Postfix smtp MDA 会随机挑选其中一个 MX地址。你可以通过 smtp_randomize_addresses参数来控制这种行为：

```
smtp_randomize_addresses = no
```

这将使得 Postfix 依照取得 MX记录的顺序来决定收信主机。

PTR 记录

为了防治垃圾邮件，现在有许多 SMTP server 要求客户端的 IP 地址必须要能够查出有效的 PTR 资源记录。因此，你的 Postfix 系统的 IP地址必须在 DNS 系统里有一个指向 Postfix 主机规范名称的 PTR记录，这样才能确保所有 SMTP server 都愿意收下你寄出的邮件。

DNS 对于收信程序的影响

要让Postfix 收下特定网域的邮件，运行 Postfix的主机本身名称必须被列在该网域的 MX记录里，而且 Postfix 也必须被设定成可以收下该网域的邮件。Postfix能收下三种网域的邮件：系统本身所处的本地网域（local domain)、转发网域（relay domains)、虚拟网域（virtual domains)。其中虚拟网域可能是使用虚拟别名（virtual aliases）或虚拟邮箱（virtual mailboxes)。每一种网域都必须被列在特定的 Postfix 参数中才有效。表 6-1 是各种网域所对应的 Postfix 参数。

表 6-1：网域类型与相关参数

网域类型	参数
本地	mydestination
转发	relay_domains
虚拟邮箱	virtual_mailbox_domains
虚拟别名	virtual_alias_domains

同一个网域不可同时列于不同参数中。如果 Postfix在两个参数找到同一个网域，它会发出警告。如果你在日志文件看到“ mail for example.com loops back to myself”这样的错误信息，通常是因为你的 Postfix server被列在某网域的 MX 列表里，但是 Postfix却没被设定成可以收下该网域的邮件。

如果你的 Postfix server 要收下 example.com和 porcupine.org这两个本地网域的邮件：则 *main.cf*配置文件里的 mydestination 参数应该像这样设定：

```
mydestination = example.com, porcupine.org
```

关于虚拟邮箱与虚拟网域，请参阅第八章；第九章将讨论转发网域的设定方法。

常见问题

DNS设定不当所造成的问题，通常没有立即可见的效果，只能从日志文件里的错误信息来分析。本节列举几种常见的DNS相关问题，包括你会在日志文件里看到什么信息、问题原因以及解决办法。

“mail for domain loops back to myself” （某网域的邮件绕回我自己）

所有关于DNS 的错误中，这可能是最常见的。问题原因是你的 Postfix server 被列在某网域的 MX列表里，但是你没让 Poistfix知道，它自己就是该网域的邮件终点站。所以，解法就是将该网域列在 mydestination参数，或是设定为虚拟网域或转发网域。倘若你的 Postfix server是列在 Porxy或NAT 设备之后，它可能无法察觉自己是某网域的 MX 主机，在这种情况下，你得将 Proxy系统的 IP 地址加到 proxy_interfaces 参数。这类错误留下的日志记录，看起来类似下面这样：

```
postfix/qmgr[3981]: 2CC3B229: from=<heloise@ora.com>,\
      size=306, nrcpt=1 (queue active)
postfix/smtp[3983]: warning: mailer loop: best MX host for\
      example.com is local
postfix/smtp[3983]: 2CC3B229: to=<abelard@example.com>,\
      relay=none, delay=0, status=bounced (mail for example.com
      loops back to myself)
```

"Host found but no data record of requested type"（能找到主机，但是没有指定数据类型的记录）

网域的 DNS 数据库里找不到 MX 记录，而且网域名称自己也没有 A 记录。你得要联络该网域的 DNS 管理员才能解决这问题。如果你自己就是 DNS 域名数据库的管理员，请确定你的邮件服务器的主机名称确实列在该网域的 MX 记录里。这类错误留下的日志记录，看起来类似下面这样：

```
postfix/qmgr[3818]: D31CD20F: from=<heloise@ora.com>,\
      size=312, nrcpt=1 (queue active)
postfix/smtp[3824]: D31CD20F: to=<abelard@example.com>,\
      relay=none, delay=1, status=bounced (Name service \
      error forname=example.com type=A: Host found but \
      no data record of requested type)
```

"no MX host for domain has a valid A record"

网域的 DNS 数据库有 MX 记录，但是查不出 MX 主机名称所对应的 IP 地址。你得要联络该网域的 DNS 管理员才能解决这问题。对于你自己的网域，请确定 MX 记录所列的每一个主机名称，都有一个有效而且正确的 A 记录。下面是这类错误留下的日志记录：

```
ostfix/qmgr[3818]: 068DB20F: from=<heloise@ora.com> \
     size=306, nrcpt=1 (queue active)
ostfix/smtp[3846]: warning: no MX host for example.com has
     a valid A record
ostfix/smtp[3846]: 068DB20F: to=<abelard@example.com> \
     relay=none, delay=1, status=deferred (Name service \
     error for name=mail.seaglass.com type=A: Host not found)
```

"Host not found, try again "（找不到主机，再试一次）

从 DNS 系统中查询不出任何有意义的结果。有可能是 DNS server 断线或拒绝服务，甚至故障。如果能确定该网域的 DNS server 正常运作无误，问题可能是网络，或是 Postfix server 系统本身的 resolver 没有设定妥当。对于最后一种可能，你应该检查系统的 */etc/nsswitch.conf* 与 */etc/resolv.conf* 配置文件。要知道自己系统的 resolver 是否有作用，请使用 nslookup、host、dig 之类的系统工具进行测试。下面是这类错误留下的日志记录：

```
postfix/qmgr[3818]: CCBED1E8: from=<heloise@ora.com> \
      size=306, nrcpt=1 (queue active)
postfix/smtp[3937]: CCBED1E8: to=<abelard@example.com> \
      relay=none, delay=1, status=deferred (Name service error \
      for name=example.com type=MX: Host not found, try again)
```

如果你的 Postfix 是在 chroot 环境下运作，你必须将那几个与 DNS 有关的配置文件（尤其是 */etc/resolv.conf*、*/etc/hosts*、*/etc/nsswitch* 等）也复制一份放在 chroot 环境下的对应目录（比方说，将它们复制到 */var/spool/postfix/etc/* 目录下）。关于如何将 Postfix 放在 chroot 环境下，请参阅第四章的“ chroot”。

第七章

本地投递与 POP/IMAP

SMTP 协议要求收下邮件的 MTA，必须负责将邮件送到最终目的地 —— 可能是本地系统的邮箱，也可能同一网络上的其他主机，这段工作过程称为投递（delivery）。本章与接下来的第八章和第九章，讨论 Postfix 收下邮件之后的投递过程。

本章探讨 Postfix将本地邮件投递到邮箱的过程以及 POP/IMAP server 如何访问邮箱。许多用户常误以为收信与寄信是同一套软件在工作，其实不然。第一章曾解释过，让用户能从邮箱取走邮件的协议是 IMAP 与 POP，而 Postfix 的职责是将收下的邮件放入邮箱。也就是说，POP/IMAP 服务是由 Postfix之外的软件所提供的。提供 POP/IMAP 服务的软件很多，本章描述的内容，应该适用于任何遵守标准规范的 POP/IMAP 服务器软件，包括 Popper、WU IMAP Kit 等。本章最后一节会示范 Postfix 如何与 Cyrus IMAP server 合作。

在我们探讨本地邮件的投递过程之前，先让我们理清“本地”、“外地”、“虚拟”这三种邮件的定义以及相关的 MDA。

Postfix 的投递代理程序

Postfix 依据来信地址来决定是否要收下邮件以及如何选择适当的 MDA来执行后续的投递任务。Postfix 会收下三种网域的邮件，分别是本地（local）、转发（relay）以及虚拟（virtual），它们的定义与相关的 MDA，分述如下：

本地邮件

若邮件终点站是 `mydestination` 参数所列出的网域之一，Postfix就视其为本地邮件，由 local MDA （或是你指定的其他程序）执行投递任务。本地邮件的收件者必须拥有本地系统（Postfix server 本身所在的主机）的用户账号，或是其名称被定

义在别名文件（传统上是 */etc/aliases*）。本地邮件会被投递到系统的邮件存储目录（通常是 */var/spool/mail/*），或个人主目录下的邮件文件（*~/mail/*）。

转发邮件

若邮件终点站是 relay_domains 参数所列出的网域之一， Postfix 就视其为转发邮件，由 relay MDA 来执行投递任务。一般而言，只有在 Postfix 被当成局域网络的邮件网关使用，而同网络上还有其他网域的邮件服务器时，才会让 Postfix 收下转发邮件。也就是说，所谓的“转发”，通常是指同局域网络上的其他主机，而 relay 其实是 smtp MDA 的翻版，只不过被刻意设计成特别适合传信给局域网络上的主机而已。关于转发邮件的投递操作，请参阅第九章的说明。

虚拟*网*域邮件

一台邮件服务器通常只服务一个规范网域（canonical domain）；如果要同时服务多个网域，则额外的网域称为虚拟网域（virtual domain）。虚拟网域的邮件由 virtual MDA 负责投递。依据用户是否有服务器的系统账号，虚拟网域的邮件还可分成“虚拟邮箱”与“虚拟别名”两种。虚拟邮箱的收件人没有系统账户，而且每一个虚拟邮箱网域都有自己的邮箱目录（mail spool），所有虚拟邮箱网域都必须列于 virtual_mailbox_domains 参数。另一方面，虚拟别名网域的收件人可以拥有本地或非本地系统的账户， Postfix 会改写这类邮件的收件地址，然后交给 smtp MDA 递送出去（如果新地址是非本地网域），或重新回到收件队列（如果新地址是本地网域）。关于虚拟网域的课题，请参阅第八章。

本章内容只讨论本地投递操作的细节。

邮箱格式

当 Postfix投递本地邮件时，邮件内容会被传送到 Postfix 系统上的适当邮箱。最常见的两种邮箱格式，分别是传统的 *mbox* 以及较新的 *maildir*。两者都是使用一般的文件来储存邮件内容，差别在于文件内部的组织安排有所不同。在 Postfix 中，当你设定任何邮件文件或目录参数时，如果在路径末端加注一个／符号，表示你想使用 *maildir* 格式的邮箱（参阅本章“邮箱投递操作”的示范）。

mbox 格式

传统上， Unix系统将同一位用户的所有邮件都塞在同一个文件里，像这样的邮箱格式通常称为 *mbox*。邮箱文件里的每一封邮件，其第一行的前五个字符必定是“ *From*”（第五个字符是空格）。习惯上，为了方便表示，我们通常将它写成“ From_”，以下划线字符强调空格的存在。请勿将 mbox 文件内用来分隔邮件的“ From ”字样与邮件标头里的

“From：”字段混为一谈。邮件在 mbox文件里的最后一行必定为空格。因此，一行空格接着一个 From_ 字样，就可视为下一封信的开始。

完整的 From_ 文本行看起来像下面这样：

```
From jmbrown@example.com  Wed Feb 25 16:07:57 2004
```

如你所见，它的开头是“ *From* ”接着一个空格。在空格之后是邮件地址（通常是该邮件的信封地址），然后是收件日期（以 Unix标准的 24字符日期格式表示）。mbox 格式允许日期之后可接着一段选择性的注释字符串。

Postfix 将邮件写入 mbox文件之前，会先使用信封上的寄件人地址与当时的日期创建好 From_ 文本行，并将该行字符串写到 mbox 文件的末端，然后才开始填入邮件内容。如果 Postfix 发现邮件内容本身有任何以“ *From* ”开头的文本行，它会在该文本行的开头加一个＞符号，避免该文本行被误认为下一封信的开头。

当 POP/IMAP server读取 mbox文件内的邮件时，第一步是扫描文件内容，找出代表邮件开头的 From_ 文本行。在读取邮件内容时，如果遇到下一个 From_ 文本行（或文件结尾），就可断定当前的邮件已经读完了。有些 POP/IMAP server 会主动恢复“>From”的原状，但有些不会。

由于 Postfix 和 POP/IMAP server 有可能会同时访问同一个 mbox 文件，所以它们必须使用“ 文件锁定机制 ”(file locking）来确保访问权。在 local 投递本地邮件之前，必须先将该文件加锁，然后才能将邮件内容写入 mbox 文件。Postfix支持多种锁定机制，视系统平台而定。利用 postconf -l命令可查看你的系统提供了哪些锁定机制可供 Postfix 使用：

```
$ postconf -l
flock
fcntl
dotlock
```

如果想知道Postfix在你的系统上列出的各种锁定机制的详细信息，请把锁定机制的名称告诉 man：

```
$ man flock
```

如果你的系统平台支持 flock和 fcntl，应该就可以找到它们的在线说明文件，因为这两者都是操作系统或函数库提供的功能，而任何系统平台都支持的 dotlock 机制，很可能找不到说明文件，因为 dotlock只是程序之间的一种不成文协议，不需要额外的函数库。dotlock 的原理很简单，举个例子就可以说明清楚。假设Postfix 要访问 user1 邮件文件，它必须先检查该文件的同目录下是否存在一个 .user1.lock文件，如果存在，表示 user1 邮件文件当前被另一个进程占用；如果 .user1.lock 文件不存在，

Postfix就自己产生一个，让其他进程知道 user1 文件当前正被占用。在 Postfix 关闭 user1 文件之后，要主动移除 .user1.lock文件，让其他进程可以使用 user1 邮件文件。dotlock 锁定机制的缺点是它没有强制性（任何进程都可以不检查 .user1.lock 是否存在而径直访问 user1 文件），而且效率不佳。

通常你可以不必担心锁定机制的细节，也不必理会系统支持哪些类型的锁定机制，因为Postfix能自动做出最佳选择。

maildir 格式

maildir 邮箱格式不同于 mbox之处，在于它使用目录结构来存储邮件。 maildir的设计原意是为了解决 mbox格式的可靠性与文件锁定问题。例如，如果在邮件内容还没完全写入 mbox 文件之前，系统就死机了，这时候可能只有部分内容在邮箱里。当系统恢复运行，MDA 将邮件写入邮箱时，新的内容会接在前次残缺内容的后面，因而造成问题。

mbox 格式的另一个缺点，是发生在 POP/IMAP server 与 SMTP server 试图同时开启同一个邮箱时。如果双方没使用相同的锁定机制，邮箱文件可能因此受损。先前说过，保护文件的锁定机制有好几种，但是并非所有邮件程序都使用锁定机制。但如果使用 maildir格式，则可以不使用文件保护锁，因为每一封邮件都是存放在单独的一个文件里。因此，不同的邮件程序，不可能同时访问同一个文件。

一个 maildir风格的目录，其下有三个子目录：*tmp/*、*new/*以及 *cur/*。这些子目录与它们的上层目录必须位于同一个文件系统中，习惯上，它们应该放在用户的主目录（home directory）的邮件目录下：

```
$ ll /home/kdent/maildir
total 12
drwxr-x---    2 kdent    kdent        4096 Mar 13 12:24 cur
drwxr-x---    2 kdent    kdent        4096 Mar 13 12:24 new
drwxr-x---    2 kdent    kdent        4096 Mar 13 12:24 tmp
```

在 *new/* 目录下的邮件文件，是 MDA 已经送达但是尚未被用户阅读的信，文件本身的修改时间，就是收下邮件的时间。邮件文件通常包含 RFC 2822格式的邮件，而且不需要 "From_"。

用户看过邮件之后，邮件文件会被转移到 *cur/* 目录。*tmp/* 目录供 MDA 将邮件内容存储成文件，在确定全部内容都写入文件之后，邮件文件会被搬到 *new/* 目录。

应该选择 mbox 还是 maildir?

这个问题没有简单的答案。哪一种邮箱格式最适合你，取决于许多因素。 mbox格式的

好处是几乎全世界都支持，但也正是因为它有文件锁定的问题，而导致了 maildir格式的出现。而 maildir格式在规模适应性方面也颇受质疑，因为某些文件系统可能无法应付太多的邮件文件。在效率方面，两种格式各有优缺点：搜索、访问、删除特定邮件时，maildir的速度比较快；但是就 MDA的投递工作效率而言，直接将邮件内容放入文件未（mbox格式）可能比较快。实际上，你的选择可能要看你所用的 POP/IMAP server而定。如果你架设的 POP/IMAP server 只支持 maildir格式，很显然你没有选择的余地。 Postfix对两种格式都支持，所以你只要考虑其他因素就行。如果你的环境让你感觉到为难，建议你测试两种格式，尽量以接近实际的运行环境和工作量来实验，依据实验结果来做出选择。

本地邮件的投递操作

若收件地址的网域部分，是列在 `mydestination` 参数的所有网域之一， Postfix 就将其当成本地邮件交给 local MDA进行投递。你可以随意在 `mydestination`列出多个网域，但是本地的个别用户会收到所有网域的邮件。举例来说，如果 *oreilly.com*、*ora.com* 和 *oreillymedia.com* 同时被列在 `mydestination` 参数，则寄给 *kdent@ora.com*、*kdent@oreilly.com*或 *kdent@oreillymedia.com*的邮件，最后都是进入同一个本地邮箱。

为了避免收下不明用户的邮件，所有本地收件人的名称都必须列在`local_recipient_maps`参数所指的表中。此参数的默认值是指向 Unix 系统的密码文件与别名表，所以你通常不需要修改它。

检查邮件地址的人名部分时， Postfix先检查别名表（参阅第四章）。如果发现相符的别名，则以该别名对应的名称为新的收件人，重新提交邮件，当成新邮件处理；否则，就试着将邮件传给系统上的用户。 Postfix 先检查当地用户是否设置了自己的 *.forward* 文件，如果有，则依据其设定内容来转寄邮件（重新提交邮件）；如果没有，则将邮件放入用户的邮箱。

.forward 文件

.forward 文件让用户可以设置自己的别名。*.forward* 文件的格式与别名文件的 RHS-value部分的格式一样，甚至比其更宽松。比方说，别名文件的 RHS-value 可以有多个以分号隔开的值，*.forward* 文件也沿用相同惯例，但同时也容许你将值分别写在不同的行。

*.forward*文件的拥有权必须是收件人的系统账户，而且通常放在用户的主目录下。你可以用 `forward_path`参数来改变 *.forward* 文件的存放路径。Postfix提供下列八个变量让你表示 *.forward* 文件的存放路径，这些变量的实际值，由投递时的系统环境决定：

`$user`

收件人的账户名称（信息来源： *letc/passwd*）。

`$home`

收件人的主目录（信息来源： *letc/passwd*）。

`$shell`

收件人的 shell （信息来源：*letc/passwd*）。

`$recipient`

收件人的完整邮件地址。

`$extension`

收件地址的人名部分的扩展部分（不一定有），以 + 之类的分隔符与人名部分隔开。以 *user1+labs@example.com* 为例，`$extension` 等于 `user1`。

`$domain`

收件地址的网域部分。

`$local`

收件地址的完整人名部分（包含扩展部分在内 ——如果有的话）。以 *user1+labs@example.com* 为例，`$local` 等于 `user1+labs`。

`$recipient_delimiter`

收件地址的人名部分与扩展部分之间的分隔符（通常是 +）。

如果你增加对一个非标准的 *.forward* 文件的支持，可以参考下面的设定：

```
forward_path = /home/$user/.forward /home/$user/other_forward
```

关于上述变量的详细信息，请参阅 Postfix `local` 的在线说明。

别名投递操作

当别名文件指定了一个命令或文件时， Postfix必须先将自己的执行身份改成别名文件拥有者，然后以该身份的权限来执行命令，或将邮件内容写入文件。唯一的例外是别名文件拥有者为 *root* 时，这时候 Postfix 使用 `default_privs` 参数所指定的账户（默认值为 *nobody*）。

邮箱投递操作

当 Postfix 将邮件投递给一位本地用户时，它必须将邮件内容写入该用户在系统上的邮箱。Postfix默认使用的邮箱格式是 mbox，当你安装 Postfix 时，它会依据你所用的 Unix

平台类型来决定邮件存储目录的位置。mail_spool_directory 参数可用来指定一个默认之外的目录，而目录路径的指定方式会影响 Postfix选择何种邮箱格式。举例来说，假如你这样设定：

```
mail_spool_directory = /var/spool/mail
```

这表示 Postfix 应该使用 mbox 格式将邮件存在 */var/spool/mail* 目录下。如果你想改用 maildir 格式，则必须在目录名称之后附加一个／符号：

```
mail_spool_directory = /var/spool/mail/
```

你也可以要求 Postfix将邮件放在用户的主目录下。设定一个相对路径给 home_mailbox 参数，表示你想用哪一个文件来作为邮箱：

```
home_mailbox = mbox
```

在路径名称之后附加一个／符号，表示 Postfix 应该使用 maildir 格式的投递程序：

```
home_mailbox = maildir/
```

这会使得 Postfix 将邮件投递到用户主目录下的 *maildir/* 子目录。

注意： 使用 maildir 格式时，Postfix 通常会自动创建必要的子目录与文件 ——如果用户的身份权限足够的话。不过，基于安全上的考虑，如果上层目录的权限模式是 world-writable (755)，则 local MDA 不会创建任何额外的文件或目录。

POP 与 IMAP

Postfix 将邮件内容填入邮箱之后，用户还需要一个渠道才能读到邮件。许多站点提供 POP/IMAP 服务，让用户能通过网络取得他们的邮件。大多数情况下， Postfix 能够与 POP/IMAP servers 合作无间，而且两边都不需要有特殊设定，当然，前提是双方必须都使用相同的邮箱格式。

POP 与 IMAP 的比较

对于不能总是保持网络连接的用户， POP协议比较理想，因为它能让用户连接到邮件服务器，取走所有邮件，然后切断网络连接。用户可以离线读信，因为所有邮件都已经在用户自己的计算机上了。大部分的POP客户端软件，都可以让用户选择在取走邮件之后，是否要清空服务器上的邮箱。如果一段时间没清理邮箱，邮件会逐渐累积，占用越来越多的服务器磁盘空间。 POP的软件相当容易设计，但是它最大的问题，是你的邮件不见

得放在你需要的地方（试想，如果你有一台以上的计算机）。此外， POP 也不支持多邮箱，而且强制要求你必须完整下载邮件之后才能读信。如果想要先看所有邮件的主题，然后再决定要下载哪几封邮件， POP则无能为力。

IMAP 协议是为了克服 POP的缺点而设计的。IMAP让所有邮件都留在服务器上，用户在看信之前必须先联机，联机之后可在远程进行任何控制管理动作，就像所有邮件都在本地端一样。由于所有实质动作都是发生在服务器上，因此，不管用户是在家里的台式计算机上、办公室里的工作站上，还是在出差时所带的笔记本计算机上，都可以看到同样的内容。在功能性方面，IMAP比 POP 更强、更有灵活性。你同样可以离线阅读（将邮件的副本存放在本地端），而且可以拥有多个邮箱，甚至可以只下载邮件的标题（寄件人、主题、日期、是否有附件），然后才决定是否要取得邮件的其余部分。所以，如果你发现某封邮件夹带了一个大文件，而你对该文件并没有兴趣，你也不必苦等漫长的下载过程。

Postfix 与 POP/IMAP Servers

Postfix与POP/IMAP servers 之间的合作相当简单。每当 Postfix收下本地邮件，就将邮件封存在邮箱里。当 POP/IMAP server收到用户的要求时，只要从同样的邮箱取出邮件即可。

图 7-1显示了 Postfix与 POP/IMAP servers 之间的简单合作关系。Postfix与 POP/IMAP server 双方都必须同意使用相同的邮箱格式以及相同的锁定机制。 Postfix 可搭配任何使用传统邮箱格式的标准 POP/IMAP server。你可能会想要调整 mail_spool_directory参数（参阅“ 邮箱投递操作 ”），但是对于大多数的 POP/IMAP servers而言，只要按照标准安装指示安装并启动服务器即可。对于不支持传统邮箱格式的 POP/IMAP servers 而言， Postfix 仍可以使用“ 本地邮件传输协议 ” （Local Mail Transfer Protocol，LMTP）来投递邮件，由 POP/IMAP server 自己将邮件存入邮箱。

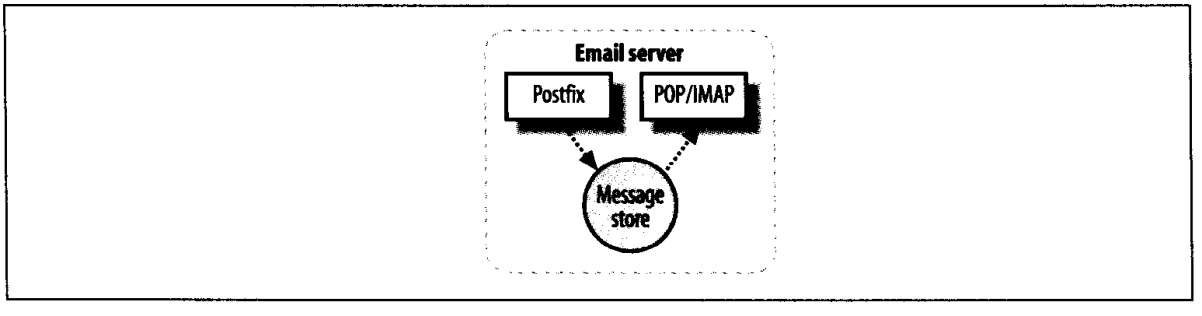

图 7-1: Postfix 与 POP/IMAP servers

本地邮件传输协议（LMTP）

某些 POP/IMAP servers 使用非标准的邮箱格式。很显然地，没道理要求 Postfix 之类的 MTA 必须认识多种不同的专属格式。因此，我们需要一种无关邮箱格式的传输渠道，让邮件能从某个邮件程序传到同机器上的另一个邮件程序，这个渠道便是 LMTP。LMTP 可说是 SMTP 的精简版。LMTP server 同样有权决定是否要收下或拒收邮件，不过，LMTP server 不负责处理无法立即投递的邮件。

当 MTA 传递一封多收件人的邮件给 SMTP server 时，若有部分收件人因故不能收下邮件，则 SMTP server 要负责将邮件排入队列以便下次传送，并对 LMTP client 宣称投递任务已经成功。然而， LMTP server 不承担这样的责任，也就是说，每一位收件人的投递状况都必须个别回复给 LMTP client 知道。对于无法投递的收件人，他们的邮件是放在 LMTP client（MTA）的队列里（而非 LMTP server），由 LMTP client 负责后续的处理过程。

LMTP 的对话可能发生在同一机器上的不同邮件子系统之间，或是同一局域网络上的不同机器之间。但是，如果对话双方中间隔着广域网络，就不保证 LMTP 一定可靠，因为此协议是以响应速度的快慢与否来判断邮件是否顺利送达。 SMTP 已经被发现其收信与送信系统之间有一个同步化问题，偶尔会导致邮件被重复传递。如果 LMTP 的对话双方分居广域网络的两端，相信问题会更严重。

注意： LMTP 除了可让 MTA 用来将邮件投递到非标准格式的邮箱，其真正的好处是让邮件管理人员可以架设出容易扩展且可靠的邮件系统。比方说，对于邮件量很大的站点，可以架设一台或多台 Postfix servers 专门接收来自 Internet 的邮件，然后投递给多个 LMTP 后台系统。当邮件量提升时，只要多架设几台前台或后台系统即可。

最知名的因使用专属邮箱格式，而需要以 LMTP 接收邮件的 POP/IMAP 服务器软件应当是 Carnegie Mellon University 的 Cyrus IMAP server。 从 Project Cyrus 的网站（*http://asg.web.cmu.edu/cyrus/*）可免费取得这套软件。如图 7-2 所示，Cyrus IMAP server 有自己的邮箱格式，Postfix 必须通过 LMTP 协议才能将邮件投递到 Cyrus IMAP 的邮箱。关于如何设定 Cyrus IMAP，请参阅 Dianna Mullet 与 Kevin Mullet 合著的《Managing IMAP》（O'Reilly 出版）。

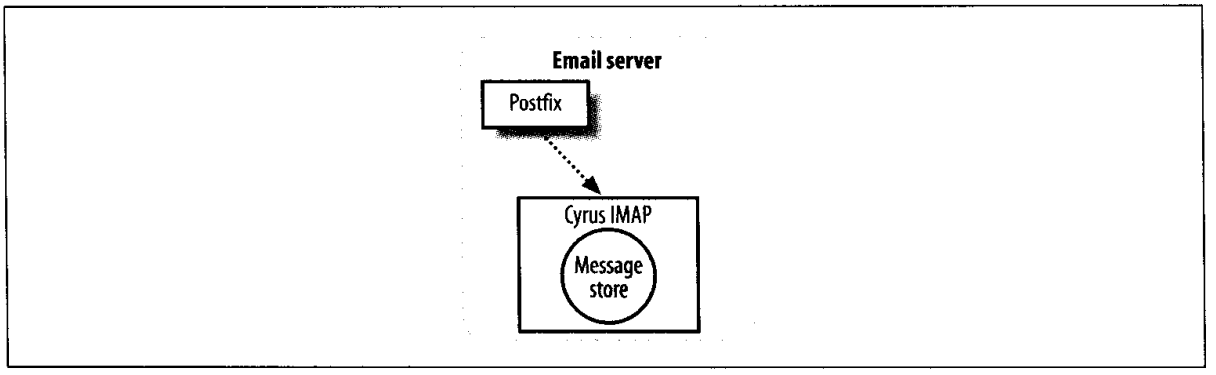

图 7-2：Postfix与 Cyrus IMAP

Postfix 与 Cyrus IMAP

Cyrus IMAP是专门提供 POP/IMAP 服务的服务器，用户不需要系统账户。如果你想要为系统现有的用户架设邮件服务器，或许应该考虑其他比较简单的 POP/IMAP 解决方案，像 Qualcomm 的 Qpopper（只有 POP 功能），或是 University of Washington的 IMAP Toolkit，这两套软件都不需要你在 Postfix 进行任何特殊设定。本节只讨论如何设定 Postfix，使其能够顺利搭配 Cyrus IMAP。

Cyrus IMAP提供两种 LMTP 投递渠道，分别是 Unix-domain socket 与 TCP socket。你必须知道 Cyrus IMAP 使用哪一种渠道，才可以适当设定 Postfix。如果使用 Unix-domain socket，则 Postfix与 Cyrus IMAP server两者都必须在同一台机器上；如果使用 TCP socket，则 Postfix 与 Cyrus IMAP server 可以在同一台机器上，也可以分居于局域网络上的不同主机。Postfix的LMTP 投递参数定义在 *main.cf*配置文件的 `transport_maps` 参数中。

要让 Postfix收下投递给本地 Cyrus IMAP server 的邮件，则收件地址的网域名称必须被列于 `mydestination` 参数（如果想使用 `virtual` MDA 来投递邮件给 Cyrus，请参阅第八章）。接着，你必须要求 Postfix将邮件交给Cyrus IMAP。使用 `mailbox_transport`、`local_transport` 或 `fallback_transport` 参数可让Postfix 知道，在邮件交给 Cyrus之前，要进行多少道本地投递手续。如果你使用了 `local_transport` 或`fallback_transport`，请将 Cyrus的所有用户名称写在 `local_recipient_maps` 参数所指的一个查询表里，以免 Postfix 拒收 Cyrus 用户的邮件。

`mailbox_transport`

邮件先交给 local MDA，由 local 检查别名文件与 *.forward* 文件，并展开收件人的邮件地址。经过处理后的邮件会被转交给 Postfix 的 LMTP client，由它负责投递到 LMTP server（Cyrus IMAP）。

local_transport

当LMTP用于本地传输时，邮件会被直接交给Postfix LMTP client，原本负责处理本地邮件的local MDA完全不插手。因此，别名文件与*.forward*文件都没有作用。

fallback_transport

当LMTP用于备用传输时，Postfix会先将邮件交给local MDA处理，执行别名文件与*.forward*文件的展开操作。如果收件人有一个正常的系统账户，则邮件会被投递到系统上的适当邮箱；如果收件人没有系统账户，则邮件会被交给Postfix LMTP client，由它负责将邮件交给LMTP server。当你的邮件服务器同时以自己的系统账户服务一群用户，而同时又代收另一群Cyrus IMAP server用户的邮件时，就需要以fallback_transport机制来进行LMTP投递操作。

使用下列格式来设定你所选的投递机制：

```
xxx_transport = service:socket_type[:/path/to/socket]
```

对于LMTP投递操作而言，service必须为lmtp（代表*/etc/postfix/master.cf*）所定义的lmtp服务，socket_type必须是unix（代表Unix-domain socket）或inet（代表TCP socket）两者之一。默认值为inet，这表示Postfix假设你的LMTP server是使用TCP socket来收信。因此，如果默认值符合你的需求，你只要这样设定就可以了：

```
local_transport = lmtp
```

如果使用local_transport与Unix-domain socket，在*/etc/postfix/main.cf*中应该做这样的设定：

```
local_transport = lmtp:unix:/var/imap/socket/lmtp
```

Postfix与Cyrus IMAP的搭配实例

Cyrus IMAP使用Cyrus SASL函数库来验证用户的身份，所以，你必须先建构、安装Cyrus SASL函数库，然后才可以顺利建构Cyrus IMAP server。此外，Cyrus软件至少需要Berkeley DB 3以上的版本，如果你系统上的Berkeley DB比第三版还旧，你可能需要全面更新整个系统。在同一个系统上混用不同版本的Berkeley DB，可能会引发难以追查的问题。如果你必须升级DB函数库，建议你重建所有需要用到Berkeley DB的软件（像Perl与Postfix），让系统上的所有软件都使用同版函数库。

请参考Cyrus SASL/IMAP包随附的说明文件来完成编译、安装的工作。某些系统平台可能有已经预先编译好的包可以直接安装。如果在建构Cyrus软件的过程中遇到麻烦，请多花点耐心研究Cyrus随附的说明文件，看看有没有什么可以解决的方法。

为了举例说明，假设我们已经架设好一个能接收 example.com 网域邮件的 Postfix server，并以同一台机器来运行 Cyrus IMAP server。假设所有邮件用户的账户都已经建在 Cyrus SASL 的数据库里，但是这台机器上仍有少数几个 shell账户。虽然这些账户不会被用来收信，但是我们希望仍然能够使用 *.forward*文件与别名文件，以便系统产生给 root 的邮件以及外界寄给 *postmaster* 别名的邮件，都可以被转寄到正确的地方。

很显然地，我们不能直接将所有邮件都直接交付给 Cyrus IMAP server （所以应该排除 local_transport)，因为我们仍需要 local MDA 的别名文件与 *.forward* 文件的展开功能（所以只剩下 mailbox_transport与 fallback_transport可以选择)。但由于 shell账户不是用来收信的，所以，最理想的选择是 mailbox_transport。我们应该将此参数指向 lmtp MDA,并确定 *master.cf*里的 lmtp服务能将邮件投递给 Cyrus IMAP server。以下是我们的设定步骤：

1. 将 Cyrus IMAP 安装到系统上。检查 Cyrus 的配置文件（通常是 */etc/cyrus.conf*)，确定它的服务渠道是 Unix-domain socket ,并记下 socket文件的位置。你应该会看到类似下面这样的内容：

```
SERVICES {
  # add or remove based on preferences
  imap          cmd="imapd" listen="imap" prefork=0
  pop3          cmd="pop3d" listen="pop3" prefork=0
  # LMTP is required for delivery
  lmtpunix      cmd="lmtpd" listen="/var/imap/socket/lmtp" prefork=0
}
```

   其中，“lmtpunix”那一行就是 socket文件的正确位置。

2. 依照 Cyrus随附文件的指示，在服务器上设置一个供 Cyrus IMAP server 使用的系统账户。

3. 检查 */etc/postfix/master.cf*的内容，确定 lmtp服务的配置符合你的系统环境。一般而言，你应该像这样设定：

```
lmtp      unix  -    -    n    -    -    lmtp
```

   如果你的 Postfix 是以默认模式安装的，上面这一行应该已经出现在你的 */etc/postfix/master.cf*配置文件里了。要注意的是第五栏（此例中的 n)，它代表 lmtp MDA 是否要处于 chroot环境下。在本例中，由于它必须要能够读取 Cyrus IMAP server的 socket文件，所以此栏是设定为 n。

4. 检查 *main.cf*,确定 mydestination 参数所列的网域包含了你要的收信网域。你可以直接列出，例如：

```
mydestination = $myhostname, localhost.$mydomain,
                $mydomain, example.com
```

或是通过 `$mydomain` 变量来间接设定：

```
mydomain = example.com
mydestination = $myhostname, localhost.$mydomain, $mydomain
```

5. 要求 `mailbox_transport` 使用 *master.cf* 所指定的 `lmtp` 服务，并指向你先前记下的 Cyrus IMAP socket 文件：

```
mailbox_transport = lmtp:unix:/var/imap/socket/lmtp
```

6. 重新加载 Postfix：

```
# postfix reload
```

第八章

虚拟网域

近年来，用同一台主机来搭建多个网域，已经蔚然成风。举个实例来说，*oreillynet.com* 和 *onlamp.com* 其实位于同一台主机，但是从外界的观点来看，它们似乎分别位于两台完全不同的主机上。一个系统通常有一个最具代表性的网域名称，称为正式网域（canonical domain），除此之外的其他网域，则被视为虚拟网域（virtual domain）。每个虚拟网域都可以提供自己的网络与邮件服务，且外界看不出正式网域与虚拟网域的差别。本章介绍多种架设虚拟网域邮件服务器的技术。虽然我们分开讨论每一种技术，但如果你需要以不同方式来处理不同网域的话，其实你也可以混用这些技术。

Postfix 如何投递虚拟网域的邮件，决定了你需要哪些技术。有两个重要因素影响你的 Postfix如何支持多个虚拟网域：

网域之间要有单独的命名空间吗？比方说，写给 *info@oreilly.com*和 *info@ora.com* 这两个地址的邮件，是放在同一个邮箱还是各自单独的邮箱？对于使用相同邮箱的情况，我们称之为共享网域（shared domain）；而另一种情况则称为独立网域（separate domain）。

每位用户都需要系统账户吗？我们将区别“系统账户”与“虚拟账户”的不同。前者是真正的 Unix 账户，可用来登录服务器系统；后者纯粹用于辨识邮箱，不能登录系统，在 */etc/passwd* 里也没有对应记录。

我们将探讨 Postfix对于虚拟网域邮件的四种处理方式：

共享网域搭配系统账户。

独立网域搭配系统账户。

独立网域搭配虚拟账户。

虚拟网域搭配非 Postfix 控制管理的特殊格式邮箱。

应该挑选哪一种技术，主要决定因素是你所使用的 POP/IMAP server 。如果你的 POP/IMAP server 不能辨认虚拟网域，那么，很可能所有邮件地址都需要系统账户。某些 POP/IMAP servers 天生就是为了支持多重网域而设计的，而且邮件是存放在本地文件系统的某个特定目录结构下，还有些 POP/IMAP server 使用自己专属的邮箱格式。对于 Postfix 不支持的邮箱格式，Postfix 可使用 LMTP 将邮件交给 POP/IMAP server。

不管采用哪一种技术，你的正式网域的 DNS 是怎么设定的，所有虚拟网域的 DNS 都必须“依样画葫芦”。关于 DNS 与 Postfix 的交互关系，请参阅第六章。

共享网域搭配系统账户

最简单的虚拟网域模式，是每位用户都可以收到每个网域的邮件。就用户的感受而言，就好像同一个邮箱有多个地址一样。这种模式的设定方法最简单，只要将所有虚拟网域名称都列在 `mydestination` 参数，并像平常一样为每一位用户（不管他们在哪一个网域）都创建自己的系统账户，他们就可以收到写给任何网域的邮件。

这种模式使用 local MDA 来执行投递操作，因此，正式网域该有的功能，虚拟网域同样一应俱全，所有网域共享同一个别名文件，每位用户都可以创建自己的 *.forward* 文件。举例来说，假设我们希望 *oreillynet.com* 网域的邮件服务器兼任 *ora.com* 与 *oreilly.com* 这两个网域的邮件交换站（也就是说，正式网域为 *oreillynet.com*，而 *ora.com* 与 *oreilly.com* 都是虚拟网域）。第一步是设定那两个虚拟网域的 DNS，将它们的 MX 都指向此服务器；第二步是修改此服务器的 `mydestination` 参数，如下：

```
# mydomain 是正式网域
mydomain = oreillynet.com
# ora.com 与 oreilly.com 是虚拟网域
mydestination = $myhostname, $mydomain, ora.com, oreilly.com
```

完成修改之后，记得要让 Postfix 重新读取其配置文件，让你所做的改变生效：

```
# postfix reload
```

用户现在可以收到寄给 `mydestination` 所列的任一网域的邮件。外界写给 *info@ora.com* 或 *info@oreilly.com* 的邮件，都会被投递到同一个系统账户的邮箱。

独立网域搭配系统账户

如果每个虚拟网域都要有自己的命名空间，设定方法就稍微复杂些。在独立网域模式下，写给 *info@ora.com* 的邮件不应该被放在 *info@oreilly.com* 的邮箱中。在这种情况下，虚拟网域不应该被列在 `mydestination` 参数中，而应该被列在 `virtual_alias_domains` 参数中：

```
virtual_alias_domains = ora.com, oreilly.com
```

你必须为每一个邮件地址分别创建对应的系统账户。每一个系统账户的名称都必须独一无二，但是不要求一定要与邮件地址的人名部分相同，因为我们可以另外准备一个查询表来定义两者之间的对应关系（这个查询表称为“虚拟别名表”）。若要区别账户所属网域，同时又要避免账户名称重复，最简单的方法是将网域名称当成账户名称的一部分。一种可行的命名法则，是使用类似 *info.ora.com* 和 *info.oreilly.com* 这样的名称来创建系统账户。不过，某些系统平台的账户名称被限制在 8个字符以下，如果你的平台不支持长名称，就不能使用这种命名法则。

在Postfix知道它应该收下哪些网域的邮件，而你也为每一个地址都准备好对应的系统账户之后，下一步是将 virtual_alias_maps 指向虚拟别名表：

```
virtual_alias_maps = hash:/etc/postfix/virtual_alias
```

此查询表定义了每一个虚拟网域邮件地址所对应的实际邮件地址（系统账户或非本地地址）：

```
info@ora.com          helene@localhost
info@oreilly.com      frank@localhost
kdent@oreilly.com     kyle.dent@onlamp.com
```

每当你修改虚拟别名表后，别忘了使用 postmap将该文件转换成Postfix 可直接查询的格式：

```
# postmap virtual_alias
```

如果 *helene* 和 *frank* 能够登录服务器系统，直接从服务器寄信出去，你可能还需要另外准备一个规范映射表，好让他们寄出去的邮件的寄件人地址被改写成正确的地址。例如：

```
canonical_maps = hash:/etc/postfix/canonical
```

canonical 文件的内容应该类似下面这样：

```
helene      info@ora.com
frank       info@oreilly.com
```

再次提醒，规范映射表也必须经过 postmap的转换：

```
# postmap canonical
```

独立网域搭配虚拟账户

前两种模式最大的缺点，就是每一个邮件地址都必须在服务器上有对应的系统账户。虽然事前的设定过程不难，但是事后的维护工作却很麻烦。尤其当虚拟网域的数量增加时，

系统账户的维护工作更是苦不堪言。此外，从系统安全的角度来看，如果绝大部分用户都只是利用服务器来收信而已，实在没有必要赋予他们能够登录系统的能力。比较务实的做法，应该是要求 Postfix 将邮件投递到本地系统上的一个虚拟邮箱目录，在此目录下，每一个虚拟邮件地址都有自己的邮箱文件，而用户可通过支持虚拟邮箱的 POP/IMAP server 来取信。虚拟邮箱与一般邮箱的差别之处，在于邮件文件与系统账户之间不要求存在一比一的对应关系。

使用虚拟邮箱模式时，你必须在 virtual_mailbox_domains参数列出每一个虚拟网域的名称：

```
virtual_mailbox_domains = ora.com, oreilly.com
```

如果你有许多虚拟网域，不妨将它们列在一个文件，然后将 virtual_mailbox_domains 指向该文件：

```
virtual_mailbox_domains = /etc/postfix/virtual_domains
```

/etc/postfix/virtual_domains 文件的每一行，各包含一个虚拟网域名称：

```
#
# /etc/postfix/virtual_domains
#
ora.com
oreilly.com
```

每当 Postfix 收到写给 virtual_mailbox_domains 所列的虚拟网域的邮件时，就交由 virtual MDA 执行投递操作。virtual 的实质是 local MDA 的“小型版本”，它以一种安全而且高效率的程序来进行投递，其代价是牺牲了本地别名文件、*.forward* 文件、MLM（mailing list manager）的支持。不过，这不是什么大缺点，因为我们仍可使用 virtual_alias_maps 参数（参阅“独立网域搭配系统账户”）来达到同样的效果，而本章稍后也会介绍另一种能将邮件传给外部程序的技术。

设定虚拟邮箱时，你应该妥善安排邮件目录的结构，使其符合 POP/IMAP server 的预期要求。支持虚拟邮箱的 POP/IMAP server，多半假设所有虚拟邮箱都位于某一个基础目录下（比方说 */var/vmail/*），在此目录下，每一个虚拟网域都有一个与其网域名称同名的子目录。因此，如果有两个虚拟网域，则它们的虚拟邮箱应该分别被放在两个子目录下：

```
/var/vmail/ora.com/
/var/vmail/oreilly.com/
```

定义虚拟邮箱基础目录的参数是virtual_mailbox_base，你必须给出完整的目录路径：

```
virtual_mailbox_base = /var/vmail
```

virtual_mailbox_maps参数定义虚拟邮箱查询表的位置：

```
virtual_mailbox_maps = hash:/etc/postfix/virtual
```

此查询表描述每一位虚拟邮箱用户的邮件地址以及对应的邮箱文件的相对路径文件名（相对于 virtual_mailbox_base 定义的基础目录）。邮箱文件的格式可以是 mbox 或 maildir （参阅第七章）。如果你选择 maildir 格式，邮箱文件名末端必须加一个“/”符号。以下是虚拟邮箱对应表的典型内容：

```
info@ora.com          ora.com/info
info@oreilly.com      oreilly.com/info
```

完成上述设定后，外界写到 *info@ora.com* 和 *info@oreilly.com* 的邮件，就不会存放在同一个邮箱了。

邮箱文件的拥有权

Unix系统上的每个文件都有主人，虚拟邮箱文件也不例外 ——虽然虚拟邮箱的“用户”没有系统账户。因此，用户如何访问邮箱文件，决定了邮箱文件拥有权的归属。通常，POP/IMAP server 的运行权限继承于某个专属的系统虚账户，并假设其权限足以访问所有邮箱文件。因此，邮箱文件的拥有者应该是 POP/IMAP server所用的那个虚账户。但是，如果必要的话，Postfix 也容许你自己决定邮箱文件的拥有者。你可以让每个邮箱文件都有自己的拥有者或是让同网域的所有邮箱文件都同属于一位拥有者，但不同网域的邮箱分属不同的拥有者。

virtual_uid_maps 与 virtual_gid_maps 参数决定了 Postfix 的 virtual MDA 投递邮件到虚拟邮箱时，应该继承的系统账户。假设所有虚拟邮箱都属于同一个账户，则你可以使用 static映射方式，直接指定 virtual所要继承的 UID与 GID：

```
virtual_uid_maps = static:1003
virtual_gid_maps = static:1005
```

这会使得 virtual MDA 以 UID 1003、GID 1005 的身份来访问任何邮箱文件。如果你希望 virtual 以不同的 UID 身份来访问不同的邮箱文件，你必须另外准备一个查询表，定义各个虚拟邮箱地址与 UID之间的对应关系，然后将 virtual_uid_maps指向此查询表：

```
virtual_uid_maps = hash:/etc/postfix/virtual_uids
```

如果大部分的虚拟邮箱文件属于固定的拥有者，但也有某些是属于不同的 UID ,你可以混合使用 static与查询表：

```
virtual_uid_maps = hash:/etc/postfix/virtual_uids static:1003
```

virtual_gid_maps参数的用法与 virtual_uid_maps 完全相同，不再赘述。

定义邮箱文件与拥有者对应关系的 */etc/postfix/virtual_uids* 文件，其索引键是邮箱文件所属的邮件地址，而对应值是邮箱文件拥有者的 UID（或 GID）。举例来说，如果我们希望 *ora.com* 网域的所有邮箱都属于同一个 UID，而 *oreilly.com* 的邮箱属于另一个 UID，则 virtual_uids 的内容应该类似这样：

```
#
# /etc/postfix/virtual_uids
#
info@ora.com          1004
kdent@ora.com         1004
info@oreilly.com      1007
service@oreilly.com   1007
```

虚拟别名

即使是虚拟网域，Postfix也能将其邮件投递到本地邮箱，或是转寄到其他站点。既然所有类型的收件地址都会被检查是否为虚拟别名，所以，只要把转寄地址放在 virtual_alias_maps所指的文件（而非 virtual_mailbox_maps所指的文件）即可。请确定 virtual_alias_maps参数指向一个虚拟别名表：

```
virtual_alias_maps = hash:/etc/postfix/virtual_alias
```

*/etc/postfix/virtual_alias*文件包含了要被转寄到他处的虚拟地址：

```
kdent@oreilly.com    kyle.dent@onlamp.com
```

不要将同一个网域同时列在 virtual_mailbox_domains和virtual_alias_domains 中。对于同时含有邮箱与别名的虚拟网域，请使用 virtual_mailbox_domains。如果所有地址都是别名，请使用 virtual_alias_domains。

无限别名地址

无论是虚拟别名还是虚拟邮箱，它们的查询表都可以含有一个“无限别名地址”（catchall address）—— 只有网域部分没有人名部分的邮件地址（例如@ *example.com*）。无限别名地址所对应的邮箱（如果是虚拟邮箱的话）或转寄地址（如果是虚拟别名的话），用来承接外界寄到该网域的不明用户的邮件。无限别名地址应该谨慎使用，因为它们一定会收到大量垃圾邮件。发送垃圾邮件者常用“字典攻击法”来滥发垃圾邮件，所以你的邮件服务器会时常收到寄信给不明用户的要求。如果设定了无限别名地址，等于是让这些发送垃圾邮件者有机会塞爆你的邮箱。

无限别名虚拟邮箱

第一步是找出一个邮箱来承接所有寄给不明用户的邮件。你可以使用现有的邮箱，或是另外创建一个新邮箱。决定好邮箱之后，在 virtual_mailbox_maps所指的查询表增加一笔无限别名地址记录：

```
@ora.com          ora.com/service
```

无限别名虚拟别名

无限别名虚拟别名的设定步骤，很类似无限别名虚拟邮箱。第一步，先选定一个邮件地址来接收所有寄到不明别名的邮件，然后在virtual_alias_maps所指的查询表定义无限别名所对应的地址：

```
@ora.com          customer.service@onlamp.com
```

除非你的系统没有任何虚拟邮箱，否则不应该设置无限别名虚拟别名。因为Postfix 会先展开虚拟别名，然后才检查虚拟邮箱；如果你设置了无限别名虚拟别名，它将拦截掉所有的邮件，包括原本应该投递到虚拟邮箱的邮件。

在设置了无限别名虚拟别名的情况下，如果还希望虚拟邮箱能够收到邮件，唯一办法是将虚拟别名展开成虚拟邮箱地址。举例来说，假设下面是虚拟邮箱查询表的内容：

```
info@ora.com          ora.com/info
info@oreilly.com      oreilly.com/info
```

那么，含有无限别名的虚拟别名表，应该要含有上述虚拟邮箱的别名地址：

```
info@ora.com          info@ora.com
info@oreilly.com      info@oreilly.com
kdent@oreilly.com     kyle.dent@onlamp.com
@ora.com              customer.service@onlamp.com
```

如此一来，寄到 *info@oreilly.com* 的邮件，就不至于被 *@ora.com* 无限别名拦截掉了。

虚拟网域搭配特殊格式的邮箱

我们要讨论的最后一种操作模式，是在一个使用特殊邮箱格式的系统上架设虚拟网域。在这类系统上，Postfix 本身不做投递工作，而是使用 LMTP协议将邮件托付给知道如何访问特殊邮箱的程序（即 LMTP server）。

因为 Postfix必须先收下邮件然后才转交给 LMTP server ,所以 Postfix 必须知道它要收下哪些虚拟网域名称的邮件。请将这些网域的名称列于 virtual_mailbox_domains 参数：

```
virtual_mailbox_domains = ora.com, oreilly.com
```

你还必须逐一列出每一个邮件地址，这样 Postfix才能够知道哪些收件地址是有效的，并拒收不明用户的邮件。请将 `virtual_mailbox_maps` 参数指向有效地址查询表：

```
virtual_mailbox_maps = hash:/etc/postfix/virtual
```

因为 Postfix 只需知道地址的有效性而不参与投递工作，所以 */etc/postfix/virtual* 查询表只有索引键的部分有意义，对应值的部分没有作用。然而，查询表的格式不容许我们省略对应值，所以仍必须随便填入数据：

```
info@ora.com          General Information Address
info@oreilly.com      General Information Address
```

为了让 Postfix 能将虚拟网域的邮件交给 POP/IMAP server ，我们还必须在 *main.cf* 的 `virtual_transport` 参数中指定正确的传送方法，也就是 LMTP server 的 socket 是何种形式。假设 LMTP server 与 Postfix 都位于同一台机器，而且 LMTP server 使用 Unix-domain socket ，其文件位置是 */var/imap/socket/lmtp*，则 `virtual_transport` 应该这样设定：

```
virtual_transport = lmtp:unix:/var/imap/socket/lmtp
```

现在，每当 Postfix 收到寄给 `virtual_mailbox_domains` 所列的任一网域的邮件，都会通过 LMTP 协议交给同机器上的 POP/IMAP server 来执行投递操作。

投递到外部程序

先前说过，virtual MDA 不能处理系统别名文件与个人的 *.forward* 文件。虽然使用 `virtual_alias_maps` 可弥补虚拟网域的地址别名问题，但是 *.forward* 文件的一项重要功能 —— 传递邮件内容给外部程序，还没解决。这表示虚拟网域的地址没有自动回信的功能，也不能架设邮递列表管理系统（Mailing-List Manager，简称 MLM）。本节将示范如何利用 Postfix的其他机制，让寄到虚拟地址的邮件也可以传给外部程序。第一个例子示范如何投递邮件到一个自动回信程序，第二个例子以 Majordomo MLM 来做示范。

自动回信程序的作用，主要是在没有任何人员介入操作的情况下，自动处理外来邮件、存储或处理来信信息，然后回信给原寄件人。回信程序可以是简单的脚本，也可以是以 C 或任何语言写成的复杂程序。

假设我们要提供一个供客户请求信息的服务，只要客户寄信到特定地址，就回复一封含有请求信息的回信。我们不希望使用人工处理这种千篇一律的机械工作，所以，最好的

办法是写一个自动回信程序。例 8-1是我们写出来的 inforeply.pl 自动回信程序，它是一个 Perl script。为了简单起见，我们并没有写得很周详，还不足以当成通用的回信程序（例如 Unix 系统上常见的 vacation），因为它不会记录哪些寄件人已经来信请求过信息了，也没检查收件地址是否确实为我们指定的自动回复地址（参阅后文“自动回信程序的设计”）。不过，你可以自己加强此程序的功能，比方说，检查来信的主题或正文里的关键字，据此回复不同类型的信息。

例 8-1：简单的自动回信程序

```
#!/usr/bin/perl -w
#
# inforeply.pl - Automatic E-mail reply.
#
# 所有信息都记录在邮件日志文件（通常是 /var/log/mailllog)。
# 运行本程序之后，可从日志文件看到结果。
#
# 使用本程序之前，你可能需要调整某些变量的值，
# 以下是各项重要的变量的说明：
#
# $UID 是本程序的运行标识。
#  正确值应该是 master.cf 中调用本程序的那一行，
#    其 user= 属性所设的 UID。
#  如果想在命令行测试本程序,则 $ UID 应该是你的 UID。
#
# $ENV_FROM是回信信封上的 FROM地址。
#  默认值为空白，这表示使用空地址（<>)
#  你可以将它设定成一个专用来接收退信的地址。
#  注意：不要使用触发本程序的那个地址，
#    否则会造成邮件循环。
#
# $INFOFILE 是含有回信内容的文本文件。
#  此文件应该包含完整的回信，
#    包括 Subject: 与 From:等必要的标题。
#    唯一例外是 To:字段，本程序使用来信者的地址来自动设定此栏。
#  注意：标题与正文之间至少要隔一空白行。
#
# $MAILBIN是 sendmail 程序文件的完整路径。
#  如果你的 sendmail不是安装在／usr/sbin/sendmail
#    请依照实际情况修改。
#
# @MAILOPTS 是一个包含 sendmail 命令行选项的数组。
#  默认值应该足以应付大部分情况。
#
# 本程序调用了 syslog，所以你的 Perl 环境必须先安装 Sys::Syslog 模块。
#

require 5.004;  # 因为需要 Sys::Syslog模块的 setlogsock
use strict;
use Sys::Syslog qw(:DEFAULT setlogsock)

#
# 配置变量。请参考前述注释来设定适当值。
```

```
#
my $UID = 500;
my $ENV_FROM = "";
my $INFOFILE = "/home/autoresp/inforeply.txt";
my $MAILBIN = "/usr/sbin/sendmail";
my @MAILOPTS = ("-oi", "-tr", "$ENV_FROM");
my $SELF = "inforeply.pl";
#
# 以下是主程序
#

my $EX_TEMPFAIL = 75;
my $EX_UNAVAILABLE = 69;
my $EX_OK = 0;
my $sender;
my $euid = $>;

$SIG{PIPE} = \&PipeHandler;
$ENV{PATH} = "/bin:/usr/bin:/sbin:/usr/sbin";

setlogsock('unix');
openlog($SELF, 'ndelay,pid', 'user');

#
# 检查我们的环境
#
if ( $euid != $UID ) {
        syslog('mail|err',"error: invalid uid: $> (expecting: $UID)");
        exit($EX_TEMPFAIL);
}
if ( @ARGV != 1 ) {
        syslog('mail|err',"error: invalid invocation (expecting 1 argument)");
        exit($EX_TEMPFAIL);
} else {
        $sender = $ARGV[0];
        if ( $sender =~ /([\w\-.%]+\@[\w.-]+)/ ) {    # scrub address
                $sender = $1;
        } else {
                syslog('mail|err',"error: Illegal sender address");
                exit($EX_UNAVAILABLE);
        }
}
if (! -x $MAILBIN ) {
        syslog('mail|err', "error: $MAILBIN not found or not executable");
        exit($EX_TEMPFAIL);
}
if (! -f $INFOFILE ) {
        syslog('mail|err', "error: $INFOFILE not found");
        exit($EX_TEMPFAIL);
}

#
# 检查异常寄件人
#
```

```
if ($sender eq ""
    || $sender =~ /^owner-|-(request|owner)\@|^(mailer-daemon|postmaster)\@/i)
{
    exit($EX_OK);
}

#
# 检查来信标头里的 Precedence字段
#
while ( <STDIN> ) {
        last if (/~$/);

exit($EX_OK) if (/^precedence:\s+(bulk|list|junk)/i);
}

#
# 开启邮件文件
#
if (! open(INFO, "<$INFOFILE") ) {
        syslog('mail|err',"error: can't open $INFOFILE: %m");
        exit($EX_TEMPFAIL);
}

#
# 将本程序的输出接到寄信程序的输入
#
my $pid = open(MAIL, "|-") || exec("$MAILBIN", @MAILOPTS);

#
# 送出回信
#
print MAIL "To: $sender\n";
print MAIL while (<INFO>);

if (! close(MAIL) ) {
        syslog('mail|err',"error: failure invoking $MAILBIN: %m");
        exit($EX_UNAVAILABLE);
}

close(INFO);
syslog('mail|info',"sent reply to $sender");
exit($EX_OK);

sub PipeHandler {
        syslog('mail|err',"error: broken pipe to mailer");
}
```

将虚拟网域的邮件传给自动回信程序

虚拟网域的邮件不能通过 *.forward*机制传给外部程序，因此，若想让自动回信程序能收到虚拟地址的邮件，我们必须在 *master.cf*中增加一个能投递邮件到外部程序的特殊 MDA，然后使用传输表将特定地址的邮件交给这个特殊的 MDA 处理。

许多自动回信程序一次只能处理一封来信，而且一次只能回一封信。对于每一种MDA，都有一个限制收件人数量的*mda*_destination_recipient_limit参数，其中的*mda*是MDA在*master.cf*里的服务名称（第一栏）。以例8-1的自动回信程序为例，假设它的服务名称为inforeply，而它一次只能回复一封邮件，则我们必须在*main.cf*设定下列参数：

```
inforeply_destination_recipient_limit =
```

自动回信程序的设计

如果你想设计自己的自动回信程序，有些重要考虑事项必须注意。首先，同时也可能是最重要的事项，就是程序的数据来源是网络——一种不可信赖的数据源。千万别假设你要处理的数据一定会是什么样子，除了它们被刻意安排成能征服你的系统之外。无论任何情况，都不应该调用shell来处理不可信的外来数据，让外界有机会夺取你的系统的访问权。

另一个要考虑的事项，是礼节教养问题。想象一下，如果自动回信程序的回信对象是一个含有几千人的邮件列表（mailling list），将客户请求的（私密）信息泄露给几千人，或是让几千人收到不相干的邮件，应该都不是你愿意见到的事。因此，在实际回信之前，最好先检查寄件人地址是否为owner-*list*或*list*-request之类的形式。此外，还有一些地址恐怕也是你不想回复的，像postmaster、daemon、majordomo等。你的程序应该将自己的信封地址设定成空字符串，以免造成邮件循环。

许多邮件列表会利用Precedence：标题栏来注明它自己。它们通常使用bulk之类的值来表示邮件的性质。你的回信程序应该要检查Precedence：标题栏的值，如果发现bulk、list或junk之类的字样，就不必回信了。

最后，确定你的程序能记录每封来信的处理状况。一旦Postfix将邮件托付给你的程序，你的程序就有检查错误的责任，并为管理员提供可以在事后追查问题的线索。

假设邮件服务器的规范网域是*example.com*，而*ora.com*是架设在此服务器上的一个虚拟网域，*info@ora.com*是为客户请求信息提供的服务邮箱，而我们希望inforeply.pl能自动回复客户寄到*info@ora.com*的邮件。以下是我们的设定步骤：

1. 创建一个可供inforeply.pl程序使用的虚账户，并确定此账户有足够权限能访问相关文件。
2. 在*master.cf*为inforeply.pl设置一个新的投递服务，称为inforeply。可能的设定方式如下：

```
inforeply     unix  -       n       n       -       -       pipe
  flags= user=autoresp argv=/usr/local/bin/inforeply.pl ${sender}
```

Postfix的 pipe daemon专用来将邮件内容传给外部程序，让外部程序可从自己的stdin得到邮件的完整内容。 flags属性用于提供特殊选项给外部程序（如果需要完整的选项列表，请参阅 pipe(8)）。user属性是外部程序的标识，任何以pipe为传输渠道的投递服务，都必须提供 user 属性。argv 属性是外部程序的完整路径以及 Postfix 提供给该程序的命令行自变量。 Postfix 提供了一系列特殊宏，让你能够传递特定的信息给外部程序。例如，${sender}宏代表信封上的寄件人地址，${recipient}宏是收件人地址。关于宏的完整解释，请参阅 pipe(8)。

3. 如果你的系统还没设置任何传输机制，请将 *main.cf* 的 transport_maps 参数指向一个传输表：

```
transport_maps = hash:/etc/postfix/transport
```

4. 传输表决定“收件地址”与“投递服务”的对应关系。在此例中，我们使用的收件地址是 *autoresp@ora.com*：

```
autoresp@ora.com              inforeply
```

现在，所有寄到 *autoresp@ora.com* 的邮件，都会交给自动回信程序来处理。

5. 使用 postmap 将传输表转换成数据库格式：

```
# postmap /etc/postfix/transport
```

6. 将 virtual_alias_maps指向你的虚拟别名查询表：

```
virtual_alias_maps = hash:/etc/postfix/virtual_alias
```

7. 在虚拟别名查询表中定义服务地址（*info@ora.com*）与收件地址(autoreply)的对应关系。为了让客服人员也能收到来信副本，我们让 *info@ora.com*对应到两个收件地址：

```
info@ora.com          autoresp@ora.com, service@oreilly.com
```

8. 运行 postmap 将虚拟别名表转换成数据库：

```
# postmap /etc/postfix/virtual_alias
```

现在，所有寄到 *info@ora.com* 的邮件，都会被投递到 *autoresp@ora.com* 与 *service@oreilly.com*。

9. 要求 Postfix重新读取配置文件，使我们在 *main.cf*与 *master.cf*所做的改变生效。

```
# postfix reload
```

每当外界写一封邮件寄到 *info@ora.com*，Postfix先检查 virtual_alias 虚拟别名表，发现该地址应该展开成 *autoresp@ora.com* 与 *service@oreilly.com* 。接着， Postfix发现 *autoresp@ora.com*被列在transport传输表，所以使用 *master.cf*所设定的 inforeply 服

务来执行投递操作。而 inforeply 使用 pipe 将邮件内容传给外部的 inforeply.pl 程序，由它回信给原寄件人。最后，这封来信还会被重新提交，寄给 *service@oreilly.com*。

在虚拟网域架设邮件列表管理系统

第二个例子，我们要示范如何在虚拟网域上设置一个邮件列表（Mailling List）。MLM是第十章的内容，如果读者没用过 MLM，你可能需要先按照第十章描述的步骤，在你自己的系统上先安装好 Majordomo。

Postfix 与 MLM之间的连接是通过别名文件：利用别名文件的：include:机制来展开列表，并将管理用途的别名对应到实际地址与 MLM管理程序。然而，只有 local MDA才会处理系统的别名文件，vitrual MDA没有这方面的功能。

虚拟邮件列表的原理，是在本地网域下创建列表的相同版本，这个本地版本仅供系统内部使用，外界用户所用的仍是虚拟地址，他们并不知道本地版本的存在。在决定本地版本的名称时，我们可设法将虚拟名称包含在内，以便容纳同系统上的其他虚拟网域列表。以下在 *astronomy@ora.com* 虚拟地址创建一个邮件列表，其本地版本为 *astronomy-ora@example.com*。

1. 按照正常的邮件列表设置程序（参阅第十章），创建本地版本的邮件列表。将下列内容加入 */usr/local/majordomo/aliases*：

```
# astronomy@ora.com list
astronomy-ora:     :include:/usr/local/majordomo/lists/astronomy
owner-astronomy-ora:     kdent@ora.com
astronomy-ora-request: "|/usr/local/majordomo/wrapper \
       request-answer  astronomy-ora"
astronomy-ora-approval:      kdent@ora.com
```

2. 重建 Majordomo 别名表：

```
# postaliases /usr/local/majordomo/aliases
```

3. 创建一个列表文件，将订阅者的邮件地址记录在此文件里。并将文件的拥有者设定为 majordom账户：

```
# touch /usr/local/majordomo/lists/astronomy
# chown majordom /usr/local/majordomo/lists/astronomy
```

4. 如果必要，在*/usr/local/majordomo/lists/astronomy-ora.info*为该列表创建一个 *info* 文件。

5. 在虚拟网域创建管理列表所需的必要地址。将下列内容加入 *virtual_alias* 虚拟别名表：

```
# astronomy@ora.com list
astronomy@ora.com              astronomy-ora@localhost
```

```
owner-astronomy@ora.com        owner-astronomy-ora@localhost
astronomy-request@ora.com      astronomy-ora-request@localhost
astronomy-approval@ora.com     astronomy-ora-approval@localhost
```

6. **重建虚拟别名表：**

```
postmap virtual_alias
```

7. **将地址加入** */usr/local/majordomo/lists/astronomy* **列表文件。**

现在，你只要寄信到 *astronomy@ora.com*，**列表文件内的所有地址都会收到信。**

第九章

邮件转发

到目前为止，我们主要的讨论焦点，都是 Postfix于邮递路径末端所扮演的角色。也就是说，抵达 Postfix服务器的邮件，主要是被投递到本机系统。但是， Postfix 也常扮演另一种角色 —— 位于邮递路径中的中继器。本章将讨论 Postfix 在扮演 MTA-to-MTA 通信客户端时，提供了哪些配置选项。

备用交换器

在 DNS 的术语中，MX 代表 Mail eXchanger （邮件交换器，见第六章）。一个 MX记录代表了一个网域承接外来邮件的主机，以及该主机的优先度。同一个网域可以同时有好几个 MX 记录，优先度最高（参考值最低者）的主机称为主交换器（ Primary MX），其余主机称为备用交换器（ Backup MX)。图 9-1 描绘了备用交换器扮演的角色。如你所见，备用交换器的任务，是在主交换器离线时，承接该网域的外来邮件；当优先度高于自己的其他备用交换器或主交换器或恢复上线时，已收下的邮件必须传给主交换器来处理，备用交换器不能自己擅作主张，将收到的邮件直接投递到最终目的地。

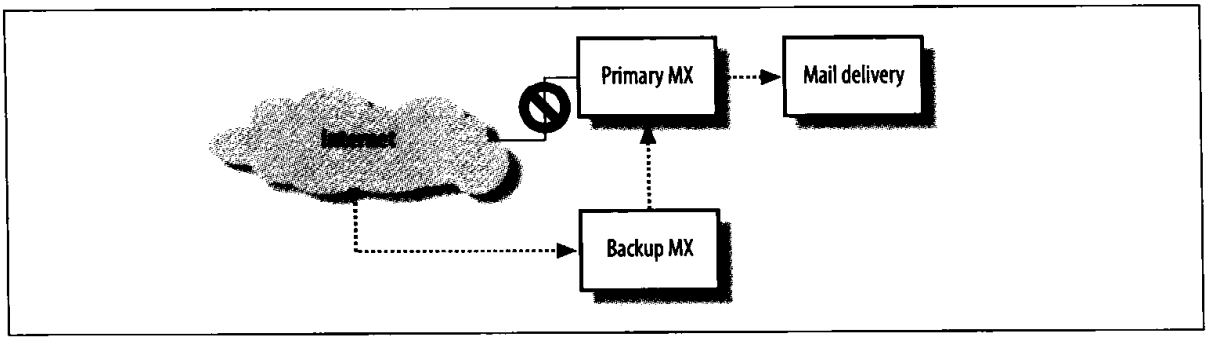

图 9-1 ：备用交换器接替主交换器承接外来邮件

架设备用交换器时，不必特别设定如何传信给主交换器，因为 Postfix 自己能从 DNS 查出如何转信给主交换器。不过，前提是你必须让 Postfix知道自己是哪一个网域的备用交换器。你的 Postfix 系统是哪些网域的备用交换器，那些网域的名称就必须被列在 relay_domains 参数中。当寄信方的 MTA发觉收信网域的主交换器离线了，它会试着联系优先度次高的交换器，直到有一个备用交换器能够收下邮件为止。如果你的Postfix 系统是某个网域的备用交换器，而且该网域被列在 relay_domains 参数中，则 Postfix 会收下该网域的邮件，并将其排入队列。每隔一段时间， Postfix 会扫描它的队列，并检查是否有优先度更高的邮件交换器恢复连接，如果有，则交出先前代收的邮件。

如果不能交出邮件， Postfix 会持续尝试，直到邮件在队列里等待的时间超过 maximal_queue_lifetime 所指定的时限为止。此参数的默认值为五天，如果邮件在等待队列里待的时间超过此上限， Postfix会发出退信通知函给原寄件者。如果你事先知道主交换器停机的时间会超过五天，可以适度提高时限。

转发列表

Postfix可从 relay_domains参数与 DNS系统知道自己的备用交换器角色，但是它如何知道转发的网域中有哪些有效收件人？如果备用交换器无法事先知道主交换器有哪些合法邮箱，它势必被迫盲目收下所有邮件，等到主交换器恢复连接时，才会发现有哪些邮件无法投递并退信给原寄件人。因此，我们郑重建议你在备用交换器设置一份列表，记录受理网域有哪些合法用户，并定期与主交换器同步更新。

当备用交换器上的 Postfix server 收到寄给 relay_domains所列网域的邮件时，它会检查 relay_recipient_maps 参数所指定的查询表，借此判断是否应该收下邮件，如不应收下则直接拒收。relay_recipient_maps参数的设定方式如下：

```
relay_recipient_maps = hash:/etc/postfix/relay_recipients
```

relay_recipients 查询表应该记录所有合法收件人的邮件地址。 Postfix只需要此表的索引键，所以对应值可以任意填写（但不可不填）：

```
#
# relay_recipients
#
user1@example.com          any_value
user2@example.com          any_value
user3@example.com          any_value
```

对于人事变动频繁的网域，我们建议你应该研制一个同步程序，利用 rsync、ssh、crond 等工具，自动从主交换器下载最新列表。

如果备用交换器的Postfix不过滤转发网域的收件地址，会有什么后果？答案是备用交换器会收到一大堆无法投递的垃圾邮件，并产生一大堆寄不出去的退信通知函。因为捏造收件人是垃圾邮件发送者常用的手段之一，他们事先不知道你的邮件交换器有哪些用户，于是以常用的人名来捏造收件地址，试图蒙混过关。此外，垃圾邮件也不可能会提供有效的回信地址（就算是有效地址，通常也是无辜受害人的地址，而非垃圾邮件发送者自己的地址）。关于垃圾邮件的问题，留待第十一章讨论。

假如备用交换器与主交换器位于同一个局域网络，除了使用 rsync、scp、ssh、crond等传统工具之外，还有更好的方法来进行列表的同步更新工作。比方说，将用户账号储存在某种数据库中，像 MySQL、LDAP等，让 Postfix 能够进行实时查询。这方面的技术留待第十五章讨论。

在你设定了 relay_recipient_maps之后，还必须面对一个潜在的问题：你必须将所有转发网域的所有合法邮件地址都列入查询表，因为Postfix将拒收任何收件地址没出现在查询表的邮件。如果你不知道某些网域的合法邮件地址，你可以为该网域设置一个无限别名地址（catchall address）：

```
#
# relay_recipients
#
user1@example.com       any_value
user2@example.com       any_value
user3@example.com       any_value
@oreillynet.com         This_is_CATCHALL_ADDRESS
```

这个例子的最后一行，会使得Postfix盲目收下任何收件地址的网域部分为*@oreillynet.com*的邮件——包括垃圾邮件。很显然地，这只是临时性的应急之举，你应该尽快地列出该网域的转发列表，否则就准备接受垃圾邮件的轰炸吧。

快速清空

为许多网域承接邮件的网络（例如 ISP网络），时常会面临无法立刻递出邮件的困境，因为他们的顾客的服务器不见得永远保持连接（否则也用不着请 ISP代收邮件了）。在客户离线的情况下，ISP只能将收到的邮件暂时存放在队列里，等到客户的服务器恢复连接时，再使用 SMTP的 ETRN 命令，一次清空队列里的所有邮件：

```
220 mail.ora.com ESMTP Postfix
EHLO mail.example.com
250-auger.seaglass.com
250-PIPELINING
250-SIZE 10240000
250-VRFY
250-ETRN
```

```
250-STARTTLS
250 8BITMIME
ETRN example.com
250 Queuing started
```

当一个网域的主交换器恢复连接并完成收信准备时，我们的队列里已囤积了大量要传给该网域的邮件，让 Postfix 逐一检查每个队列文件的收信网域为何，将会损耗许多时间。为此，Postfix 提供了一种称为“快速清空”（fast flush）的功能，可用来在队列中找出寄给特定网域的所有邮件。快速清空功能是由 flush daemon 控制管理的，对于 Postfix 所代理的每一个收信网域，flush 各准备了一份列表，记录该网域邮件在队列里的编号。如此，在发出 ETRN 命令之后，Postfix 可以迅速找出所有要投递到该网域的邮件。

默认情况下，flush 只管理 relay_domains 所列的网域。如果还有其他网域也需要快速清空服务，你可以将它们的网域名称列在 fast_flush_domains 参数中，像这样：

```
fast_flush_domains = $relay_domains, example.com
```

此例中，*example.com* 是一个没列在 relay_domains 的网域。

你可以使用 postqueue -s 命令（或是等效的 sendmail -qR 命令）来通知 Postfix，某个快速清空网域已经准备好接收先前累积的邮件了：

```
$ postqueue -s example.com
```

传输表

当你想改变默认的邮递流程时，可利用传输表（transport map）来达成愿望。也就是说，如果你希望 Postfix 以你指定的方式来处理特定网域的邮件，而不管 DNS MX 记录是怎么设定的，你可将相关网域与传输方式写在一个传输表中，然后将 transport_maps 参数指向此传输表。

本节讨论 transport_maps 参数的基本用法，后面几章会陆续谈到此参数在其他方面的应用。transport_maps 参数可指向一个或多个传输表，例如：

```
transport_maps = hash:/etc/postfix/transport
```

传输表的索引键可以是完整的邮件地址（译注 1）、网域名称或子网域名称。当收件地址或网域名称符合传输表某个记录的索引键时，则以该记录的对应值所指定的传输法来投递该封邮件。

译注 1：　Postfix 2.0 版之前，传输表的索引键不可以是邮件地址。

例 9-1：传输表内容

```
example.com          smtp:[192.168.23.56]:20025
oreilly.com          relay:[gateway.oreilly.com]
oreillynet.com       smtp
ora.com              maildrop
kdent@ora.com        error:no mail accepted for kdent
```

传输表对应值的格式，随传输方法而异，但大体上符合 *transport:nexthop* 这样的格式。某些传输方法的 *nexthop* 可表示成 *host:port* 形式，表示递送路径下一站的主机名与通信端口。以下分别说明可组成对应值的三种元素：

transport

传输方法的名称。此名称必须是 *master.cf* 所定义的传输类型之一。如果你增加了新的传输方法，则必须先在 *master.cf* 定义其名称与传输类型。

host

收信主机或网域。host只能搭配 `inet` 传输类型（SMTP 或 LMTP）。Postfix按照一般收信网域的处理流程来处理 `host` 先查询 MX记录来决定邮件的去处，如果没有 MX记录，则传到 A记录所指的 IP地址。如果将主机名称放在一对方括号内（例如：`[gateway.oreilly.com]` ），则 Postfix会直接传信到 host的 A记录所指的IP地址。但如果你直接使用 IP地址，则一定要加方括号，例如 `[192.168.23.56]`

port

收信主机的通信端口。通常只有 `inet` 类型的传输服务（例如 SMTP与 LMTP）才会指定通信端口。port的格式可以是十进制数，也可以是 */etc/services* 文件定义的服务名称。

例 9-1 列出的传输表内容，展示了 *transport:host:port* 的多种可能组合，分别解释如下：

example.com smtp:[192.168.23.56]:20025

收下所有写给 *example.com* 的邮件，然后使用 `smtp` MDA传送到位于192.168.23.56的主机，而且 `smtp` MDA必须连接到该主机的port 20025 ,而非默认的 SMTP port 25。请注意，由于我们直接使用了 IP地址，所以必须加上方括号。

oreilly.com relay:[gateway.oreilly.com]

收下所有要寄到 *oreilly.com* 的邮件，然后使用 `relay` MDA 转寄给 *gateway.oreilly.com* 主机。由于没指定通信端口，所以 `relay` 使用默认的 port 25。由于主机名称被放在方括号内，所以邮件是直接传到 *gateway.oreilly.com* 的A记录所指的IP地址，而非 MX记录所指的 IP地址。

`relay` MDA是 Postfix 2.0 版以后才引进的，它修正了队列调度算法可能引起的潜

在效能瓶颈。当你要将入站邮件送到内部系统时，应该直接通过 relay MDA，以避免这类邮件与出站邮件（要送到 Internet 上许多不同网域的邮件）竞争资源。

oreillynet.com smtp

收下所有目的地为 *oreillynet.com* 网域的邮件，然后交给 smtp MDA 执行投递操作。由于没指定 *host:port*，所以 smtp 依照 *oreillynet.com* 网域的 DNS MX 或 A 记录来决定目的地，并使用 port 25 来联系收信服务器。实际上这个例子实属多余，因为只要将 *oreillynet.com* 列在 relay_hosts 或 relay_domains 参数，就可以达成相同效果。

ora.com maildrop

收下所有写给 *ora.com* 网域的邮件，然后交给 maildrop 处理。maildrop 的运作方式必须被明确定义在 *master.cf* 中。由于 maildrop 不需要 inet socket，所以不必指定 *host:port*。

kdent@ora.com error:No mail accepted for kdent

error 是一种特殊的传输服务，它唯一的作用是当场拒收邮件。冒号之后的字符串是回复给传送方的错误信息。

传输表不一定用来将邮件传递到外界，也可以用于将特定邮件交给本地系统，以便进行特殊处理。比方说，过滤邮件内容（参阅第十四章）、暂时扣留某个网域的所有邮件等。为了展示传输表的用法，下一小节将示范如何暂时扣留特定网域的邮件，直到你愿意放行为止。

延迟投递时间

在某些情况下，你可能会希望 Postfix 先收下邮件，但不立刻投递出去。等到你下达 postqueue -f *domain* 命令之后，或是 Postfix 从支持快速清空功能的网域收到 ETRN SMTP 命令之后，Postfix 才递送出被暂时扣留的邮件。

常见的一种情况是 ISP 为客户的网域代收邮件，但是客户的网络并不总是保持连接。ISP 必须先储存邮件，直到客户的网络恢复连接，并且能收下邮件为止。类似的情况还有客户网络上的用户，应该要能够通过本地的邮件网关来寄信，该网关必须能先暂存外寄邮件，直到网络恢复连接时才投递出去。本小节分别示范能应付上述两种情况的技术。

暂缓转发邮件

在以下的程序中，我们要先创建一种新型的传输法，称为“ ondemand”，然后设定传输表，要求所有寄到 *example.com* 网域的邮件，一律先暂时扣留在队列里。

1. 在 *master.cf* 设置一种新的传输法，并命名为 ondemand。它的配置应该与 smtp 传输法一样（除了名称之外）：

```
ondemand      unix  -    -    n    -    -    smtp
```

2. 要求 Postfix 自动延缓任何通过 ondemand 递送的邮件。只要将新设的 ondemand 传输法列在 *main.cf* 的 defer_transports 参数中，就可以达到自动延缓的效果：

```
defer_transports = ondemand
```

3. 确定 transport_maps 参数指向我们的传输表：

```
transport_maps = hash:/etc/postfix/transport
```

4. 编辑修改传输表，使得 *example.com* 网域的所有邮件都通过 ondemand 传输法来投递：

```
example.com          ondemand
```

5. 运行 *postmap* 将传输表转换成数据库格式：

```
# postmap /etc/postfix/transport
```

6. 要求 Postfix 重新读取配置文件，使我们所做的改变开始生效：

```
# postfix reload
```

现在，所有要寄到 *example.com* 网域的邮件都会被扣留在队列里，直到你下达以下命令为止：

```
$ postqueue -f example.com
```

暂缓投递外地邮件

假设有一个小型办公室网络，他们与 Internet 之间没有连接，而是由网管人员架设了一台邮件服务器来接收内部员工寄到外界的邮件。网管人员希望能够控制外地邮件的投递时间，让 Postfix 只有在有连接时才送出外地邮件。

1. 将 smtp 服务名称列在 *main.cf* 的 defer_transports 参数中：

```
defer_transports = smtp
```

2. 要求 Postfix 重新加载配置文件，使我们的改变生效：

```
# postfix reload
```

以后，每当办公室网络与 Internet 接通时，网管人员就可以使用 postqueue -f 命令来送出所有外地邮件。

本章的其余章节描述各种 Postfix 转发邮件到其他系统的情况。在大多数情况下，若想要影响邮件传递路径，传输表是最好的工具。

入站邮件网关

邮件网关（mail gateway）是一种能收下邮件然后将其转交给其他系统的邮件服务器系统。在网络术语中，“网关”一词的意义有两个：一是物理上的含义，代表某个网络到达另一个网络的必经通路；另一层含义是协议的转换，将一种协议所传达的信息以另一种协议传到目标系统。本节与下一节分别讨论如何使用Postfix来架设入站通路与出站通路，最后一节简略说明 Postfix 如何与其他协议接轨。

邮件网关通常被设置在企业网络与 Internet 之间，甚至与防火墙系统搭配在一起，以尽可能减少需要直接访问 Internet 的服务器的数量。

假想有一个企业网络，其布局方式如图 9-2 所示。这家企业有多个部门，各部门都有自己的子网域，也有自己的内部邮件服务器。网关系统 *gw.example.com* 负责收下企业网络的所有邮件。假设人力资源部门的邮件系统位于 *mail1.example.com* ，此部门员工的邮件地址是 *user@hr.example.com* ；业务部门的邮件系统位于 *mail2.example.com* ，他们的邮件地址格式是 *user@sales.example.com* 。每个子网络上的主机，应该分别从他们各自的内部邮件服务器取信。邮件网关 *gw.example.com* 必须设置传输表，才能将收下的邮件交付给正确的内部邮件服务器。

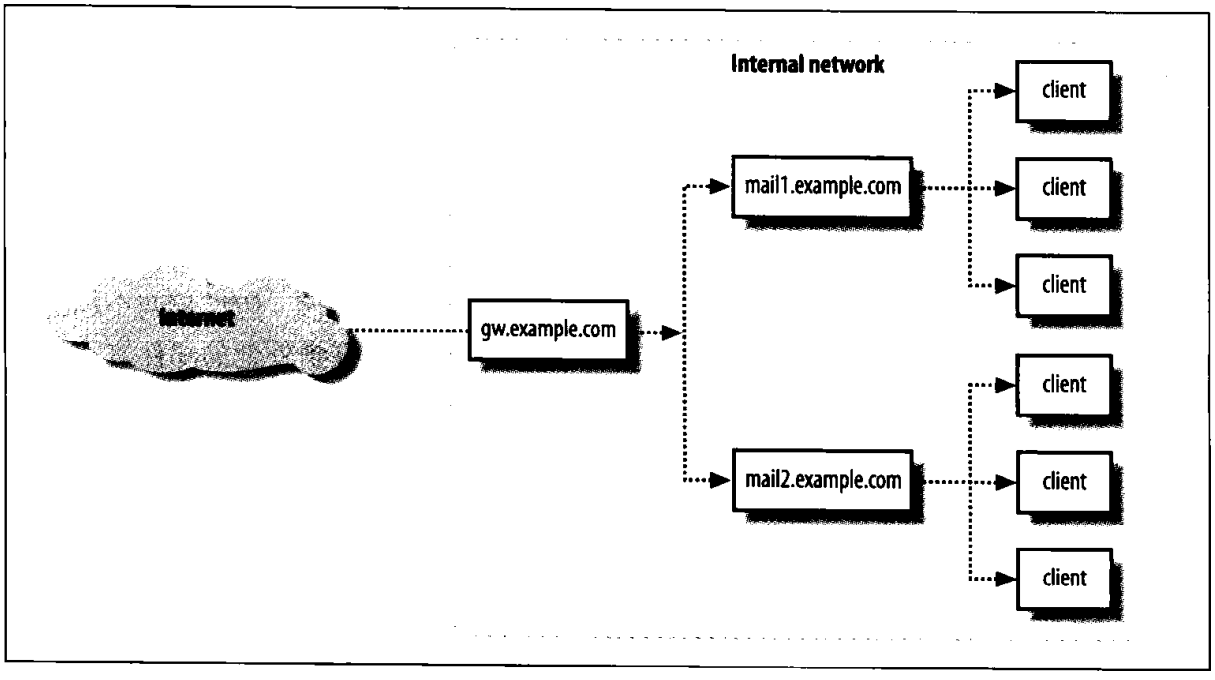

图 9-2：企业网络上的邮件网关

下列程序示范要如何设定 *gw.example.com*，才能使它将邮件交给正确的内部服务器。

1. 确定 *hr.example.com*与 *sales.example.com* 的DNS MX 记录都指向 *gw.example.com* 网关。

2. 编辑网关系统的 *main.cf* 配置文件，将两个子网域列入 relay_domains 参数：

```
relay_domains = hr.example.com, sales.example.com
```

3. 确定 transport_maps参数指向正确的传输表：

```
transport_maps = hash:/etc/postfix/transport
```

4. 编辑修改传输表内容，将各网域指向正确的内部服务器：

```
#
# transport maps
#
hr.example.com              relay:[mail1.example.com]
sales.example.com           relay:[mail2.example.com]
```

注意，内部邮件服务器的主机名称是放在一对方括号内，这使得网关系统可跳过MX查询，而直接将邮件递送到主机名称所对应的 IP地址。

5. 重新加载配置文件，使改变生效：

```
# postfix reload
```

郑重建议你将 mail1与mail2的所有合法邮箱列表汇整成一个查询表，放在gw 系统上，并将网关系统的 relay_recipient_maps参数指向此查询表（参阅本章的“转发列表”）;否则，gw.example.com将会收下许多垃圾邮件。

出站邮件网关

若邮件系统没有足够的信息或能力将邮件直接送达目的地，它可将邮件交给位于更有利位置的其他系统，间接送到目的地。同样以图 9-2 的企业网络为例，假设那两个内部邮件系统不能直接访问 Internet，它们不能将子网络上的用户要寄到外地的邮件直接递送到目的地，但它们可以将所有外地邮件都托付给网关系统，由网关代为递送。下列步骤示范如何设定 mail1.example.com 上的 Postfix server，使它将收下的所有外地邮件都交给 gw.example.com。

在开始设定内部邮件系统之前，请确定邮件网关已被设定成能接受内部邮件系统的转发要求。网关系统的 mynetworks 参数（见第四章）应该涵盖内部邮件系统的所有 IP 地址；此外，如果网关系统使用了 SMTP UBE 过滤规则（参阅第十一章），请确定 permit_mynetworks 被列在过滤规则里。

1. 在各部门的办公室贴公告，告诉你的同事，要求他们设定 MUA使用的内部邮件系统（mail1、mail2 …）为 SMTP server。

2. 设定内部邮件系统（mail1 与 mail2）的 mynetworks（或 mynetworks_style）参数，确定子网络上的全部客户端系统都被涵盖在内。

3. 编辑内部邮件系统（mail1 与 mail2）的 *main.cf*，将 relayhost 参数指向网关系统：

```
relayhost = [gw.example.com]
```

4. 重新加载 Postfix 配置文件，使改变生效：

```
# postfix reload
```

现在，所有送到 *mail1.example.com*（或 mail2）的外地邮件，都会通过 *gw.example.com* 送到目的地。

UUCP、传真以及其他投递机制

Postfix 的在线文件描述了如何设定 Postfix 使其将邮件交给传真系统以及如何使用 Postfix 来架设 UUCP 网关。当你的 Postfix 需要与任何特殊设备接轨时，这些例子提供了很好的参考。原则上，如果你需要在不同类型的系统或网络之间架设网关，传输表提供了导引邮件到其他系统或设备的机制。

第十章

邮件列表

邮件列表（Mailing list）提供了一种便利的管道，让用户只要写一个收件地址、只寄一次邮件，即可让许多人收到同一封邮件。就用户的感受而言，一个邮件列表就好像是一群人共同的邮件地址。相较于对同一封邮件指定多位收件人的方式，以服务器为基础的集中管理式邮件列表有许多优点。如果你经常寄信给同一群人，打一长串收件地址未免太不切实际，因为这太麻烦、太容易出错了。 MUA通常会有一种贴心的设计，让你能以一个容易记忆的名称，来代表一群时常联络的对象。像这种个人式的别名机制确实适合个人使用，但如果想要与其他人分享通信列表，就不是一个切合实际的解决方案了。集中控管式的邮件列表，最主要的优点在于列表成员不必实际拥有列表内容。因此，只要更改服务器上的列表，新列表就可以立刻生效，且不必发布给列表成员。其他的优点在你使用 Mailing List Managers（MLM ）来管理列表时，才会逐渐显现出来。

Postfix本身提供了支持邮件列表的基本机制，也可以与 MLM互相搭配。本章先示范如何在 Postfix本身创建简单的邮件列表，然后以两种最热门的 MLM为例，示范如何设定 Postfix将邮件传给 MLM来进行更细腻的列表管理。当你拿不定应该将列表建在 Postfix还是 MLM上时，请考虑列表的变动频繁程度、谁要负责修改列表、是否需要 MLM特有的功能（审阅、摘要、主题汇整等）。比方说，你想创建一个公开的“ 邮件列表 ”，让网络上的志同道合者可通过 E-mail来讨论特定议题，而且任何人都可以随时参与，也随时可以退出。在通信列表频繁变动的情况下， MLM 能有效分摊管理员的琐碎负担，所以是比较好的选择；相对地，如果列表成员相当固定，不会时常有新面孔加入，也很少有老面孔退出，则 MLM 反而是多余的。当然，如果情况允许的话，你也可以同时使用 Postfix 与 MLM。

当然，管理一个邮件列表不是那么简单的事，有许多层的微妙细节需要注意。如果你打算担任这样的工作，建议你先研读专门论述管理邮件列表的书籍，如 Alan Schwartz 所著的 《Managing Mailing Lists》（O'Reilly 出版）。

简易的邮件列表

Postfix对于邮件列表的支持机制，实质上就是平常的别名机制（参阅第四章），因为一个别名可代表一组邮件地址（或是记录许多邮件地址的文件）。因此，使用别名就可以模拟出邮件列表的效果。你可将代表邮件列表的别名建在系统的别名文件（*/etc/aliases*、*/etc/mail/aliases*或*/etc/postfix/aliases*），或是`alias_maps`参数所指的任何文件中。

假设你是*example.com*网域的邮件主管，你想为讨论新宇宙论的人们设立一个新的邮件列表，而你决定以*neocosmic@example.com*为此邮件列表的别名。有如下设定步骤：编辑你的*/etc/aliases*别名文件，将参与者的邮件地址列入列表：

```
neocosmic:        rgrier@oreilly.com, gmhopper@onlamp.com,
   grayburn@oreilly.com
```

完成编辑之后，使用postalias重建别名数据库：

```
postalias /etc/aliases
```

现在，任何寄到*neocosmic@example.com*的邮件，会被自动转寄到列表所列的每一个邮件地址。

邮件列表的拥有者

如果邮件不能顺利寄达列表中的任一地址，原本的寄件人便会收到退信通知函。对于小型的非公开列表，这完全是可以接受的。然而，对于开放的、公众参与的、成员众多的列表，成员之间可能彼此不认识，在这种状况下，错误通知函应该统一寄给列表管理员，而非个别的成员才对。有个不成文的惯例，即每个列表都应该有一个“拥有者”，而且其邮件地址（或别名）的格式必须类似*owner-list_alias@example.com*。以先前的`neocosmic`列表为例，其拥有者的别名应该是*owner-neocosmic@example.com*（注1），这个`owner-`别名应该指向管理员（或是比原寄件者更适合收到退信通知函的任何人）实际的邮件地址：

```
owner-neocosmic: kdent@example.com
```

如何让`owner-`别名收到错误通知函？答案是将每一封列表邮件的信封上的寄件人地址，从原寄件人改成`owner-`别名。例10-1是一封名单邮件的典型标题。

注1　某些MLM系统的惯例是将`-owner`字样放在列表别名之后，例如：*neocosmic-owner@example.com*。

例 10-1：邮件列表邮件的标题

```
Return-Path: <owner-neocosmic@example.com>
Delivered-To: rgrier@oreilly.com
Received: from cowrie.example.com (cowrie.example.com[192.168.100.7])
     by mail.oreilly.com (Postfix) with ESMTP id B712120DD5B
     for <neocosmic@example.com> Mon, 13 May 2002 11:55:40 -0400 (EDT)
Date: Mon, 13 May 2002 12:00:43 -0400 (EDT)
From: G.M. Hopper <gmhopper@onlamp.com>
X-Sender: gmhopper@cowrie
To: neocosmic@example.com
Subject: Just finished latest project
Message-ID: <Pine.GSO.4.10.10205131200230.692-100000@cowrie>
```

当 Postfix发现存在 *owner-*别名时，它会自动改写任何寄给邮件列表的邮件的标题，将信封上的寄件人地址改成 *owenr-* 别名，然后再寄给列表成员。如果你希望保留原寄件人的地址，而不希望 Postfix 擅自帮你换成 *owner-* 别名，你可将 owner_request_special 参数设定成 no：

```
owner_request_special = no
```

你也可以要求 Postfix以管理员的真实邮件地址来代替 *owner-* 别名：

```
expand_owner_alias = yes
```

在这种情况下，Postfix 使用 *kdent@example.com* 为信封上的寄件人地址，而非 *owner-neocosmic@example.com*。

虽然列表成员通常不会寄信给列表拥有者，但即使是简单的列表，只要列表成员中含有其他邮件系统的用户，你都应该设置 *owner-*别名，好让其他邮件系统的管理员能够联络到你。

另一项关于邮件列表的不成文惯例，是每个邮件列表都应该设置一个格式类似 *list_alias-request@example.com* 这样的别名（例如：*neocosmic-request@example.com*），此别名用来接受外人的入会申请、会员的退会申请或是要求提供的关于邮件列表的非技术性信息。

列表文件

当列表成员日益增多时，将成员的邮件地址另外存储成个别的文本文件，比直接写在别名文件方便。以下是在系统别名文件里引入外部文件的语法：

```
E-mail_alias: :include:/path/to/file
```

同样以 neocosmic 别名为例，假设我们要将列表成员的邮件地址另外存放在 */etc/postfix/neocosmic* 文件中，则 neocosmic 别名在 */etc/aliases* 文件里的定义应该改成下面这样：

```
neocosmic:          :include:/etc/postfix/neocosmic
```

/etc/postfix/neocosmic 列表文件含有全体成员的邮件地址，每个地址各占一行，如下：

```
rgrier@oreilly.com
gmhopper@onlamp.com
grayburn@oreilly.com
bogus@example.com
```

在此，刻意安排了一个无效地址（*bogus@example.com*），以便稍后的测试。

额外的别名文件

回顾第四章曾介绍过的 alias_maps参数，它能让你指定任意多个别名文件让 Postfix 使用。例如，倘若我们设置了多个邮件列表，每个列表里都有不少成员，我们可将不同列表的成员分别记录在各自专属的列表文件，然后设定 alias_maps 参数，使其包含所有的列表文件以及系统别名文件。此外，我们应该同时设定一模一样的 alias_database参数，好使我们用一次 newaliases命令就可同时更新所有别名数据库：

```
alias_maps = hash:/etc/postfix/aliases, hash:/etc/postfix/mail_lists
alias_database = hash:/etc/postfix/aliases, hash:/etc/postfix/mail_lists
```

或许，比较合理的做法，应该是将所有别名文件都列在 alias_database，然后将 $alias_database 赋值给 alias_maps。如果你使用其他形式的别名信息来源，则只要设定 alias_maps 参数就够了：

```
alias_database = hash:/etc/postfix/aliases, hash:/etc/postfix/mail_lists
alias_maps = $alias_database, nis:mail.aliases
```

注意，每次修改 *main.cf* 之后，都要使 Postfix重新加载其配置文件。

建立一个简单的邮件列表

让我们复习先前所讨论的一切，将关于 neocosmic邮件列表的片段知识结合在一起，正式建立 neocosmic 邮件列表。首先是设定系统别名文件（*/etc/aliases* 或 */etc/postfix/aliases*）的内容：

```
neocosmic:          :include:/etc/postfix/neocosmic
owner-neocosmic:     kdent@example.com
neocosmic-request:   kdent@example.com
```

第一行程序使得所有寄送到 *neocosmic@example.com* 的邮件，皆被转寄到 */etc/postfix/neocosmic*列表文件所列的每一个地址。此文件应该含有所有成员的邮件地址。退信通知函与行政管理的邮件，会被转寄到管理员的真实地址（*kdent@example.com*）。必要

时，用户或其他邮件系统的管理员也可以寄信给 *owner-* 别名，而用户也可以透过 `-request` 别名来要求参与或退出邮件列表或索取相关信息。

寄送到邮件列表的每一封邮件，其 `To:` 字段只包含列表别名，而不展开成列表中的所有地址（可能有成千上百个邮件地址）。列表成员所收到的邮件，其标题部分应该类似例 10-1 那样。值得注意的是，`Return-Path` 包含的是 *owner-* 别名，而非实际的寄件人（此例中的 *gmhopper@onlamp.com*）。

测试你的新列表

要测试你的新列表，请对列表别名寄出一封邮件，然后观察日志文件。例 10-2 是我们寄出的一封测试邮件到达 *neocosmic@example.com* 之后，在日志文件留下的相关记录。假设列表中的 *bogus@example.com* 是无效地址。

例 10-2：寄信到 neocosmic 邮件列表所留下的日志记录

```
postfix/local[7411]: 6C2CE20DD5B: to=<neocosmic@example.com>,
   relay=local, delay=1, status=sent (forwarded as ACDC120DD70)
postfix/qmgr[8163]: ACDC120DD70: from=<owner-neocosmic@example.com>,
   size=1121, nrcpt=8 (queue active)
postfix/local[0835]: ACDC120DD70: to=<bogus@example.com> relay=local,
   delay=1, status= bounced (unknown user: "bogus")
postfix/smtp[6556]: ACDC120DD70: to=<grayburn@oreilly.com>
   relay=mail.oreilly.com[10.82.6.11], delay=1,
   status=sent (250 Mail accepted)
postfix/smtp[6556]: ACDC120DD70: to=<rgrier@oreilly.com>
   relay=mail.oreilly.com[10.82.6.11], delay=1,
   status=sent (250 Mail accepted)
postfix/smtp[5954]: ACDC120DD70: to=<gmhopper@onlamp.com>
   relay=mail.onlamp.com[10.171.8.111], delay=1,
   status=sent (250 Message received: GZCLUC00.E8F)
```

为了简洁起见，我们刻意移除了时间与主机名称等不相干的信息。请注意第一条记录末端的注释（`forwarded as ACDC120DD70`），这表示这封邮件以一个新的 Queue ID 被转寄到其他地址，因此，后续的相关记录都是使用这个 Queue ID。第一条记录是这封邮件刚进入系统时的状况，其收件地址为 *neocosmic@example.com*。第二条记录显示 Postfix 使用 *owner-* 别名来作为信封上的寄件人地址（从 `from=<owner-neocosmic@example.com>` 可看出），但是收件地址已经改为成员的实际邮件地址了。第三条记录是我们刻意安排的无效地址所造成的，你可以发现其状态是“ bounced”（退信）。退信通知函会被寄到 *owner-*别名所指的实际地址（*kdent@example.com*），例 10-3 是这封退信通知的样本。请注意，退信通知是递送给 *owner-neocosmic@example.com*，原寄件人不会收到通知。

例 10-3：无效收件地址所造成的退信通知

```
From MAILER-DAEMON@mail.example.com Tue Jul 16 12:03:49 2002
Date: Tue, 16 Jul 2002 11:25:27 -0400 (EDT)
From: Mail Delivery System <MAILER-DAEMON@mail.example.com>
To: owner-neocosmic@example.com
Subject: Undelivered Mail Returned to Sender

...

<bogus@example.com>: unknown user: "bogus"

...
```

邮件列表管理系统

对于有固定成员的列表， Postfix 本身的支持功能就足够了。但如果成员变动频繁，人工管理的代价就显得太高了些，最好的办法是将其交给邮件列表管理系统（Mailing-List Manager，MLM）来处理。MLM 的主要作用是将列表的维护工作自动化，管理员不必自己动手编辑列表文件。每当收到订阅要求时， MLM 能自动将订阅者加入列表；遇到有人要求退订时，则自动将该成员除名。 MLM 还有管理通信内容的功能，像是定期打包来往邮件、讨论主题摘要，它甚至容许管理员事先审阅邮件内容，再决定是否要发布给所有列表成员。

Postfix与 MLM 之间的联接方法，是在 Postfix 中设置一个指向 MLM 管理程序的“管理别名”。列表管理操作 ——诸如订阅、退订、列出列表成员、回复列表基本信息、处理退信通知函甚至过滤邮件等，都是借由送出含有特定命令的邮件到管理别名来完成的。至于列表本身的运行原理，与前一节所述的简单别名完全一样。每个列表的成员都是记录在自己的列表文件中，且同样通过指向列表文件的别名来广播邮件。只不过，列表文件的管理工作由 MLM 自己负责，而不必靠人工编辑修改。

接下来的两小节，分别介绍两种最热门的 MLM：Majordomo 与 Mailman。

Majordomo

Majordomo 是始于 20 世纪 90 年代早期的老牌 MLM 之一，它提供完整的 MLM 功能。管理员只需创建初期列表，之后，几乎所有的管理工作都可借由传送特殊邮件信息以触发相关命令来完成。此外，Majordomo也提供一个网络管理接口，让管理员可远程通过网页进行控制管理工作。

Majordomo软件的主页位于 *http://www.greatcircle.com/majordomo/*,你可从该处免费下载它。Majordomo本身是以 Perl 写成的一组脚本，在本书翻译期间，最新版本是 1.94.5，

它需要用户安装 Perl4 Version 4.036、Perl5 Version 5.002 或更新版的 Perl。未来新版的 Majordomo 可能不再继续支持 Perl4。此外，Majordomo 包包含了一个以 C 写成的 wrapper 程序，如果你打算从头安装 Majordomo 包，你的系统必须要有一个 ANSI C 编译器。

如果你将 Majordomo 设定成审阅模式（邮件在发布到列表成员之前，必须经过管理员使用 Majordomo 的 `approve` 指令来批准才可以），你必须做一些调整工作，才可使 Postfix 与 Majordomo 正确地结合在一起。Postfix 会在它经手的每一封邮件的标头前端安插一个 `Delivered-To:` 字段，利用此字段来侦测邮件循环。当一封 Majordomo 邮件被递送给审阅者批准，审阅者以 `approve` 命令核准后，该邮件连同原本的标头会被送回 Postfix，以便寄给列表成员。当 Postfix 见到这封被批准的邮件时，它会发现先前加注的 `Delivered-To:` 字段，而回复发生了一次邮件循环。

实现此功能的最简单的方法，是稍微修改 Majordomo 的 `approve` 脚本（它是一个 Perl script）。你必须编辑此脚本（通常位于 Majordomo 安装目录下的 *bin/* 子目录），找出其中一个名为 `process_bounce` 的函数，在该函数里可以找到一个 `while` 循环，如下所示：

```
while (<$FILE>) {
        if (/^>?From / && ! defined($from_skipped)) {
                # Skip any initial "From " or ">From " line
                $from_skipped = 1;
                next;
        }
        next if ( /^delivered-to:/i );  # Added for Postfix
        s/^~/~~/;
        print MAIL $_;
}
```

黑体字的那一行，就是要你自己添加的程序代码。

创建一个 Majordomo 列表

以下程序逐步示范了如何使用 Majordomo 与 Postfix 来开设一个名为 `astronomy` 的邮件列表。为了方便说明，我们假设你已经在系统上创建了一个 majordom 虚账户，而且将 Majordomo 包安装在 */usr/local/majordomo/* 目录下。如果你使用的账户名称与安装目录不同于我们的假设，请依据实际情况修改调整。

1. 首先，确定你的系统安装了 Perl，而且其版本至少比 Version 5.002 新。Perl 几乎是 Unix 系统必备的工具，你可用 which（或 whereis）查出其安装位置，如果能找到，可接着使用 `perl -v` 确定其版本。例如：

```
# which perl
/usr/local/bin/perl
```

```
# perl -v
This is perl, v5.8.3 built for sparc64-linux
Copyright 1987-2003, Larry Wall
...
```

2. 从 Majordomo 主页下载一个 Majordomo 源程序包或是从你知道的其他软件来源取得预先封装好的包。解压包之后，依照随附的 INSTALL 文件的引导，将 Majordomo 安装到你的系统上。如果你要从源程序开始安装，你需要一个 ANSI C 编译器来建构这个软件（译注 1）。

   如果你想自己从头开始构建 Majordomo，当你修改 *Makefile* 与 *majordomo.cf* 文件时，你应该可以假设 Majordomo 所合作的 MTA 是 Sendmail。只要 *majordomo.cf* 里的 `$sendmail_command` 是指向正确位置（通常应该是 */usr/sbin/sendmail*），则 `$mailer` 与 `$bounce_mailer` 变量的默认值就是正确的。

3. 创建 */usr/local/majordomo/aliases* 文件来存储 Majordomo 的别名。依照 Majordomo 包随附的 NEWLIST 文件的引导，设置一个触发 majordomo 命令的别名以及两个

译注 1：我简略写下我的安装步骤，供没有软件安装经验的读者参考：

```
$ wget http://www.greatcircle.com/majordomo/1.94.5/majordomo-1.94.5.tar.gz
$ tar xzvf majordomo-1.94.5.tar.gz
$ cd majordomo-1.94.5/
$ less INSTALL (郑重建议你仔细阅读此文件)
# adduser -u 45 -g 2 -d /usr/local/majordomo -s /bin/nologin -c
  "Majordomo  User" majordomo
$ vi Makefile
```

（以下是我所修改的内容）

```
PERL = /usr/local/bin/perl
W_HOME = /usr/local/majordomo
MAN = /usr/share/man
W_USER = 45
W_GROUP = 2
W_PATH = /bin:/usr/local/bin:/usr/bin
TMPDIR = /var/tmp
$ cp sample.cf majordomo.cf
$ vi majordomo.cf
(请依据实况设定 $whereami、$whoami、$whoami_owner、$homedir、$listdir、
$log、$sendmail_command、$mailer、$bounce_mailer    等变量)
$ make wrapper
# make install
# make install-wrapper
$ cd /usr/local/majordomo/
$ ./wrapper config-test
...
Enjoy!    <-- 如果你看到这个词，表示先前的设定都对了
$
```

其余步骤请参考作者的说明。

“拥有者”别名。接着，将你的列表的别名也加入此文件。此别名文件的内容应该类似下面这样：

```
majordomo:              "| /usr/local/majordomo/wrapper majordomo"
owner-majordomo:        kdent@example.com
majordomo-owner:        kdent@example.com
# astronomy list
astronomy:              :include:/usr/local/majordomo/lists/astronomy
owner-astronomy:        csagan@example.com
astronomy-request:      "|/usr/local/majordomo/wrapper \
                         request-answer astronomy"
astronomy-approval:     csagan@example.com
```

4. 编辑 */etc/postfix/main.cf*，将 Majordomo 的别名文件列在 alias_maps 参数中：

```
alias_maps = hash:/etc/aliases, hash:/usr/local/majordomo/aliases
```

5. 你也可以将新的别名文件列在 alias database 参数中，以便你使用 newaliases 命令来重建别名数据库：

```
alias_database = hash:/etc/aliases, hash:/usr/local/majordomo/aliases
```

6. 要求 Postfix 重新读取 *main.cf* 配置文件的内容，让上述修改生效：

```
# postfix reload
```

7. 创建一个列表文件来存储 astronomy 列表成员的邮件地址。此列表文件的拥有者必须为 majordom 虚账户，这样 Majordomo 才有足够权限帮你自动维护列表文件：

```
# touch /usr/local/majordomo/lists/astronomy
# chown majordom /usr/local/majordomo/lists/astronomy
```

8. 制作一个 *info* 文件，写下寄给新成员的欢迎信，介绍你的列表的使用方法以及应该遵守的规矩。例如：

```
Welcome to the astronomy discussion list at example.com. The
purpose of this list is to discuss new astronomical phenomena.
To send a message to all the members of the list, address your
E-mail to <astronomy@example.com>.
The basic rules and etiquette for the list are as follows:
1. ...
```

9. 假设上述内容是放在 */usr/local/majordomo/lists/astronomy.info* 文件中，我们必须确定 *majordom* 虚账户有权访问此文件：

```
# chown majordom /usr/local/majordomo/lists/astronomy.info
```

10 创建别名数据库：

```
# postalias /usr/local/majordomo/aliases
```

如果你是将 Majordomo 的别名文件列在 alias_database 中，则只要下达 newaliases 命令即可。

寄一封邮件给 majordomo 别名，测试你所安装的 Majordomo 是否能正常发挥作用：

```
$ echo 'lists' | mail majordomo
```

运行上述命令后，系统会寄出一封含有“ lists ”命令的邮件到 Majordomo，收到‘ lists”命令的 Majordomo 会将它所维护的所有列表的相关信息寄回给你。以下是我们的 Majordomo 示范系统所回复的邮件内容：

```
Date: Tue, 16 Jul 2002 18:14:59 -0400 (EDT)
From: Majordomo@example.com
To: kdent@example.com
Subject: Majordomo results

--

>>>> lists
Majordomo@example.com serves the following lists:

  astronomy

Use the 'info <list>' command to get more information
about a specific list.
>>>>
>>>>
```

你或你的用户现在可以发送 Majordomo 命令到 *majordomo@example.com* ，以取得帮助信息或要求加入列表。比方说，若你自己想要加入新邮件列表，只要传送一封含有“subscribe”（订阅）命令的邮件给 majordomo 即可：

```
To: majordomo@example.com
From: tbrahe@porcupine.org
Subject:

subscribe astronomy
```

送出订阅申请之后，应该会收到 Majordomo 回复的确认信函。你必须依照信函上的指示，回复一封含有授权码的邮件，才能完成订阅手续（参阅 Majordomo 的说明文件）。

潜在问题

如果在安装 Majordomo 的过程中没遇到任何问题，前述的设定过程应该能达到预期的效果。如果遇到麻烦，则最有可能的问题是文件权限。如果你寄信到邮件列表，并收到类似下面这样的退信通知函：

```
...
                        The Postfix program

<astronomy@example.com>: cannot open include file
    /usr/local/majordomo/lists/astronomy: Permission denied
```

当 Postfix 启动 Majordomo 来发布信息给列表成员时， Majordomo 的权限要足以读取列表文件(*/usr/local/majordomo/astronomy*) 以及列表的配置文件(*/usr/local/majordomo/astronomy.config*)。当 Majordomo 的 wrapper 被 Postfix 启动时，所得到的身份权限是“含有 `majordomo` 别名的数据库文件（即 */usr/local/majordomo/aliases.db*）的拥有者”(Postfix就是因为读取了该别名数据库文件才触发了 Majordomo)。为了确保 `wrapper` 有足够的访问权限， Majordomo 的安装程序会将 `wrapper` 程序文件的拥有者设定为 *root*，并设定其 SUID 权限位。这表示不管是以哪个身份来启动 wrapper 程序，wrapper 进程都有 *root* 权限。如果你的安装程序出了问题，以至于没有妥善设定 wrapper 程序文件的拥有者或权限位，你就会看到上述的退信通知。解决这一问题的办法很简单，只要修正 wrapper 程序文件的权限位与拥有者即可：

```
# chmod 4755 /usr/local/majordomo/wrapper
# chown root /usr/local/majordomo/wrapper
```

不过，比较好的解决方法是将 */usr/local/majordomo/aliases.db* 的拥有者设定成 *majordom* 虚账户，这样，当 Postfix 开启该文件时以及 wrapper 程序被 Postfix 触发时，两者的身份都是没有危险性的 *majordom*，但同时又有足够权限来访问相关文件。

Mailman

Mailman 是继 Majordomo 之后的另一个全功能 MLM。它是 GNU Project 开发的软件，你可从 *http://www.gnu.org/software/mailman/* 免费取得它。Mailman 也提供了一个 web-based 管理接口，而且你可以分别为每一个论坛都创建专属的主页，供版主与成员进行管理工作。此外，它也能像 Majordomo 那样，让管理员可以通过 E-mail 来传达管理命令。

Mailman 本身是一组以 Python 写成的程序（需要搭配 Python 2.1 以后的版本），但是它的封装程序（类似 Majordomo 的 wrapper 那样）也是以 C 写成的，所以，你的系统必须要有 ANSI C 编译器才能重头编译、构建这套软件。

要使 Postfix 与 Mailman 能够正确地结合在一起，需要稍微多一点的技巧。因为 Mailman 预期它会被特定 GID 的进程所启动。然而，这个 GID 值是在编译期决定的，而不能通过配置文件来修改。因此，如果你打算自己编译 Mailman 包，请先在你的系统上设置一组名称为 `mailman` （或任何你喜欢的名称）的账户与组。如下：

```
# /etc/passwd
mailman:x:41:42:GNU Mailing List Manager:/var/mailman:/bin/bash
# /etc/group
mailman:x:42:
```

请确定 mailman 账户的主要组是 mailman 组。以本例而言，mailman 组的 GID 为42（第三栏），所以 mailman 账户的第四栏也必须是 42。

当你运行 configure Mailman源程序时，确定你加上了一with-mail-gid=*xxx*选项，其中的 *xxx* 是 mailman 组的实际 GID值。依照刚刚的例子，你应该像下面这样运行 configure：

```
$ ./configure --with-mail-gid=42
```

当然，configure 还有其他选项，请依照你的系统环境来决定要使用哪些选项。我们郑重建议你先详读 Mailman包的说明文件，然后再决定要使用哪些编译选项。如果你的系统上已经安装好现成的 Mailman 包，那就再重建一次吧。如果你不想自己编译 Mailman 包，请参阅“想要 GID 12却得到 GID 99？”的说明。

想要 GID 12却得到 GID 99？

如果你使用别人预先编译好的 Mailman 包（而且你无法自己编译它），除了观察错误信息之外，没有其他好办法能够知道 Mailman到底需要哪一个 GID。如果你的 Postfix 进程的 GID不符合 Mailman的预期，在你对 Mailman所管理的列表送出一封邮件之后，你将会收到一封退信通知函，而 Mailman也会在日志文件留下错误信息：

```
Failure to exec script. WANTED gid 12 GOT gid 99 (Reconfigure
to take 99?)
```

为了让 Postfix 能以正确的 GID传信给 Mailman，你必须修正 Mailman 别名文件的拥有组。当 Postfix 进行标准的本地投递操作时，local进程会切换成收件人的身份；如果收件地址是一个别名，则会切换成别名文件拥有者；但如果该拥有者为 *root*，则切换成 default_privs 参数所指定的身份。因此，你要将 Mailman别名文件的拥有者设定为 mailman账户，而此账户以 mailman GID 为其主组。唯有这样，Postfix 才会拥有 mailman 组成员的身份，而得以传信给 Mailman 系统。

如果你不能自己编译 Mailman 包，当然也就不能控制 Mailman使用哪一个 GID，所以只能想办法让 Postfix迎合 Mailman 的要求。要产生像上面那样的错误信息，首先你得制作一份列表（参阅本节的步骤），然后对该列表送出一封邮件，再等着接收退信通知（或是观察 Mailman 的日志记录），最后记下 Mailman 要求的 GID（“WANTED gid 12 ...”）。将 mailman 账户的主组改成 Mailman 要求的 GID 值，并再次确定 Mailman别名文件的拥有者是 mailman 账户。

制作 Mailman 邮件列表

接下来的步骤逐步示范如何使用Mailman与Postfix来成立一个astronomy邮件列表。我们假设你的系统已设置了一对mailman账户与组，而且准备将Mailman包安装在*/home/mlm/*目录下。

1. 请确定你的系统安装了Python，而且至少是Version 2.1以上的版本。直接在命令行下达python命令，就可以知道自己的系统是否安装了Python，以及所安装的版本（按Ctrl+D可离开Python编译器）：

```
$ python
Python 2.2.3 (#1, Oct 15 2003, 23:33:35)
[GCC 3.3.1 20030930 (Red Hat Linux 3.3.1-6)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> ^D
```

2. 取得并安装Mailman包。我们建议你从Mailman的主页下载最新版本，并且自己重新编译（译注2）。如果使用别人预先编译好的包，你可能要面对先前说过的GID匹配问题。

3. 创建一个独立的别名文件来存储Mailman的所有别名，并正确设定该别名文件的拥有者与组。将操作身份切换成*mailman*，然后依序运行下列命令（假设你的*mailman*账户的主目录为*/home/mlm/*，而且你想将Mailman别名文件放在这目录下）：

```
# su - mailman
$ cd /home/mlm
$ touch aliases
$ postalias aliases
```

由于我们先将身份切换成*mailman*，然后才创建别名文件与数据库文件，所以产生出来的文件属于*mailman*账户。

译注2：以下是我在自己的RedHat Linux系统上的编译步骤与安装过程。我使用的环境值不同于作者的示范，谨供读者参考：

```
# /etc/passwd :
mailman:x:16:17:GNU MLM:/home/mlm:/bin/bash
# /etc/group :
mailman:x:17:
[root@mlm /usr/src/]# tar xzvf mailman-2.1.4.tgz
[root@mlm /usr/src/]# cd mailman-2.1.4
[root@mlm /usr/src/mailman-2.1.4]# mkdir /home/mlm
[root@mlm /usr/src/mailman-2.1.4]# chown mailman.mailman /home/mlm/
[root@mlm /usr/src/mailman-2.1.4]# chmod 02775 /home/mlm/
[root@mlm /usr/src/mailman-2.1.4]# ./configure --with-mail-gid=16 \
>    --prefix=/home/mlm --mandir=/usr/share/man
[root@mlm /usr/src/mailman-2.1.4]# make
[root@mlm /usr/src/mailman-2.1.4]# make install
```

4. 编辑 */etc/postfix/main.cf* 配置文件，将 Mailman别名文件列入 alias_maps 参数：

```
alias_maps = hash:/etc/aliases, hash:/home/mlm/aliases
```

5. 建议你将该别名文件也列入 alias_database 参数，以便往后可以使用 newaliases 命令来重建数据库：

```
alias_database = hash:/etc/aliases, hash:/home/mlm/aliases
```

6. 重新加载 Postfix 的配置文件，使我们所做的改变生效：

```
# postfix reload
```

7. 运行 Mailman 的 newlist 命令来初始化你的新邮件列表。 newlist 的输出信息中，含有你应该放在 */home/mlm/aliases* 文件里的内容。参考例 10-4，将黑体字的部分“剪贴”到 */home/mlm/aliases* 文件里（译注 3）。

8 重建新的别名数据库：

```
# postalias /home/mlm/aliases
```

或者，如果你将 Mailman 别名文件列在 alias_database，则只要运行 newaliases 命令就可以了。

例 10-4：Mailman newlist 命令的输出

```
[root@mlm /home/mlm]# bin/newlist
Enter the name of the list: astronomy
Enter the E-mail of the person running the list: kdent@example.com
Initial astronomy password:
To finish creating your mailing list, you must edit your /etc/aliases (or
equivalent) file by adding the following lines, and possibly running the
newaliases' program:

## astronomy mailing list
astronomy:              "|/home/mlm/mail/mailman post astronomy"
astronomy-admin:        "|/home/mlm/mail/mailman admin astronomy"
astronomy-bounces:      "|/home/mlm/mail/mailman bounces astronomy"
astronomy-confirm:      "|/home/mlm/mail/mailman confirm astronomy"
astronomy-join:         "|/home/mlm/mail/mailman join astronomy"
astronomy-leave:        "|/home/mlm/mail/mailman leave astronomy"
astronomy-owner:        "|/home/mlm/mail/mailman owner astronomy"
astronomy-request:      "|/home/mlm/mail/mailman request astronomy"
astronomy-subscribe:    "|/home/mlm/mail/mailman subscribe astronomy"
astronomy-unsubscribe:  "|/home/mlm/mail/mailman unsubscribe astronomy"
```

译注 3： 如果你不会“剪贴”，我有更简单的方法：

```
[root@mlm /home/mlm]# bin/newlist -q ortbooks lin@oreilly.com.tw
    mypass >/home/mlm/aliases
[root@mlm /home/mlm]# vi /home/mlm/aliases
```

（已删掉不必要的额外文字）

```
Hit enter to notify astronomy owner...
```

现在，你可以寄信到 *astronomy-request@example.com* 来取得帮助，用户也可以寄信到该地址来申请加入列表。你也可以使用 Mailman 的 web-based 或 E-mail-based命令接口来设定列表的选项。关于 Mailman的用法与高级功能，请自行参阅它的说明文件。

第十一章

反垃圾邮件

“不请自来的大宗邮件”(Unsolicited Bulk E-mail，UBE），或“不请自来的广告邮件”(Unsolicited Commercial E-mail，UCE)，就是一般俗称的“垃圾邮件”（spam)。寄件人没有事先征得收件人的同意，就擅自寄送大量（烦人的）邮件给一大群陌生的收件人，这样的行为称为“滥发垃圾邮件”(spamming），而专门干这种事的人，叫做“垃圾邮件发送者”(spammer)。垃圾邮件之所以存在，是因为寄送成本便宜。邮递量从几百人增加到几千人并不会增加太多成本，所以，垃圾邮件发送者的目标就是尽量收集多的邮件地址。本章要探讨垃圾邮件的问题以及 Postfix 在这方面提供了哪些抵制措施。

垃圾邮件的本质

大部分的垃圾邮件都是商业广告，但是这些广告绝大部分都是唬人的骗局。垃圾邮件发送者才不在乎邮件内容是否能引起收件人的兴趣，而且他们的信息通常就是谎言，若有其事地宣称收件人曾经跟他们公司有过怎样的关系，或是从天而降的伙伴那时得知你需要什么信息，所以特地写信给你云云……简直是鬼话连篇。有些信息被刻意编造成煞有其事的样子，让你误以为有两个人正在进行某种交易，谈论到他们的产品或服务有多么好，而你是因为他们不小心寄错邮件所以才会收到邮件……总之，他们想尽办法意图勾起你贪小便宜的心，而诱使你上当。

垃圾邮件经常提供“我不要再收到这样的邮件”的指示，比方说，要求你回信到某个邮箱，或是到某个网页填写你的邮件地址，或是点击一个含有特殊查询字符串的网址等。然而，这样的选退机制本身通常也是骗局，垃圾邮件发送者利用这种方法来确定你的邮件地址确实是有效的，而且你是属于那种比较容易受骗上当的人。换言之，如果你真的傻傻地遵照指示执行，换来的只是被更多垃圾邮件发送者知道你的邮件地址，而你也将会收到更多垃圾邮件。

垃圾邮件发送者通常会试图隐藏他们行踪的线索，避免被人逆向追查出来。假的回信地址与捏造的邮件标题，是最常使用的伎俩。他们在 Internet 上到处搜寻没有妥善设定的系统，好让他们能够匿名寄出大量垃圾邮件。更高明的技术是入侵防备不当的系统，植入他们刻意设计的秘密 relay server 。此外，垃圾邮件的内容通常经过特殊编码或是安插随机字母，试图通过过滤程序的分析。

垃圾邮件发送者使用的某些伎俩，其副作用甚至比滥发垃圾邮件的行为本身更糟糕。就“漫无目标法”(scatter-shot）而言，他们不管邮件地址实际存在与否，而盲目地将垃圾邮件送到他们认为可能存在的邮件地址。另一种“字典攻击法”则是预先收集所谓的“人名”,并从 DNS 查出各网域的 MX服务器，集中寄出垃圾邮件到成功可能性最高的邮件地址。两种伎俩都会无端造成受害系统的额外负担。

垃圾邮件的问题

虽然小规模的垃圾邮件似乎不是什么大问题，但是在 Internet 上，这却是相当严重的问题，甚至被认为是电子商务的杀手。一个为几百人或几千人提供邮件服务的系统，如果每人每天都收到几百封垃圾邮件，那这个系统就快要瘫痪了。垃圾邮件的发送成本很低，但是垃圾邮件受害人却要付出真实的代价，这显然是不公平的，因为垃圾邮件无端地占据了收件人的邮箱配额，浪费收件人的磁盘空间与网络带宽。

垃圾邮件带来的另一项损害是浪费技术支持人员的时间。虽然技术人员或管理员可以帮助用户清理被塞爆的邮箱，但是，垃圾邮件的量有时候大到系统甚至无法胜任其原来的任务（因为带宽或磁盘空间被占满）。垃圾邮件的这种副作用，无异于刻意的“拒绝服务攻击”(Denial-of-Service attack)。就算没严重到这种地步，夹杂在正常邮件中的垃圾邮件还是会干扰用户的阅读，如果垃圾邮件太多，甚至容易使人忽略重要邮件或误删正常邮件。

关于垃圾邮件的一项重要课题，在于如何处理寄给不明用户的邮件。有些邮件系统在发现收件地址无效时，当场就予以拒收（reject)，邮件没机会进入队列；有些系统则必须先收下邮件，然后才当成“无法投递的邮件”而退信，而这些退信很容易地就塞满队列，进而干扰合法邮件的递送。由于垃圾邮件的回复地址通常是无效的，所以退信不能立刻寄出去，而会在队列里待上好一阵子，在服务器尝试了多次无谓的递送之后，才会因为过期而被移出队列。

垃圾邮件发送者还有一种令人发指的恶行，他们会冒充无辜第三者的名义来滥发垃圾邮件，使得该受害者收到大量的退信通知。大量滥发垃圾邮件时，发信系统势必时常遭到退信（或被当场拒收）。当垃圾邮件发送者使用别人的 open relay （参阅下一节的说明）

滥发垃圾邮件时，如果放任让退信或拒收通知函寄送到管理员的邮箱，势必会引起管理员的警觉，一旦管理员关闭掉 open relay 或修补系统漏洞之后，他们将不能再继续使用该系统滥发垃圾邮件。因此，他们冒用无辜第三者的邮件地址为发信地址，使退信通知送到该无辜地址，借此延后管理员发觉系统被滥用的时间。而那位被冒用名义的无辜受害人将收到上千甚至上百万封退信通知。这种现象我们称之为“背黑锅”，因为受害人并未参与滥发垃圾邮件的恶行。在大多数情况下，摆脱黑锅的唯一办法是放弃那个地址，改用另一个新地址。

开放转发（Open Relay）

如果你的邮件系统不得不服务来自 Internet 的客户端，你有责任避免系统成为开放转发（Open Relay），以免垃圾邮件发送者使用你的系统来转发垃圾邮件。 Open Relay 是一种特殊的邮件系统，这类系统不限制客户端的位置，也不查证邮件是否寄给自辖网域，就愿意收下并代为转寄到最终目的地。在垃圾邮件还不泛滥的年代，邮件系统管理员多半愿意提供转发服务，好让他们的用户不管旅行到世界哪一个角落，都能通过他们的服务器寄信。因此，开放转发服务是当时的 SMTP 服务器软件（ Sendmail 8 之前的版本）的默认配置。然而，随 Internet 的普及，立意很好的 Open Relay 现在反而成为垃圾邮件跳板的代名词。因为对于邮件的终点站而言，这类邮件看起来就像是直接从 Open Relay 发出的一样，垃圾邮件发送者就是利用此一特性来隐藏真实的发信地点。现在，几乎所有新一代的 SMTP软件的默认配置都会限定转发功能的服务对象，而 Postfix 当然也不例外。

倘若你的系统被当成 Open Relay 使用，最可能出现的症状是寄信效率明显变低，因为它正忙着寄送大量的垃圾邮件，无暇递送合法用户的正常邮件。如果你愿意让自己的系统成为垃圾邮件的帮凶（当然，这是你的自由），你还得小心有心人利用你的 Open Relay 来攻陷其他系统。而且在你的系统转发了大量的垃圾邮件之后，你的网络将有很高的机率被列入黑名单，到那时候，将有许多站点拒收来自于你的网络的全部邮件 —— 包括垃圾邮件与合法邮件。第四章讨论了如何架设 Postfix ，才能避免系统被用于非法用途。

辨别垃圾邮件

只要你的系统不被当成 Open Relay 使用，至少可以放心它不会被用来攻击其他系统，而你下一步，应该是保护自己以及你的用户，想办法尽量减少收到垃圾邮件。理论上，邮件服务器应该能够拒收任何看起来像垃圾邮件的邮件。然而，虽然人类有能力一眼就认出垃圾邮件，但是要求计算机精确认出垃圾邮件而不发生误判，根本是不可能的。实际上，一旦你开始拒收垃圾邮件，必定有可能将合法邮件误挡在门外。

拒收被误当成垃圾邮件的合法邮件，这种事件我们称之为误判（false-positive identification）。你的垃圾邮件抵制措施要尽量达成两个互相违背的目标：尽可能挡掉可疑的垃圾邮件和尽可能降低误判机率。你采取的管制措施越严格，发生误判的风险就越高。要采取严格到何种程度的管制措施，取决于发生误判所造成的严重性有多高。对于完全不容许误判的环境（比方说，专用来接受客户订单的邮件系统），只能采取最宽松的管制，也就是接收所有的垃圾邮件。相对地，如果不在乎误判或是能够事先协调好用户，则可以采取最严格的管制：只有事先特许的个人才能使用邮件系统，也就是所谓的“白名单”(whitelist)。事先批准似乎过于严苛，但是，当垃圾邮件问题已经严重到不可收拾的地步，白名单已经成为越来越普遍的选择。

计算机要如何判断来信是否为垃圾邮件？主要判定依据有二，首先是“邮件来源”，其次是“信息内容”。前者是在收信期间检查送信方是否为已知的垃圾邮件来源，或是其寄信行为是否符合滥发垃圾邮件的条件。后者是在收下邮件后，检查信息内容是否包含可被判定为垃圾邮件的字眼或其他条件。先不管实行细节上的困难，只要邮件管理员能够预先设想到各种判定技术的盲点，就能够兼顾“尽可能挡掉可疑的垃圾邮件”与“尽可能降低误判机率”这两项彼此违背的目标。

依据客户端判别垃圾邮件

这里所谓的“客户端”是指在 SMTP 交互过程中的寄信方。如你所知，在 SMTP对话过程中，服务器端（收信方）可得到对方的 IP 地址、主机名称与邮件地址。这些信息片段都可以用来与一组已知的黑名单比对，借此决定应该接受还是拒收邮件。被列在黑名单中的系统，有可能真的是垃圾邮件发送者拥有的主机，但也有可能是无辜的 Open Relay 系统。目前，Internet 上已经很难找到“刻意为了提供转发服务而架设的服务器主机”，大部分的 Open Relay 多半是由倒霉的、无辜的或无知的管理员所架设的系统， Open Relay 并非他们的本意。但不管是哪一种情况，如果你发现某特定对象经常给你寄垃圾邮件，你可以考虑将对方列入黑名单。不过，依据 IP 地址、主机名称或邮件地址来判定垃圾邮件，最大的问题是这类信息都可被假造。虽然伪造 IP 地址需要非常高的技术，但是信封地址是很容易假造的。

DNS-based 黑名单

为了抑止 Internet 上垃圾邮件的日益泛滥，各种自发性的抵制技术与网络服务应运而生，其中一类颇有争议的技术称为“DNS-based 黑名单”(DNS-based Blacklists，DNSBL)或“实时黑名单”(Realtime Blacklists，RBL)。这类技术的原理，是将已知的 Open Relay 或垃圾邮件来源记录在一个动态数据库，并通过 DNS 系统（或其他方法）开放给 MTA server 查询。他们的构想是抢先一步提供预判结果给合法的 MTA server，减轻邮件管理

员自己维护黑名单的负担。以往，垃圾邮件发送者还会使用自己的系统来滥发垃圾邮件，或是搜寻 Open Relay 来转发垃圾邮件；近年来，这群人的行为越来越猖獗，他们几乎已经不再使用自己的系统发邮件，而是寻找有机可乘的受害系统，植入可让他们滥发垃圾邮件的代理软件，而被胁持的系统甚至可能被用来发动DoS攻击。有些机构的DNSBL专门提供这类非自愿的垃圾邮件转发的查询服务。

通常，RBL系统的查询服务是通过 DNS 提供的。对于每一个被认定为垃圾邮件来源的IP地址，在RBL所有者的网域空间里都有一个对应的PTR 记录，MTA在收信期间向RBL系统查询客户端的 IP 地址，借此决定应该接受还是拒收邮件。举例来说，假设有一家（虚构的）RBL 服务机构，他们的网域名称为 *nospam.example.com*，位于 192.168.254.31 的主机已经被他们认定为垃圾邮件来源，于是， NoSpam 公司在他们的网域空间里创建了一个 DNS PTR 记录：31.254.168.192.nospam.example.com。当 192.168.254.31 主机连接到你的 Postfix 系统，Postfix 可查询该 IP地址是否在 *nospam.example.com* 网域下有一个 PTR 记录，如果有，表示该 IP地址已被认定为垃圾邮件来源，Postfix可当场拒绝该主机的寄信要求。

在你决定使用 DNSBL服务之前，必须非常谨慎小心。有许多被用来转发垃圾邮件的 Open Relay 系统，它们本身也提供服务给一般的正常用户。你虽然挡掉了来自这些系统的垃圾邮件，但也挡掉了源自同系统的合法邮件。此外，将是否收信的决定权交给无责任的第三方，由别人决定谁能够谁不能够寄信给你的用户，是一种推卸责任的行为。但如果你真的深受垃圾邮件之害，在你研究出适当的抵制方法之前， DNSBL 服务确实能够帮上忙。如果你决定使用某家公司的服务，请务必仔细检视他们的服务条款与政策。再一次地，你在“尽可能挡掉可疑的垃圾邮件”与“尽可能降低误判机率”之间取得平衡点。

依据内容判别垃圾邮件

除了以客户端为判别条件之外，邮件内容通常也可作为供判别垃圾邮件的特征。垃圾邮件时常出现某些特定的广告语（例如“ Our Rates Have Never Been Lower!!”） 。依据邮件内容来辨识垃圾邮件，并非没有误判的可能。想象一下，如果你经常收到大量关于低率房贷的垃圾邮件，你可能会想要挡掉含有类似“ lowest mortgage rate”词组的邮件。当然，这样确实可挡掉不少垃圾邮件，但是，你可能也挡掉了银行同意核发低利率贷款给你的朋友（或是某位用户的朋友）的通知信。

判别技术的困难处

不管以客户端还是邮件内容作为判别依据，垃圾邮件发送者总是有办法找到办法避过你

的抵制措施。合法邮件与垃圾邮件之间，总是一有段难以厘清的模糊地带。当你努力收集黑名单的同时，垃圾邮件发送者也正在寻找新的代罪羔羊（这种无辜系统的数量之多，远超过你的想象）。

你可能发现，在你收到的垃圾邮件中，有许多具有相同的回信地址。当然，你确实可使用回信地址来阻挡垃圾邮件，但是，他们多半采用“边打边跑”战术。他们先向免费邮件网站申请一个邮箱，然后使用该地址送出千万封垃圾邮件。过了几天之后，他们便放弃原来的邮箱，另外再申请一个新邮箱。所以，被你列入黑名单的回信地址，过不了几天就不会再出现了。

就算是专门的内容过滤系统，也必须随垃圾邮件发送者的干扰伎俩而调整。举例来说，很多滤信软件会挡掉正文含有“ viagra”字样的邮件，于是，垃圾邮件发送者便将 HTML 注释标记嵌在这类关键词句中 —— 将“ viagra ”改成‘ vi<!--oxo-->agra”（译注 1），借此避开过滤程序的字符串比对条件。如果你调整过滤条件，挡掉含有 HTML 注释标记的邮件（好主意！虽然有点风险），过几天，你会发现他们又换花招了，含有 *v1agra* 或 *vi@gra* 字样的垃圾邮件将闯进了你的系统，于是，你又被迫增加新的过滤条件。因此，如果你决心向垃圾邮件发送者宣战，你最好有个心理准备，这会是一场长期抗战。

反垃圾邮件的措施

一般来说，侦测到垃圾邮件之后，你有下列几种选择：

在SMTP对话过程中，当场拒收垃圾邮件。于第一时间把垃圾邮件挡在大门之外确实是个好主意，因为垃圾邮件不会进到邮箱，而你也不必担心后续的处理工作，发信方会承担处理拒信错误的善后工作。然而，如果你的站点不能够容忍拒收合法邮件的后果，或许你应该考虑第二种选择。

暂时将可疑的垃圾邮件另外存储在专用邮箱，再研究一套事后程序来处理它们。比方说，定期检阅被集中起来的可疑垃圾邮件，确定其中没有合法邮件之后才一并删除。虽然这样可确保不错失任何合法邮件，但是却可能耽搁合法邮件的时效性，除非你的站点有足够的人力（与财力）请人专门检阅可疑的垃圾邮件。此外，事后的检阅工作相当无聊，而且有道德风险（万一看到不该看的邮件，怎么办？）。因此，你应该考虑将垃圾邮件存放在个别用户能看到的专用邮箱。不过，这需要用户改用 IMAP协议来收信，如果你的用户都使用 POP协议收信，这可能不是个好办法。

译注 1：由于 MUA 软件多半具有 HTML 自动译码能力，所以，大多数收件人看到的依然是“viagra”，而感觉不到 HTML 注释标记造成的差异。

将可疑的垃圾邮件打上某种特殊标记，投递到用户的正常邮箱，由用户收下邮件后自行判断。比方说，用户可用他们的 MUA或 MDA软件设定一个过滤条件，将含有特殊标记的可疑邮件与正常邮件分开存储。这种技术的好处，在于用户可自己决定如何处理垃圾邮件，既不会错失合法邮件，也不必担心私密邮件曝光。 Postfix 目前没内置可标示垃圾邮件的机制，但是 Postfix 可以搭配外部的内容过滤程序（例如：Spamassassin），由外部程序处理垃圾邮件的分析与标示事宜（参阅第十四章）。

分析垃圾邮件是件沉重的负担，如果运行MTA的机器还有充足资源来分析垃圾邮件，那么，在第一关就挡掉垃圾邮件通常是最理想的选择。但如果需要较大的灵活性，将过滤工作交给 MDA或MUA 层来执行或许比较合理。不过，更理想的选择是前两种技术的结合：在 MTA 层挡掉明显的垃圾邮件，将可疑的邮件交给 MDA 或 MUA 层处理。

在协助辨识垃圾邮件来源、拒收垃圾邮件的功能方面， Postfix 是相当优秀的工具。用 Postfix 阻挡垃圾邮件所需的资源，远低于外部过滤程序所需的资源。如果你担心误判合法邮件， Postfix还提供了一些补救的措施。

Postfix 的挡信机制

本章的其余内容讨论阻挡垃圾邮件的技术层面的问题，解释 Postfix提供的 UBE 检查机制。按照 UBE的判别依据，Postfix 提供的检查机制可分为四大类：

客户端判别规则

Postfix提供四种检查客户端身份的参数，每一种参数都可设定一系列决定如何响应客户端的限制条件（restriction）。如果条件成立，可能的响应动作包括 OK（收下邮件）与 REJECT （当场拒收）。例如，你可以设定一条检查规则，挡掉来自特定IP地址的邮件。相对地，如果条件不成立，则由后续条件继续处理（这种结果通常以 DUNNO 表示）。

语法检查参数

Postfix 内置一系列专用于检查语法的参数，可用来核验客户端的 SMTP 对话内容是否符合标准规定。由于垃圾邮件发送者通常不遵守标准规定，则对于不符合规定的客户端或邮件，你可以要求 Postfix予以拒收。有些语法检查参数也可作为客户端判别规则。

内容检查

你可以将一组描述垃圾邮件特征的正则表达式（ regular expression)写在查询表中（称为“ 模式表 ”),要求 Postfix 依据样式表来检查邮件的标题与正文内容。

自定义过滤规则

你可将一系列内置的限制条件组织成新的过滤规则。

设定 Postfix 的垃圾邮件辨识参数时，你还必须指定如何处理被辨认出来的垃圾邮件。一般而言，Postfix 可以当场拒收，或是收下邮件但是暂存在另一个队列，或是交给外部过滤程序去处理。

客户端判别规则

下列过滤规则可设定一系列对客户端信息的限制条件：

- `smtpd_client_restrictions`
- `smtpd_helo_restrictions`
- `smtpd_sender_restrictions`
- `smtpd_recipient_restrictions`
- `smtpd_data_restrictions`

上述的每一项参数，分别用于检查 SMTP 对话过程中的特定阶段。在每一个阶段，客户端分别提供不同类型的信息。 Postfix 依据你给各规则设定的限制条件来检查这些信息。图 11-1 描绘了上述参数与 SMTP 对话过程各阶段的对应关系，其中的 `header_checks` 与 `body_checks` 参数是“内容检查”的议题。

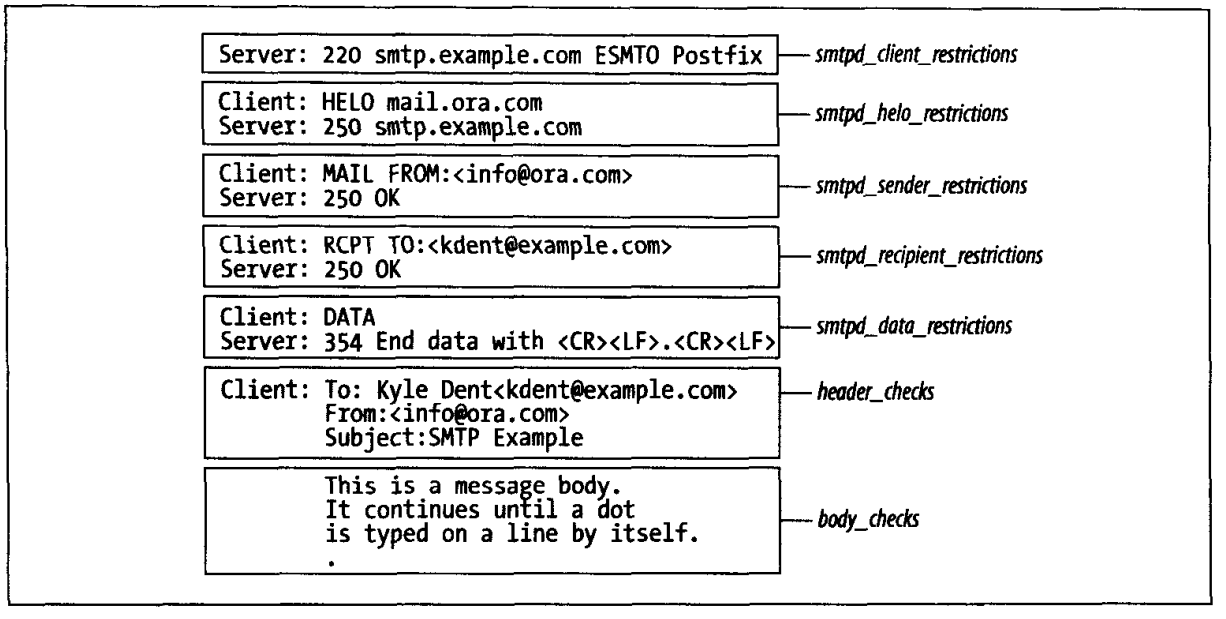

图 11-1：SMTP 对话过程以及各阶段对应的限制条件

让我们先复习 SMTP的对话过程，以及上述参数在各对话阶段的作用。

SMTP 对话过程（简述）

曾经阅读过第二章的读者，应该已经很熟悉图 11-1 的 SMTP 对话过程。例 11-1 是这段对话所产生的日志记录。首先，一个 SMTP client 通过 socket 连接到 Postfix 。由于是 socket 方式连接，所以 Postfix 在建立连接时就可以知道客户端的 IP 地址。图 11-1 看不到客户端的 IP地址，但是 Postfix会将它记录在日志文件中。 smtpd_client_restrictions让你可依据客户端的IP 地址或主机名称来决定是接受还是拒收该信息。

例 11-1：SMTP对话过程的日志记录

```
1. postfix/smtpd[866062]: connect from mail.ora.com[10.143.23.45]
2. postfix/smtpd[866062]: D694B20DD5B: client=[10.143.23.45]
3. postfix/cleanup[864868]: D694B20DD5B: \
      message-id=<20030106185403.D694B20DD5B@smtp.example.com>
4. postfix/qmgr[861396]: D694B20DD5B: from=<info@ora.com>, \
      size=486, nrcpt=1 (queue active)
5. postfix/local[864857]: D694B20DD5B: to=<kdent@smtp.example.com>, \
      relay=local, delay=98, status=sent (mailbox)
6. postfix/smtpd[866062]: disconnect from mail.ora.com[10.143.23.45]
```

连接成功之后，客户端送出 HELO命令来显示自己（送信方）的主机名称。 Postfix 依据 smtpd_helo_restrictions 的限制条件来检查这个主机名称（译注 2）,借此决定应该接收还是拒收邮件。

下一步，客户端发出 MAIL FROM命令来显示寄件人的邮件地址，接着以一个 RCPT TO 命令来表明收件人的邮件地址。寄件人与收件人的邮件地址的限制条件，分别设定在 smtpd_sender_restrictions 与 smtpd_recipient_restrictions中。

如果到 DATA 命令为止之前的一切都可以接受，客户端就可以开始传送邮件的内容。邮件内容分成两部分，前半部是标题（header），由 header_check 过滤；后半部是正文（body），由 check_body 过滤。关于邮件内容的检查，请参阅本章的“内容检查”。如果最后的标题与正文检查也过关， Postfix 就收下信息，并交给适当的 MDA执行投递操作。

Postfix 以“响应码”让客户端知道它的要求被接受还是被拒绝。关于 SMTP标准响应码请参阅第二章，本章只考虑在 4xx 和 5xx 范围内的响应码。

译注 2： 很多垃圾邮件发送者在 HELO 步骤送出的是“收信方”（SMTP server）的主机名称。任何遵守规定的 SMTP client 都不会这样做，所以你可以在 smtpd_helo_restrictions 设一个限制条件，禁止对方使用我们的主机名称。

设定限制条件

当你设定 Postfix 的 UBE 限制条件时，不一定要将它们分别设定给不同的参数，可以集中在同一个参数中，而对其他参数不加任何限制。如果你的 *main.cf* 没有设定 UBE 过滤规则，则 Postfix 的默认配置看起来就像这样：

```
smtpd_client_restrictions =
smtpd_helo_restrictions =
smtpd_sender_restrictions =
smtpd_recipient_restrictions =
     permit_mynetworks, reject_unauth_destination
```

这使得只有局域网络上的主机可以通过 Postfix 来转发邮件，而其他系统一律不允许，除非它们的信是寄给 Postfix 所辖网域的用户。

Postfix 提供一组内置的限制条件，而你也可以用访问表（access map）来定义自己的限制条件（参阅表 11-1）。许多人第一次面对 Postfix 的 UBE 限制条件时，常误以为某些条件只能用于特定过滤规则，其实不然。你必须记住一个重要概念：任何限制条件都可以用于任何过滤规则。虽然逻辑上 `check_helo_access` 条件应该是设定给 `smtpd_helo_restrictions`，但其实它也可以被用于 `smptd_sender_restrictions` 或任何其他过滤规则。Postfix 的这种设计，主要是为了让你可以更有灵活性地安排限制条件的顺序。

表 11-1 限制条件与对应的受检信息

限制条件	客户端提供的受检信息
reject_rbl_client reject_rhsbl_client reject_unknown_client	客户端的 IP 地址或主机名称
check_client_access *type:mapname* permit_naked_ip_address reject_invalid_hostname reject_non_fqdn_hostname reject_unknown_hostname	HELO 提供的主机名称
check_helo_access *type:mapname* reject_non_fqdn_sender reject_rhsbl_sender reject_unknown_sender_domain	MAIL FROM 提供的寄件人邮件地址
check_sender_access *type:mapname* permit_auth_destination	RCPT TO 提供的收件人邮件地址

表 11-1：限制条件与对应的受检信息（续）

限制条件	客户端提供的受检信息
permit_mx_backup	
reject_non_fqdn_recipient	
reject_unauth_destination	
reject_unknown_recipient_domain	
check_recipient_access *type:mapname*	
reject_unauth_pipelining	DATA命令
permit	无条件批准
reject	无条件拒绝
defer	无条件延迟
warn_if_reject	将原本的REJECT动作改成WARN（需放在其他条件之前，不能单独使用）
reject_unauth_pipelining	禁止非授权客户端使用 PIPELINING命令

从形式上看，表 11-1所列的限制条件大致可分成三类。第一类是内置条件，它们的名称以 permit_和 reject_为开头，不需要额外自变量。第二类为自定义条件，它们的名称类似 check_*_access，需要一个 *type:mapname* 自变量，其中 *mapname*是访问表（access table）的名称。在技术上，访问表与一般的 Postfix 查询表没两样，同样都是由一系列的 key-value 组合而成；在用途上，访问表供管理员定义自己的过滤规则：索引键描述客户端的匹配条件，对应值描述如何处理符合条件的客户端。关于访问表，留待本章“定义限制条件”的“访问表”小节再一并讨论。第三类为通用条件，它们没有额外自变量，也不检查任何信息，用于直接改变邮件的处理流程，或是改变其他限制条件的作用。

内置限制条件的工作流程

表 11-1 中的每一种内置限制条件（reject_和 permit_系列），分别检查不同的客户端信息，Postfix依据检查结果决定如何处理邮件。处理方式包括： OK、REJECT 以及 DUNNO。当你在同一个过滤规则中列出多个限制条件时， Postfix依照它们的排列顺序来进行检查过程。检查过程中，如果某条件的检查结果为 REJECT，则 Postfix 会立刻拒收邮件；如果检查结果为 OK，则略过同一过滤规则中的其余条件，继续检查下一个过滤规则，直到所有规则都检查完毕或是出现REJECT结果为止。请注意，按照上述流程，即使邮件在某条件的检查结果是 OK，该邮件仍有机会被另一个过滤规则的条件予以 REJECT。总而言之，任何条件都可以拒收邮件，但邮件必须通过所有过滤规则才会被收下。如果邮件通过一整组过滤规则都没有得到明确结果（全部都是 DUNNO），Postfix 的默认行为是收下邮件。

当过滤规则的检查结果为REJECT时，Postfix的默认行为并不是真的当场拒收，除非客户端已经送出 RCPT TO 命令。换言之，即使 Postfix 在客户端送出 HELO 命令时就决定拒收，它也会先响应 OK ,等到客户端送出 RCPT TO 命令之后才会响应REJECT。Postfix这样的默认行为的设计，主要是顾虑到某些 SMTP client 在对话过程中不会检查服务器端的响应码，而盲目地送出信息。对于这种情况，如果在第一时间就送出 REJECT 响应码，结果只是使得连接时间比原本应有的时间更长，而且会在日志文件里留下好几笔警告记录。将 REJECT 时间延到 RCPT TO之后的另一个好处是 Postfix 可获得足够的客户端信息.，而日志文件留下的记录也比较完整。如果你想改变延迟 REJECT 时机的行为，让 REJECT 动作在第一时间发生，你可将 *main.cf* 的 smtpd_delay_reject 设定成 no：

```
smtpd_delay_reject = no
```

如果你能完全掌握控制 Postfix server 的周围环境，而且事先确定所有 SMTP client 都遵守规则，则可容许 Postfix在第一时间拒收问题邮件；否则，默认行为还是比较理想的。

测试新条件

反垃圾邮件是一项长期的工作，垃圾邮件发送者的花招层出不穷，管理员也需要定期修正或增加新的限制条件。然而，新条件的效果很难完全事先预测，如果没经过实际测试，很难知道新条件是否会误判合法邮件。问题是，如果要等到真的发生误判才能知道效果，这代价未免也太大了。为此， Postfix提供了一个参数，让你用来测试新限制条件：

```
soft_bounce = yes
```

在 soft_bounce 模式下，硬性拒绝码（5xx）会被改成软性拒绝码（4xx）。对 SMTP client而言，5xx 代表服务器端彻底拒绝请求，即使下次再尝试寄出同一封邮件，服务器端也不会接受。所以，收到 5xx 响应码的客户端不会继续尝试同样的动作。另一方面，4xx 代表服务器端只是暂时拒绝请求，客户端下次仍有机会成功寄出邮件，所以 SMTP client 会保留邮件以等待下次寄送时机。利用 soft_bounce 模式来测试新条件，即使发生误判，只要我们能实时更正，被误判的 SMTP client就有机会寄出它的信。

每增加一条你没把握的新限制条件，都可以打开 soft_bounce 模式，然后观察日志文件，看看是否发生误判，并调整条件设计，等一切都妥当之后，再关掉 soft bounce模式。

另一个有用的条件测试工具是warn_if_reject修饰符。将这个修饰符放在限制条件之前，该条件原本的 REJECT动作都会被改成 WARN。也就是说，SMTP client 不会被挡在门外，但是 Postfix会在日志文件里留下一笔警告记录，表示该条件发生作用了。如果你不确定新条件在实际环境下会造成什么后果，但是又不喜欢soft_bounce模式那种全

面性的影响，你可以用 warn_if_reject 来测试新条件，等到确定新条件的效果符合预期之后，再拿掉 warn_if_reject修饰符。举例来说：

```
smtpd_recipient_restrictions =
        permit_mynetworks
        reject_unauth_destination
        warn_if_reject reject_invalid_hostname
        reject_unknown_recipient_domain
        reject_non_fqdn_recipient
```

在此例中，如果客户端使用 HELO 命令送出了一个无效的主机名称， Postfix 仍然会容许对方送出邮件（如果没被其他条件挡掉的话），但是会在日志文件留下一笔警告记录。

一个简单实例

在继续深入研究如何定义限制条件之前，让我们先看一个简单例子：

```
smtpd_recipient_restrictions =
     permit_mynetworks
     reject_unauth_destination
     reject_invalid_hostname
     reject_unknown_sender_domain
```

这个例子对默认配置扩充了两个额外限制条件： reject_unauth_destination 以及 reject_invalid_hostname。当客户端建立连接时，若它来自 mynetworks 或 mynetworks_style 定义的网络，则 permit_mynetworks 返回 OK，整个 smtpd_recipient_restrictions 过滤规则也就跟着结束，不再检查其他条件；如果客户端来自外界网络，permit_mynetworks 不会返回 OK，也不会返回 REJECT，而是返回 DUNNO，于是 Postfix 接着检查 reject_unauth_destination。

如果邮件的收件人不在 Postfix 所辖的网域（由 mydestination 定义），reject_unauth_destination返回 REJECT 否则返回 DUNNO。假设这次的结果是 DUNNO，Postfix 会接着检查 reject_invalid_hostname,如果客户端使用 HELO命令提供的主机名称是无效的，这项检查的结果是 REJECT 如果有效,则为 DUNNO。最后，Postfix 检查 reject_unknown_sender_domain ，如果客户端在 MAIL FROM 提供的邮件地址的网域部分是无效的（用 DNS 查不出来），则检查结果会是 REJECT。如果邮件能顺利通过上述四项限制条件而不发生任何 REJECT，Postfix 就会收下邮件，并交给适当的 MDA 处理。

定义限制条件

依据检查对象的不同，限制条件可分成六大类型，分述如下：

访问表

名称为 check_*_access 形式的限制条件，依据访问表所列的 IP 地址、主机名称或邮件地址（因参数而异）来决定 Postfix 应该采取拒收还是接收。

客户端参数

有些限制条件是依据配置文件里的一般信息而非访问表来检查客户端的资格的。例如，先前的 permit_mynetworks 条件就是检查客户端是否来自 mynetworks 或 mynetworks_style 定义的网络。

严格语法检查

某些限制条件要求 Postfix 非常严格地贯彻 SMTP 标准。由于垃圾邮件发送者经常刻意违反 SMTP 标准，所以这类条件可挡掉大量的垃圾邮件。

DNS 检查

利用 DNS 系统提供的信息来审查客户端是否诚实。垃圾邮件发送者最常出没在那些没有妥善设定好 DNS 的网络，适当运用这类限制条件，确实可以大量减少垃圾邮件。然而有许多合法邮件也是来自 DNS 没有设好的网络，所以误判的机会也很高。除非你对反垃圾邮件抱非常积极的态度，否则不应该过度倚赖 DNS 检查。

实时黑名单

实时黑名单（RBL）是一种依附在 DNS 系统下的网络服务，供 MTA 实时查询客户端是否来自恶名昭彰的垃圾源。虽然 Postfix 也提供 RBL client 支持，但是这种服务仍有争议性。

通用限制条件

无条件拒绝（reject）或收下邮件（permit），或是改变其他条件的行为（warn_if_reject）。这类条件通常出现在整组过滤规则的最后，作为该组规则的默认行为。

接下来的六个小节分别详细说明上述六种限制条件。

访问表

任何涉及客户端资格检查的限制条件，都需要你提供一个访问表（access map）。访问表也是 Postfix 查询表的一种（关于查询表，请参阅第四章），同样都是由一系列 key-value 组成，只不过访问表的索引键是客户端标识信息（IP 地址、邮件地址、主机名称等），而对应值是处理动作（OK 或 REJECT）。

check_client_access *maptype:mapname*

check_client_access 指向一个含有 IP 地址、网络地址、主机名称、从属网域名称的访问表。设定此限制条件时，Postfix 先从 DNS 系统反查出客户端 IP 地址的完

整主机名称（PTR记录），并自己分析出主机本名与从属网域，然后以这些信息与访问表中的每一个索引键比对，如果发现相符的索引键，则采取对应值所指定的处理动作。

check_helo_access *maptype:mapname*

check_helo_access所指的访问表含有主机名称与从属网域，用于比对客户端在HELO命令中显示的主机名称。如果发现相符的索引键，则Postfix执行对应值所指定的处理动作。

check_recipient_access *maptype:mapname*

check_recipient_access 指向一个含有邮件地址、网域名称、人名的访问表，用于比对客户端在 RCPT TO命令中提供的收件地址。如果发现相符的索引键，则Postfix执行对应值所指定的处理动作。

check_sender_access *maptype:mapname*

check_sender_access指向一个含有邮件地址、网域名称、人名的访问表，用于比对客户端在MAIL FROM命令中提供的寄件人邮件地址。如果发现相符的索引键，则 Postfix 执行对应值所指定的处理动作。

check_sender_access和check_recipient_access这两种限制条件都是用来检查客户端提供的邮件地址。这两者所使用的访问表，可包含完整的邮件地址（例如 *user@example.com*），也可能只有网域名称（例如 *example.com*）或是只有人名部分（例如 *user@*）。对于完整邮件地址，必须完全符合才算条件成立；如果只有网域名称部分，则该网域与所有子网域的邮件地址相符都算（例如 *user1@example.com*、*user2@host.example.com* 都算符合“*example.com*”条件）；如果只有人名部分，则不管任何网域，只要人名相符就算条件成立（例如 *user@abc.com*、*user@xyz.com.tw* 都符合“*user@*”条件）。

check_client_access和check_helo_access 两者要检查的数据都是IP地址与主机名称，访问表的索引键可以是数字形式的 IP地址（例如192.168.143.23），也可以是网络地址（10、10.12 或 10.12.154）。

访问表的对应值是发现相符的索引键之后， Postfix应该采取的动作。可能的动作如下：

OK

通过当前过滤规则的检查，Postfix 继续检查下一组过滤规则。

REJECT

REJECT message-text

拒收邮件。你可以另外加注一段简短信息，说明拒绝服务的理由，这段信息会连同拒绝码一起返回给客户端，并且被记录在 Postfix的日志文件里。access_map_

reject_code参数定义所有check_*_access限制条件的默认拒绝码，maps_rbl_reject_code参数定义reject_maps_rbl的默认拒绝码，两者的默认值都是554。

DUNNO

停止检查当前的访问表，假装没找到符合的项目。Postfix会继续检查下一个限制条件。

FILTER transport:nexthop

将邮件转交给指定的内容过滤器。你必须像设定传输表（transport table）那样，指定一个传输方法（transport）与其下一步（nexthop）。

HOLD

HOLD message-text

将邮件放在保留队列（hold queue）。如果你加注一段简短信息，则Postfix会将这段信息记录在日志文件里；否则只记录一般信息。被扣留的邮件会一直留在队列里，直到有人删除它或是将它释放给适当的MDA处理。

DISCARD

DISCARD message-text

要求Postfix收下邮件后立刻丢弃，但是让客户端误以为它已成功送出邮件。如果你加注一段简短信息，则Postfix会将这段信息记录在日志文件里；否则只记录一般信息。别轻易尝试DISCARD动作，除非你已经谨慎考虑过可能的后果。暗自丢掉邮件的行为，有违邮件系统的设计初衷。虽然丢掉垃圾邮件是理所当然的动作，但是万一发生误判，势必无法挽回已经消失的合法邮件，而且有损用户对于Internet E-mail的信任。

4xx message-text

返回指定的拒绝码与信息给客户端，表示暂时拒收邮件。400～499范围内的拒绝码代表服务器端发生暂时性问题，客户端应该先保留邮件，下次再尝试传送。

5xx message-text

返回指定的拒绝码与信息给客户端，表示彻底拒收邮件。500～599范围内的拒绝码代表服务器端不管怎样都不会收下邮件，客户端应该发出退信通知给原寄件人。

访问表的索引键也可以是“正则表达式”（regular express）。一般而言，使用正则表达式来描述访问表是一种浪费，因为Postfix已经有分析邮件地址、网域名称、IP地址的能力，所以正则表达式带来的好处并不多。但如果要检查邮件的内容（标题或正文），正则表达式就非常好用。关于邮件内容的检查，请参阅本章的“内容检查”。

让我们使用一些访问表来扩充先前的范例：

```
smtpd_client_restrictions =
     check_client_access hash:/etc/postfix/client_access
smtpd_sender_restrictions =
     check_sender_access hash:/etc/postfix/sender_access
smtpd_recipient_restrictions =
     permit_mynetworks
     reject_unauth_destination
     reject_invalid_hostname
     reject_unknown_sender
```

应该使用 4xx 或 5xx 来阻挡垃圾邮件？

阻挡垃圾邮件时，有两种拒绝码可以选择。4xx 范围内的拒绝码代表其是暂时性问题，客户端下次还有成功寄出邮件的机会；5xx 范围内的拒绝码代表永久性错误，要求客户端不必再试。

乍看之下，5xx 显然是阻挡垃圾邮件的优先选择，因为我们要求垃圾邮件发送者彻底死心。然而，4xx 其实也有优点，因为假设发生了误判事件，而你在查阅日志文件时发现了这件事，这时候还有补救机会，只要能实时修正导致误判的过滤规则，就还有机会收到该封被误判的合法邮件。但如果使用 4xx 阻挡垃圾邮件，而你的网域里还有其他 MX 备用服务器，假设备用系统没挡掉垃圾邮件，那么，你的暂时性拒绝动作，将导致 MX 备用系统的队列里充斥着大量寄不出去的垃圾邮件。

在填写你的访问表时，你应该依据被阻挡的对象、原因以及你的信心来选择适当的拒绝码。但无论如何，切记一件事：垃圾邮件发送者不会理会你的拒绝码，因为他们本来就是不守规矩的人，期待他们遵守没有法律约束力的 SMTP 协议，无异于缘木求鱼。换言之，不要天真的以为一切都在你的掌控之下。

现在增加了两个要检查访问表的限制条件，所以必须另外准备两个访问表文件，一个是 *client_access* 文件，其内容如下：

```
10.157                  REJECT
192.168.76.23           REJECT
currentmail.com         REJECT
```

另一个文件是 *sender access*，其内容看起来应该类似这样：

```
hardsell@example.com                REJECT
marketing@                           REJECT
specials.digital-letter.com          REJECT
```

准备好这两个文件之后，记得使用 *postmap* 将它们转换成适合 Postfix 查询的数据库格式：

```
# postmap /etc/postfix/client_access
# postmap /etc/postfix/sender_access
```

以后每次修改访问表时，都要重复一次上述动作。

其他客户端条件限制

表 11-1所列的内置限制条件，其中有部分是依据“ Postfix 本身的配置信息 ”与“ 客户端提供的信息 ”的比较结果来决定处理动作的，这类条件不需要访问表。

permit_auth_destination

如果收件地址位于 Postfix 的辖域，则批准请求。所谓“ Postfix 的辖域 ”，包括 mydestination、inet_interfaces、virtual_alias_maps 或 virtual_mailbox_maps以及 relay_domain所列的网域及子网域，而且收件地址不能包含任何“ 发信方指定的递送路径 ”（例如： *user@example.com@example.net*）。如果收件地址不符合 permit_auth_destination 条件，则它返回 DUNNO 而非 REJECT，所以 Postfix 可继续检查后续的限制条件。

permit_mynetworks

如果客户端的 IP 地址位于 mynetworks 参数所列的任何地址范围内，则批准请求。我们经常使用此条件将本地客户端排除在 UBE过滤规则之外，让他们可无条件使用 Postfix server 来转发邮件。

reject_unauth_destination

如果收件地址不位于 Postfix 的辖域，则拒绝请求。 Postfix 的“ 辖域 ”包括 mydestination、inet_interfaces、virtual_alias_maps 或 virtual_mailbox_maps 以及 relay_domain所列的网域以及子网域，而且收件地址不能包含任何发信方指定的递送路径（例如： *user@example.com@example.net*）。本条件的拒绝码为 relay_domains_reject_code，其默认值是 554。

严格语法条件

表 11-1所列的内置限制条件，有部分是以“ 客户端在SMTP对话过程提供的信息是否符合标准 ”为判断依据的。这类条件可侦测出大量垃圾邮件，但是发生误判的机会也很高。你应该研究你收到的垃圾邮件与真实邮件之间的差异（或共通处），才会知道哪些条件可以使用（或不可使用）。如果你事先知道哪些寄件人可能被误判，建议你将他们列入白名单里（访问表的 OK项目）。

reject_invalid_hostname

如果客户端在 HELO命令提供的主机名称不是有效的主机名称，则返回 invalid

hostname_reject_code 参数指定的拒绝码（默认值为 501）。大多数合法寄件人都应该会提供有效主机名称。

reject_non_fqdn_hostname

如果客户端在 HELO 命令提供的主机名称不是 RFC 要求的完整形式（FQDN），则返回 non_fqdn_reject_code 参数指定的拒绝码（默认值为 504）。并非所有合法寄件人都会提供 FQDN 完整名称（尤其是 Windows 系统）。

reject_non_fqdn_recipient

如果客户端在 RCPT TO 命令提供的收件地址的网域部分，不是 RFC 要求的完整形式（FQDN），则返回 non_fqdn_reject_code 参数指定的拒绝码（默认值为 504）。大部分合法寄件人提供的收件地址应该都有 FQDN 完整名称。

reject_non_fqdn_sender

如果客户端在 MAIL FROM 命令提供的寄件人邮件地址的网域部分，不是 RFC 要求的完整形式（FQDN），则返回 non_fqdn_reject_code 参数指定的拒绝码（默认值为 504）。

reject_unauth_pipelining

流水线操作（Pipelining）是一种加速处理大宗邮件的技术，其原理是容许客户端一次送出多个 SMTP 命令。协议要求客户端必须先检查服务器端是否支持 pipelining，才可以开始流水线操作，但是有些 MUA 与 MTA 不遵守规定，它们在还没确认服务器端是否同意 pipelining 之前，就开始流水线操作。Postfix 支持流水线操作，而你可用 reject_unauth_pipelining 立刻拒绝那些不遵守规定的 SMTP client。

DNS 限制条件

DNS 限制条件确认客户端所在的网络以及信封上的邮件地址，是否有可查验的 DNS 信息。如果邮件管理员总是能取得有效的 DNS 信息，Internet E-mail 系统的可用度将获得普遍的提升，因为垃圾邮件发送者将难以遁形。不幸地， DNS 系统是网管人员最常疏忽的一环，有许多合法用户所在的网络并没有可兹查验的 DNS 信息，因此，完全倚赖 DNS 来过滤垃圾邮件，反而成了不切实际的想法。你应该研究你所收到的垃圾邮件与真实邮件的本质，看看哪些限制条件既能够阻挡最多的垃圾邮件，而误判机率又不令人担心。如果你事先知道哪些寄件人可能被误判，建议你将他们列入白名单里，以免受到 DNS 限制条件的影响。

reject_unknown_client

当一个 SMTP client 通过 socket 建立连接时，Postfix server 可从 socket 连接知道客户端的 IP 地址。设定 reject_unknown_client 限制条件时，如果 Postfix 从

DNS查不出客户端IP地址的PTR记录（该IP地址在DNS系统登记的主机名称），则会拒绝服务；如果能从DNS查出主机名称，则Postifx还会使用该主机名称再向DNS查出其对应的IP地址，若查出的IP地址不符合socket连接提供的远程IP地址，则Postfix也会拒绝服务。此限制条件的拒绝码定义在unknown_client_reject_code参数，其默认值为450。实际上，reject_unknown_client确实可挡掉很多垃圾邮件，但是挡掉的合法邮件可能更多，因为有很多网络并没有妥善维护他们的DNS数据库，Internet上有一大半IP地址查不出PTR记录，其中包括合法用户所在的网络。

reject_unknown_hostname

如果HELO命令提供的主机名称既没有A记录，也没有MX记录，则拒绝服务并返回unknown_hostname_reject_code参数指定的拒绝码，其默认值为450。实际上，有许多客户端的HELO所提供的主机名称不是完整形式（FQDN），就会被这个条件挡在门外。

reject_unknown_recipient_domain

如果RCPT TO命令提供的收件人地址，其网域部分查不出有效的DNS A或MX记录，则以unknown_address_reject_code参数定义的拒绝码拒绝服务。该参数的默认值为450。

reject_unknown_sender_domain

如果MAIL FROM命令提供的寄件人地址，其网域部分查不出有效的DNS A或MX记录，则以unknown_address_reject_code参数定义的拒绝码拒绝服务。该参数的默认值为450。

实际上，由于MAIL FROM地址是退信通知的收件地址，所以要求客户端提供一个已知的有效网域名称是非常合理的要求。一般合法用户应该都能够提供正确地址，而捏造MAIL FROM地址也是垃圾邮件发送者常用的伎俩之一，所以我们强烈建议你加上此限制条件。

上述四个限制条件都使用unknown_*_reject_code参数来决定拒绝码，而且这些参数的默认值都是450。如果你修改这些默认值，Postfix会改用你指定的拒绝码，但是有一个例外：如果Postfix从DNS系统查不出数据，是因为DNS server暂时性的故障（这种事常有）而非DNS server真的回复查不出数据，则Postfix仍会使用450拒绝码，而非你指定的值。

实时黑名单

实时黑名单(Real-Time Blacklist, RBL）是一种专为抵制垃圾邮件而设计的网络服务，让SMTP server可通过DNS系统实时查询客户端是否为垃圾源。由于这种服务依附于

DNS系统，所以也称为 DNSBL。在使用 DNSBL 服务之前，你必须先挑选一家值得信赖的服务提供者，有些机构提供免费服务，但也有些管理员要求付费。 Postfix提供下列三种关于 DNSBL的限制条件，分别用于支持不同技术的 DNSBL服务。

`reject_rbl_client rblprovider.domain`

将客户端 IP地址（例如 1.2.3.4）颠倒顺序（成为 4.3.2.1），搭配 RBL管理员的网域名称（例如 *dnsbl.example.com* ）构成一个主机名称（成为 *4.3.2.1.dnsbl.example.com*），然后以此主机名称向 DNS系统查询，如果能查出一笔 A记录，表示该 IP地址已被列入黑名单，则 Postfix 会当场拒收邮件。

`reject_rhsbl_client rblprovider.domain`

如果客户端的主机名称在指定的 `rblprovider.domain` 网域下有一笔 A记录，则拒绝服务。

`reject_rhsbl_sender rblprovider.domain`

如果寄件人的邮件地址的网域部分，在指定的 `rblprovider.domain`网域下有一笔 A记录，则拒绝服务。

应不应该使用 DNSBL服务，是一个见仁见智的问题。虽然有些 DNSBL管理员勤于维护他们的 DNS数据库，但由于宽带接入服务的兴起，使得很多垃圾邮件发送者有了新的藏身之处，而且更加捉摸不定，使得 RBL 抵制技术的效果大打折扣。

通用限制条件

除了针对特定信息的检查条件之外，Postfix还提供了四个可用于任何过滤规则的限制条件，它们适合放在过滤规则的最后，作为该组规则的默认处理政策。

`permit`

批准收下邮件。Postfix 不再继续当前的过滤规则，但是会跳到下一组过滤规则（如果有的话）。

`reject`

无条件拒收。Postfix 不再继续处理任何过滤规则。

`defer`

婉拒请求，客户端被告知稍后再试。

模拟限制条件组合的检查流程

在目前所介绍的范围内，我们可以用一组简单的 HELO 限制条件为例，模拟四种不同情况下的检查流程。假设 `smtpd_helo_restrictions`过滤规则被设定成下列两项条件：

```
smtpd_helo_restrictions =
    check_helo_access hash:/etc/postfix/helo_access
    reject_invalid_hostname
```

其中，*helo_access* 访问表的内容为：

```
greatdeals.example.com      REJECT
oreillynet.com              OK
```

现在，假设有四个客户端分别连接到 Postfix，并送出了四种不同的 HELO 命令，让我们分别分析 Postfix 在这四种状况下的反应：

HELO *example*

第一个条件是 check_helo_access，所以 Postfix 先检查此条件指定的 *helo_access* 访问表。查表结果找不到指定的 *example* 主机名称，所以它接着检查 reject_invalid_hostname 条件。由于 *example* 不是符合标准规定的完整主机名称，所以 Postfix 当场拒收。

HELO *greatdeals.example.com*

Postfix 先检查 check_helo_access 条件指定的 *helo_access* 访问表，结果找到 *greatdeals.example.com* 的记录，而其对应动作是 REJECT，因此，Postfix 当场拒绝收信。

HELO *oreillynet.com*

Postfix 先检查 check_helo_access 条件指定的 *helo_access* 访问表，结果找到 *oreillynet.com* 的对应动作是 OK，所以 Postfix 停止处理 smtpd_helo_restrictions 的其余条件，接着检查 smtpd_sender_restrictions 过滤规则（如果有的话）。

HELO *mail.ora.com*

Postfix 先检查 check_helo_access 条件指定的 *helo_access* 访问表，结果找不到 *mail.ora.com*，所以接着检查 reject_invalid_hostname 条件。由于 *mail.ora.com* 是符合标准规定的主机名称，所以 Postfix 继续检查 smtpd_sender_restrictions 过滤规则（如果有的话）。

SMTP 语法规范参数

SMTP 是相当宽松的协议，有些 SMTP client/server 的对话过程甚至比协议本身的要求更宽松。为此，Postfix 提供两个参数，让你自己决定是否要求严格遵守 SMTP 协议的对话过程与语法。第一个参数是 smtpd_helo_required，它决定是否要求 SMTP client 以 HELO 或 EHLO 动词为开场白。按照 SMTP RFC 的规定，SMTP clients 连接到 server 后的第一句话必须是 HELO 或 EHLO。

默认情况下，Postfix对客户端抱持宽容态度，容许客户端不严格遵守 SMTP协议。但如果你设定 smtpd_helo_required = yes，而客户端见面不打招呼，则 Postfix将拒绝服务。

SMTP RFC 也明确规范了信封地址的格式。正常情况下， Postfix几乎接受任何有意义的信封地址，但是，如果你设定了 strict_rfc821_envelopes = yes ，则没有提供正确格式地址的客户端将被 Postfix挡在门外。

实际上，要求对方必须提供 HELO/EHLO或许是个好主意，因为大部分的客户端至少都会遵守 SMTP协议的基本对话过程。但并非所有客户端都提供符合标准格式的邮件地址，如果要求太严格，可能会错失一些合法邮件。

内容检查

直接让 Postfix将垃圾邮件过滤的最后机会，是检查邮件内容本身。 Postfix 提供了四个检查邮件内容的参数:

header_checks 检查标题。

mime_header checks检查标题的 MIME 相关字段。

nested_header_checks 检查夹带附件的标题。

body_checks 检查邮件的正文。

内容检查的影响是全面性的，要么全部邮件都必须受检，否则就全部都不检查。也就是说，该阶段已经没有机会让特定寄件人或收件人绕过检查。

内容检查可能是阻挡垃圾邮件最有效的手段，但是如果你不熟悉正则表达式或者没有时间收集垃圾邮件样本来分析，你应该考虑使用专门检测垃圾邮件的外部过滤程序，例如 Spamassassin（*http://spamassassin.org*）。 第十四章示范了如何设定 Postfix，使其可以搭配外部的内容过滤程序。

上述四个参数都需要你提供一个查询表，其中含有“模式”（pattern）与“动作”（action）。这些模式（即“正则表达式”）是用来比对邮件内容里的字符串，如果发现符合模式的字符串，Postfix就执行对应的动作。默认的模式匹配原则是不区分大小写字母。至于如何在 Postfix 查询表里使用正则表达式，请参阅第四章的说明。

设定内容检查参数

在默认情况下，如果你没有设定 mime header checks和 nested_header_checks参

数，它们将与 header_checks 共享同一个查询表。如果你想将其区分开来，则必须分别提供不同的查询表给它们。设定检查参数时，必须先注明正则表达式的语法格式（regexp 代表 Postfix 的标准正则表达式语法， pcre 代表 Perl 兼容的语法）以及模式表的完整路径。例如：

```
header_checks = regexp:/etc/postfix/header_checks
body_checks = regexp:/etc/postfix/body_checks
```

模式表的索引键是描述比对特征的正则表达式，此正则表达式本身必须放在一对分隔符（通常是 /）之间，例如：

```
/match pattern/              REJECT
```

典型的 header_checks 模式表看起来类似下面这样：

```
/free mortgage quote/        REJECT
/repair your credit/         REJECT My credit is very good.
/take advantage now/         REJECT
```

如果标题里出现任何上述字符串（机会最大的字段是 Subject:），则予以拒收。Postfix 会将拒收事件连同你提供的信息（如果有的话）记录在日志文件里，客户端也会见到你提供的信息。因此，如果有任何人寄信给你，表示有能力恢复你的信用记录，他将被拒绝，并得到“ My credit is very good.” （我的信用很好）这样的回复。

内容检查的响应动作

模式表的对应值，是符合比对条件后的响应动作以及一段可有可无的文字信息。如果比对条件成立，你指定的信息不仅会被传给客户端，也会被记录在日志文件；如果你没提供信息，则 Postfix 使用默认信息。以下是可能的各种响应动作：

REJECT message-text

如果模式匹配成功，则拒收邮件，并且将 *message-text* 传给客户端。

WARN message-text

模拟拒收动作。不会真的拒收，只将 *message-text* 记录在日志文件中，并停止比对后续模式。当你想测试正则表达式的匹配效果时，可利用 WARN 来避免误判合法邮件。

IGNORE

删除符合模式的标题字段或整行文字。这项动作常用来删除某些敏感信息。比方说，如果你不希望寄到外地网络的邮件包含内部网络的信息，就可利用这种方法。不过，使用这项功能时请务必小心，因为大部分的标题字段都有其作用，有些更是 SMTP 协议标准的硬性规定。此外，保留完整标题有助于追查电子邮件的问题。

HOLD message-text

将整封邮件放在保留队列（hold queue）。关于保留队列，请参阅第五章。

DISCARD message

要求 Postfix 假装接收邮件，但其实偷偷丢掉。有时候，专发垃圾邮件的软件根本不理睬响应码，即使你用 5xx 予以回拒，它还是照送不误，这会导致联机时间比直接收下信息所需的时间更长。 DISCARD 就是专门为应付这种技术而设计的机制。此外，DISCARD也能有效解决迂回攻击的问题（参阅先前在“应该使用 4xx 或 5xx 来阻挡垃圾邮件？”的说明）。而且，如果有无辜用户的邮件地址被用来当成垃圾邮件发信者地址，也可以用 DISCARD来没收邮件，以免无辜受害者收到退信通知。

FILTER transport:nexthop

先将邮件排入队列，然后转交给指定的外部过滤程序。至于 Postfix 如何与外部程序联接，请参阅第十四章的说明。

模式表不同于访问表，你不能使用特殊响应码，也不能自定义限制条件。

模式匹配

执行标题检查时，标题里的每一个字段都要依序与模式表里的正则表达式匹配。如果字段内容跨越多行，在比对之前，它们会先被合并成一行（译注 3）。Postfix先从模式表里的第一个正则表达式开始比对，只要发现字段符合某正则表达式，整个比对过程就立刻结束，并执行该正则表达式对应的动作。只有在所有字段都不符合所有模式的情况下，整个标题才算通过检查。

执行正文检查时，`body_checks` 指定的模式表里的每个正则表达式，依序与正文里的每一文字行比对，每次只比对一行。如果发现符合模式的字符串，整个比对过程就立刻结束，执行此模式所对应的动作。

如果要比对的文字行（不管是在标题还是正文）超过长度上限， Postfix会将它们拆成符合长度限制的段落，分段检查。文字行的字符数上限由 `line_length_limit`参数决定，其默认值为 2 048 。如果标题的总长度超过 `header_size_limit`（默认值为 100 K），Postfix 也是以同样原则分段处理。最后，如果正文总长度超过 `body_checks_size_limit`（默认值为 50 KB），Postfix不会检查超过限制的部分。这项限制相当有用，因为可避免 Postfix去扫描整个文件。

译注 3：被并成一行的字段，原本的 CR、LF、Tab 等字符也会成为字段值的一部分，也就是说，你的模式必须将这些特殊字符也考虑进去。

有些管理员运用 header_checks 来进行简单的病毒过滤。例如，使用下列正则表达式可挡掉夹带危险文件的邮件：

```
/name ?="?.*\.(bat|scr|com|dll|exe|hta|pif|vbs)"?/     REJECT
```

如果你的 Postfix 系统要服务许多 Windows 系统的计算机，上述模式或许能帮你的用户群减少不少困扰，但同时也阻断了“交流正常程序文件”的机会。请注意，这个模式的防毒效果有限，因为我们还漏掉了一些扩展名，不足以阻挡所有 Windows 可执行文件，况且有许多 PC 客户端不用扩展名就可以直接运行文件。

下面是一个典型的 *body_checks* 模式文件的内容：

```
/increase your sales by/           REJECT
/in compliance (with|of) strict/   REJECT
/lowest rates.*\!/                 REJECT
/[:alpha:]<!--.*-->[:alpha:]/      REJECT Suspicious embedded HTML comments
```

前两个模式很简单，不再解释。第三个模式挑出任何含有“lowest rates”字样且其后跟着任何文字（.*）以及一个感叹号（!）的字符串，例如“We have our lowest rates in 40 years!”）。最后一个模式检查是否有 HTML 注释嵌在字句中间，例如“VIA<!--ooxx-->GRA”。垃圾邮件发送者常用这种技巧试图通过过滤程序，但是，这也成为垃圾邮件的绝佳特征，因为一般的正常邮件多半不包含 HTML 注释，就算有，也不会刻意夹在字句中间。

使用 postmap 工具可以测试新写好的正则表达式。假设 *msg.txt* 是一个垃圾邮件样本文件，我们可用下列方式将它导入 postmap：

```
$ postmap -q - regexp:/etc/postfix/body_checks < msg.txt
opportunity. increase your sales by 500%. Consider    REJECT
```

postmap 会显示出符合模式的字符串，以及对应该模式的动作。

每个站点面临的垃圾邮件源都不太一样，你应当研究你收到的垃圾邮件，根据你自己的研究结果来写出适当的过滤模式。编写正则表达式时一定要相当谨慎，如果设计不当，可能会严重降低服务器的效率。另一个潜在的问题在于没有任何邮件可以绕过内容检查，即使邮件已通过所有 smtpd_*_restrictions 过滤规则的检验，甚至已被纳入白名单，最后仍有可能被 header_checks 与 body_checks 挡在门外。

在你设计判别垃圾邮件的过滤规则时，请留心用户群之间的不同需求与心态。有些人对垃圾邮件深恶痛绝，甚至愿意承担较高的误判风险；有些人则宁愿多收一些垃圾邮件，也不愿意错失任何真实邮件。被你挡掉的化妆品广告邮件，有可能是某位女同事订阅的电子报。如果无法兼顾所有人的要求，且必须分别为不同用户设定不同的过滤规则时，

则最好不要在 MTA 做这件事，应该考虑使用特殊的 MDA，例如 procmail、maildrop，或任何能依用户类别来选择过滤规则的软件。

倘若你真的希望在 Postfix 中解决不同用户的争议，则 Postfix 提供了分级机制，可针对不同收件人使用不同的过滤规则。

自定义过滤条件组合

为了满足不同层次的观众，政府设计了电影分级制度；同样地，为了满足不同用户的需求，Postfix 也提供了分级制度，让你可依据用户的身份来选择适当的过滤条件组合。Postfix 的分级机制称为“规范等级”（restriction class），这是非常强大的工具，让你能更灵活运用 Postfix 的垃圾邮件过滤条件。如果你的用户需要不同松紧程度的过滤条件组合，或是有一两位用户需要与众不同的过滤规则，投资时间研究如何设定规范等级绝对是值得的。

具体做法如下：

1. 定义多组“规范等级”（restriction class），每个等级都有自己的专属名称，不同的等级各由松紧不等的限制条件（参阅表 11-1）组合而成。
2. 制作一个访问表，索引键是用户的标识信息（通常是邮件地址），对应值是该用户适用的限制等级的名称。
3. 在一般的 smtp_*_restrictions 过滤规则中，加上一条检查访问表的条件。任何 check_*_access 条件都可以用来检查访问表，换言之，你可以依据客户端（check_client_access）、寄件人（check_sender_access）或收件人（check_recipient_access）来执行分级过滤。

规范等级实例

为了举例说明，假设我们有两组用户：一组是电信警察，以打击垃圾邮件发送者为职责，研究垃圾邮件是他们的专业；另一组人是八卦部队，如果让他们收到垃圾邮件，天下只会更乱而已。

很显然地，这两组人不能共享相同的过滤规则，否则一定会出事。为了满足这两种极端的要求，我们拟定了两级规范，分别命名为“spamlover”与“spamhater”。所有规范等级的名称都必须列在 smtpd_restriction_classes 参数，像这样：

```
smtpd_restriction_classes = spamlover, spamhater
```

接下来，我们当初怎样设定一般的 smtpd_*_restrictions 过滤规则，现在就可以怎样定义我们的规范等级。以下是 spamhater 的定义，其限制条件相当严格：

```
spamhater =
        reject_invalid_hostname
        reject_non_fqdn_hostname
        reject_unknown_sender_domain
        reject_rbl_client nospam.example.com
```

下面是 spamlover 的定义，只有一个简单的 permit（无条件批准）：

```
spamlover = permit
```

当然，现实环境中不太可能遇到如此极端的情况，你应该视实际情况调整规范等级的定义，增加或减少某些限制条件。

现在，我们已经完成新规范等级的声明与定义，下一步，我们要制作一个访问表，让 Postfix 知道哪些人适用哪种规范等级。由于我们的规范等级是针对不同用户（收件人）而设计的，所以访问表的索引键是收件人的邮件地址，而对应值是该用户适用的规范等级。假设这个访问表的名称是 *per_user_ube*，它的内容看起来应该类似这样：

```
#
# per_user_ube
#
abelard@example.com    spamhater
heloise@example.com    spamlover
```

最后，要求 Postfix 在审核收件人限制时，检查你指定的访问表：

```
smtpd_recipient_restrictions =
    permit_mynetworks
    reject_unauth_destination
    check_recipient_access hash:/etc/postfix/per_user_ube
```

一切就绪之后，每当外界有人写信给 *abelard@example.com*，Postfix 先执行一遍正常的默认过滤规则，在遇到 check_recipient_access 时，它会检查指定的收件人访问表，并查出 *abelard@example.com* 适用的规范等级为 spamhater，然后执行 spamhater 定义的限制条件。如果 spamhater 下的任何条件都返回 REJECT，Postfix 就拒收邮件 否则就将邮件交给适当的 MDA 处理。外界写给 *heloise@example.com* 的邮件也是依照同样的过程来处理，如果 Postfix 所执行的规范等级为 spamlover，会无条件收下任何邮件。

反垃圾邮件实例

至此，我们已经涵盖 Postfix 的所有垃圾邮件的抵制机制，现在可以举一个具体实例来总结本章的讨论。由于每个站点面对的网络环境与实际需求都不一样，所以我们不可能设

计一个面面俱到配置文件，也无法给你太多具体建议。例 11-2可给你作为一个起点，你必须随实际情况来调整它：

例 11-2：反垃圾邮件的 main.cf 配置文件样本

```
smtpd_restriction_classes =
        spamlover
        spamhater

spamhater =
        reject_invalid_hostname
        reject_non_fqdn_hostname
        reject_unknown_sender_domain
        reject_rbl_client nospam.example.com

spamlover = permit

smtpd_helo_required = yes
smtpd_client_restrictions =
        check_client_access hash:/etc/postfix/client_access
smtpd_helo_restrictions =
        reject_invalid_hostname
        check_helo_access hash:/etc/postfix/helo_access
smtpd_sender_restrictions =
        reject_non_fqdn_sender
        reject_unknown_sender_domain
        check_sender_access hash:/etc/postfix/sender_access
smtpd_recipient_restrictions =
        permit_mynetworks
        reject_unauth_destination
        reject_non_fqdn_recipient
        reject_unknown_recipient_domain
smtpd_data_restrictions =
        reject_unauth_pipelining
header_checks = /etc/postfix/header_checks
body_checks = /etc/postfix/body_checks
```

这个例子需要好几个访问表，你应该分析实际收到的垃圾邮件，收集垃圾邮件发送者的IP或邮件地址，创建你自己的访问表。不过，就经验来说，`check_helo_access`和`check_sender_access`的效果有限，阻挡不了太多的垃圾邮件。基本上，垃圾邮件发送者有无限多的邮件地址与主机名称可以使用，每次你更新了访问表之后，他们又从别的地方冒出来，让你疲于奔命。况且，主机名称与发信者地址太容易捏造了，而且他们时常假冒来自拥有大群合法用户的大型网站（如 *hotmail.com*、*aol.com* 等），使你防不胜防。

虽然挡不住精明的垃圾邮件发送者，但是阻挡垃圾邮件发送者的效果还挺不错。有些家伙不懂得变化，总是千篇一律地使用相同信息（还不一定是捏造的！），对付这种连干坏事也没有脑筋的垃圾邮件发送者（为数不少！），访问表是最有效的武器。

有些在线营销服务发送信息真实的广告信，而且也遵守规定地提供真实的选退机制，让你决定是否愿意继续收到广告信。然而，如果你不相信那些听都没听过的公司的选退机制，你可用 MAIL FROM 或 HELO 的信息来阻挡他们的广告信，而不要冒着让他们有机会证实你的邮件地址的风险。当然，有些站点恶名远扬，你完全不想再见到他们，不管他们是否诚实、提供选退机制，你都可以直接挡掉它们的垃圾邮件。

此外，有很多垃圾邮件假冒它们来自某些小国家。如果你确定不可能会收到来自 Republic of Maldives （马尔代夫共和国）的合法邮件，你可以将该国的顶层网域名称填入访问表，一举阻断所有假冒来自该地址的垃圾邮件。然而，如果你的邮件系统服务很多用户，你或许不应该如此武断，将你个人的独断意识强加到每位用户身上。焉知你的用户没有住在马尔代夫的亲戚？或是特别喜欢到该国度假？

第十二章

SASL 身份验证

基本的 SMTP协议没有验证用户身份的能力。虽然信封上的寄件人地址已经隐含了发信者的身份，然而，由于信封地址实在太容易假造，所以不能当成身份凭据。为了判断客户端是否有权使用转发服务（relay），服务器端必须确认客户端（寄件人）是否当真是对方所自称的那个人。在不能以寄件人地址为身份证书的前提下， SMTP势必需要其他补充机制，才能验证客户端的身份。本章将介绍如何使用 *Simple Authentication and Security Layer*（SASL）来决定谁有权使用转发服务或是辨认谁在使用你的服务器。

从 Postfix 的角度看，它需要扮演两种角色：当它身为 SMTP server时，需要能够验证用户个人的身份（让他们能使用 SMTP server寄出邮件）当它身为 SMTP client 时，它需要能够提出自己的身份证书给其他 MTA检验（以便通过远程 MTA将邮件递送到最终目的地)。因此,我们也会解释如何设定 Postfix，使其能通过其他 MTA 的身份验证。第四章讨论了 mail relay 的一般问题以及其他值得考虑的解决方案。

第四章中的“转发控制”描述了 Postfix 如何认定哪些客户端具备转发服务的使用资格。大多数邮件系统只容许内部网络上的客户端使用转发服务，换言之， IP地址成了客户端身份的识别凭据。然而，并非所有合法用户都具有固定 IP 地址。比方说，带着笔记型计算机出差到远方的同事，他们可能使用邻近 ISP或饭店旅馆提供的临时性 IP地址；或者，有些在家工作的用户，他们位于办公室之外的网络，但是需要透过办公室里的邮件系统来寄信。不管你是否能够事先知道用户的 IP 地址，SASL 都能提供可靠的身份验证。

RFC 2554“SMTP Service Extension for Authentication”制定了如何在基本 SMTP 协议上增加验证功能的机制，此机制使得 SMTP 能使用 SASL协议来验证客户端身份。我们将示范如何使用 Carnegie Mellon 大学开发的 Cyrus SASL 函数库来扩充 Postfix，使其具备 SASL 验证能力。在你安装了 SASL之后，你可能还会想要增添对 TLS的支持（见

第十三章)。TLS(其前身为 SSL(译注 1))通常用于加密 web server/client 之间的对话,但是也可用于保密 mail servers/client 之间的通信内容。由于 SASL 的某些密码交换协议是以明文形式(plaintext)来传递密码,你可以使用 TLS 加密通信数据,避免密码在传递过程中外泄。

由于 Cyrus SASL 以“函数库”的形式存在,要让你的 Postfix 支持 SASL,你必须在编译 Postfix 时就将 Cyrus SASL 函数库链接进去。此外,远程用户也必须设定他们的 MUA,使其在通过你的邮件系统转发邮件时,能送出正确的标识信息。

SASL 概论

有许多客户机/服务器协议(client/server protocol)没有验证能力,SASL 就是用于加强或增加这类协议的一种通用方法。当你设定 SASL 时,你必须决定两件事:一是用于交换“标识信息”(或称“身份证书”)的验证机制(authentication mechanism);一是决定标识信息存储方法的验证架构(authentication framework)。SASL 验证机制规范 client 与 server 之间的应答过程以及传输内容的编码法,SASL 验证架构决定服务器本身如何存储客户端的身份证书以及如何核验客户端提供的密码。图 12-1 描绘了这两种过程。如果客户端能成功通过验证,服务器端就能确定用户的身份,并借此决定用户具有怎样的权限。对 Postfix 而言,所谓的“权限”指的就是转发服务的访问权。你也可以决定通过验证的用户在转发邮件时,是否要使用特定的寄件人地址。

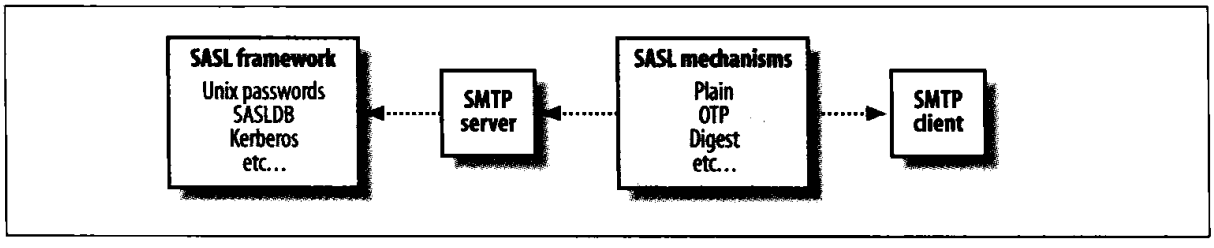

图 12-1: SASL 验证机制与架构

选择适当的验证机制

Cyrus SASL 支持多种验证机制,至于要使用哪一种验证机制,客户端与服务器端双方必须事先取得共识。以下是一些比较常见的机制:

PLAIN

PLAIN 是最简单的机制,但同时也是最危险的机制,因为身份证书(登录名称与密

译注 1: 事实上,TLS 是 SSL 3.1 版。

码）是以 base64 字符串格式（译注 2）通过网络，没有任何加密保护措施。因此，使用 PLAIN 机制时，你可能会想要结合 TLS（参阅第十三章）。

LOGIN

LOGIN 不是其正式支持的机制，但某些旧版的 MUA 使用这种机制，所以 Cyrus SASL 让你可选择其是否要支持 LOGIN 机制。如果你的用户仍在使用这类老掉牙的 MUA，你必须在编译 SASL 函数库时，指定要包含 LOGIN 的支持（参阅附录三）。LOGIN 的证书交换过程很类似于 PLAIN。

OTP

OTP 是一种使用“单次密码”（One-Time Passwords，其前身为 S/Key）的验证机制。此机制不提供任何加密保护，因为没必要 ——每个密码都只能使用一次，每次联机都要改用新密码。SMTPclients 必须要能够产生 OTP 证书。

DIGEST-MD5

使用 DIGEST-MD5 机制时，client 与 server 共享同一个隐性密码（secret password），而且此密码不通过网络传输。验证过程是从服务器端先提出 challenge （质询）开始，客户端使用此 challenge 与隐性密码计算出一个 response （应答）。不同的 challenge，不可能计算出相同的 response；任何拥有 secret password 的一方，都可以用相同的 challenge 算出相同的 response。因此，服务器端只要比较客户端返回的 response 是否与自己算出的 response 相同，就可以知道客户端所拥有的密码是否正确。由于真正的密码并不通过网络，所以不怕网络监测。

KERBEROS

Kerberos 是一种网络型的验证协议。除非你的网络已经使用 Kerberos，否则你应该用不到 KERBEROS 机制；相对地，如果你的网络已经架设了 Kerberos 验证中心，SASL 就能完美地将 SMTP 验证整合进现有的体系。

ANONYMOUS

ANONYMOUS 机制对 SMTP 没有意义，因为 SMTP 验证的用意在于限制转发服务的使用对象，而不是为了形成 Open Relay。SASL 之所以提供这种机制，主要是为了支持其他协议。

当客户端连接到一个支持 SASL 的邮件服务器时，服务器会以优先级列出可用的机制供客户端选择。如果客户端也支持多种机制，则当第一种机制验证失败时，客户端可能会继续尝试第二种机制，直到通过验证或是所有机制都失败为止。如果双方在一开始就无法协调出共同的机制，验证过程就算失败。

一旦双方在使用哪种机制上达成共识，就开始进行验证过程。实际的交互过程随机制而

译注 2：　base64 是一种公开算法，能将 8-bits binary 编成 7-bits ASCII 数据。

定，但通常包含一次或多次应答过程（challenge-response）。验证协议本身也规定了应答内容的编码格式（注 1）。

选择适当的验证架构

SASL 验证架构（SASL authentication framework）可以使用现有的 Unix 系统密码（比方说，*/etc/passwd*、*/etc/shadow* 或 PAM），也可用 SMTP 用户的专用密码文件。如果你的网络上有 Kerberos 之类的中间控制式验证架构，也可以使用。

哪一种验证架构最适合你，取决于你的服务器从何、如何取得证书信息。举例来说，如果 SMTP 与 POP/IMAP 的所有用户的证书数据都是存储在系统密码文件（*/etc/passwd* 或 */etc/shadow*），并通过 PAM 来验证，那么，SASL 就应该透过 PAM 来取得证书数据。另一方面，如果所有 SMTP user 都只有虚账户（不能登录系统），你或许应该将证书数据（登录名称与密码）存放在专用的数据库，并设定 SASL 与 POP/IMAP server 从该数据库取得证书数据。

Postfix 与 SASL

在开始使用 SASL 之前，你应该决定好，要采用哪一种机制与架构的组合，因为你的决定将影响编译、安装、设定的过程。首先，你必须先将 SASL 函数库安装到你的系统上，或是确定 SASL 函数库的安装目录与版本。接着，使用你收集到的 SASL 安装信息来设定 Postfix 的编译选项，然后重新编译 Postfix，使其具备 SASL 验证能力（译注 3）。某些系统平台可能已经预先安装了 SASL 函数库，甚至提供支持 SASL 的 Postfix 包，不过，大部分预先编译好的 Postfix 包都不支持 SASL。因此，如果你使用现成的 Postfix 包，最好先研读相关说明文件，或是按照本章“测试验证配置”一节所说的测试方法，确认你的 Postfix 确实支持 SASL。此外，你也要确认你的 SASL 函数库支持客户端可能使用的每一种验证机制。比方说，若你的用户中还有人使用老版的 Outlook Express，你的 SASL 函数库就必须支持 LOGIN 验证机制。

Cyrus SASL 函数库的研发进度，目前分成两条路线，即 SASL 与 SASLv2。其中，SASL

注 1：请注意，“编码”（encode）不等于“加密”（encryption）。“编码”的目的是为了让通信内容能通过 7-bits 的 SMTP 协议，而“加密”是为了让第三者看不懂通信内容。证书信息是否加密，由机制本身决定。

译注 3：附录三的“Cyrus SASL”小节示范了 Cyrus SASL 本身的编译过程以及 Postfix + Cyrus SASL 的编译过程。

已经逐渐被 SASLv2所取代。在未来， Postfix 可能只支持 SASLv2，所以本章只讨论 SASLv2。此外，Postfix 与 SASL 函数库两者的版本都必须正确，才能组合在一起。

Postfix从 1.1.7-20020331 实验版开始支持 SASLv2 函数库，在这之前的版本，只能使用 SASLv1 。理论上，最新版的 Postfix与最新版的 Cyrus SASLv2 应该可以顺利结合在一起。由于本章不打算讨论 SASLv1，因此后文提到' SASL '时，都是指 SASLv2 函数库。

Postfix 的 SASL 配置

假设你已经安装好 SASL 函数库，而且 Postfix也支持 SASL，并下定决心要使用哪一种验证机制与架构的组合。现在，让我们逐步设定 Postfix ,使其能够使用 SASL 来验证用户的身份。

设定验证架构

对于每个使用 Cyrus SASL 函数库的应用系统，Cyrus SASL各提供一个独立的配置文件。影响 Postfix 的 SASL配置文件是 *smtpd.conf*，此文件通常位于 */usr/local/lib/sasl2/smtpd.conf*。基本上，*smtpd.conf* 至少要指出所要使用的验证架构。我们打算讨论两种最常用的架构：Unix系统密码以及独立的 SASL 专用密码。关于 *smtpd.conf* 支持的其他选项，请参阅 Cyrus 的说明文件。

Unix 系统密码

通常，让 SASL直接使用现有的系统密码来验证用户身份是最方便的。传统的系统密码文件应该是 */etc/passwd*，但是讲究安全性的现代系统则比较可能使用 */etc/shadow*、PAM 或诸如此类的证书数据库。由于这类密码文件只有特权进程才能访问，而 Postfix 却被刻意设计成避开特权身份，所以 Postfix 不能直接访问系统密码文件。

Cyrus 函数库对于这个问题的解决办法，是提供一个特殊的验证服务器程序，称为 `saslauthd` ,它能够代替 Postfix 来取得密码数据。`saslauthd` daemon 本身需要特权身份，不过，由于它是一个独立于 Postfix 之外的进程，而且通常不必与外界进行网络通信，所以安全性的危害已经被降到最低。如果你打算让 SASL使用 Unix系统密码，你必须启动 Cyrus SASL 包随附的 `saslauthd` daemon 。请注意，使用 `saslauthd`来访问 Unix系统密码，表示你只能使用明文密码（plaintext password),因为 `saslauthd`需要实际密码才能进行核验。关于 Postfix 与 SMTP client 之间的加密通信，请参阅第十三章。

要让 SASL知道 Postfix 将通过 `saslauthd` daemon来访问证书数据库，你必须将下列内容加入 *smtpd.conf* 配置文件：

```
pwcheck_method: saslauthd
```

Cyrus SASL包随附的saslauthd应该会被安装在$PATH环境变量所列的某个目录下。你必须先在后台启动saslauthd daemon，Postfix才能使用它来验证客户端。当你启动saslauthd 时，你必须使用 -a选项指定密码系统的类型。最常见的类型包括pam、shadow或getpwent（用于传统的 */etc/passwd*）。举例来说，在一个使用 PAM的系统上，你应该使用下列命令来启动 saslauthd daemon：

```
# saslauthd -a pam
```

关于 saslauthd 实际支持的其他选项，请参阅Cyrus的说明文件。此外，你可能需要为saslauthd写一个启动脚本，使其能在开机时被自动启动，让Postfix总是能够使用它。

SASL 专用密码

如果你不想使用系统密码来验证 SMTP client ，你可以另外建一个无关系统密码的独立账户数据库。当你的Postfix 系统纯粹用于提供寄信服务，而不用于接收外来邮件，用户也不用登录服务器系统本身时，使用 SASL密码可能是个好主意。请将下列内容加入你的 *smtpd.conf*配置文件：

```
pwcheck_method: auxprop
```

在 Cyrus的术语中，auxprop的意义是Auxiliary Property Plug-ins（辅助性的专属外挂模块），其作用是使用外部程序来进行验证。默认的辅助外挂模块是 sasldb（这也是Cyrus SASL包随附的程序之一），它应该能满足 Postfix 的所有需求。关键字 auxprop只是要求使用外部的 SASL 密码文件。

使用SASL 密码时，不需要用到 saslauthd daemon，但是你必须将所有的SMTP client账户与密码存放在一个专用的外部密码文件中。 SASL默认使用的密码文件是 */etc/sasldb2*。Postfix SMTP server至少要具备能读取此文件的权限；如果你使用Cyrus SASL的 auto_transition功能（参阅 Cyrus说明文件），则 Postfix将需要能够写此文件的权限；如果你用不到 auto_transition 功能，最好不要将写权限开放给 Postfix。

如果还有其他进程也需要能够访问 SASL 密码文件（比方说，POP/IMAP server），你必须适当调整该文件的拥有权与访问权限，让相关进程都能访问它。举例来说，你可以建立一个 *sasl*组，并确定 *postfix*与其他需要访问该文件的账户都隶属于此组。如果有任何其他进程需要更新该文件，则只读模式可能太严格，而你必须提供写权限给必要的进程。下列命令将 */etc/sasldb2*的访问模式设定为 440，这使得 *sasl*组的所有成员都能够读取该文件，除此之外的其他进程则没有访问权：

```
# chown postfix:sasl /etc/sasldb2
# chmod 440 /etc/sasldb2
```

Cyrus SASL包所提供的 saslpasswd2 工具，可用来产生、维护 */etc/sasldb2* 密码文件。对于每一个账户，你必须提供三项信息：登录名称、 SASL 网域名称、密码。就 Postfix 而言，网域名称必须与 myhostname 参数的值吻合。所以，最保险的设定方法，就是利用 postconf -h myhostname 来决定网域名称，如下：

```
# saslpasswd2 -c -u `postconf -h myhostname` kdent
Password:
Again (for verification):
```

"-c"选项要求 saslpasswd2 创建（create）一个账户。"-u"选项指出该账户所属的网域，其值直接取自 Postfix 的配置文件。

设定 Postfix

所有与 SASL密码验证相关的 Postfix参数，全部都是以 smtpd_sasl*（关于SMTP server 的参数）或 smtp_sasl*（关于 SMTP client 的参数）为前缀。对于服务器端的配置，你至少需要设定 smtpd_sasl_auth_enable 参数，并且将 permit_sasl_authenticated 限制条件列在某一个 smtpd_*_restriction 的过滤规则里（参阅第十一章）。

启用 SASL验证

smtpd_sasl_auth_enable 参数决定 Postfix SMTP server 是否支持 SASL 验证：

```
smtpd_sasl_auth_enable = yes
```

有些老的 MUA没有完全正确遵守 SMTP验证协议(注 2)。依照规范说明书的标准规定，当 SMTP client送出 EHLO命令之后，SMTP server应该要列出其验证机制支持列表，而且此列表是出现在关键字 AUTH 与一个空格之后，例如：

```
220 neon.oreilly.com.tw ESMTP Postfix
EHLO client.oreilly.com.tw
250-neon.oreilly.com.tw
250-PIPELINING
250-SIZE 10240000
250-ETRN
250-AUTH PLAIN LOGIN DIGEST-MD5 CRAM-MD5
250-XVERP
250 8BITMIME
quit
221 Bye
```

注 2: 据报道，Microsoft Outlook 与 Outlook Express 5.0之前的版本，会有这种不遵守标准的行为。不过，你可能需要实验，才能判断你的客户端是否如此。

不过，有些 MUA 却期待收到 AUTH 与一个等号：

```
250-AUTH=PLAIN LOGIN DIGEST-MD5 CRAM-MD5
```

Postfix 容许你接受这种不遵守规定的行为：

```
broken_sasl_auth_clients = yes
```

设定此参数之后，Postfix会分别以标准与非标准两种格式来列出它所支持的 SMTP验证机制，例如：

```
220 neon.oreilly.com.tw ESMTP Postfix
EHLO client.oreilly.com.tw
250-neon.oreilly.com.tw
250-PIPELINING
250-SIZE 10240000
250-ETRN
250-AUTH PLAIN LOGIN DIGEST-MD5 CRAM-MD5
250-AUTH=PLAIN LOGIN DIGEST-MD5 CRAM-MD5
250-XVERP
250 8BITMIME
quit
221 Bye
```

由于两种格式都出现在 SMTP server 的响应中，所以既不会影响标准的 MUA，同时又能让那些不标准的 MUA 使用 SMTP SASL 验证。

避免寄件人冒名

当客户端通过 Postfix寄信时，要如何确定客户端使用的是真实的寄件人地址？比方说，某人以 A 身份通过 SMTP 验证，但是却以 B 为发信人地址，要如何避免这种冒名情况？Postfix容许你设定寄件地址与 SASL登录身份的对应关系。举例来说，假设某人的邮件地址是 *kdent@example.com*，其 SASL登录身份为 *kdent*，如果你希望 *kdent* 只能以该地址的名义来发邮件，而不能使用其他寄件地址，你应该将下列对应关系定义在一个查询表中：

```
kdent@example.com                kdent
```

这是一个普通的Postfix查询表，你可以逐一列出每一个地址与每一位SASL用户的对应关系，也可以使用正则表达式（regular expression）来表示邮件地址的人名部分或网域部分（关于Postfix 查询表的格式，请参阅第四章）。制作好查询表之后，请将 *main.cf* 的 smtpd_sender_login_maps 参数指向此表：

```
smtpd_sender_login_maps = hash:/etc/postfix/sasl_senders
```

下一步是将reject_sender_login_mismatch限制条件纳入某个 smtpd_*_restrictions 过滤规则组合里（参阅第十一章关于 UBE过滤规则的说明），这样一来，如果 *kdent* 通

过 SASL验证，但是他却试图以 *mary@example.com* 的名义寄出邮件，那么 Postfix 将拒绝帮他寄信。

核准授权用户

如果你的UBE过滤规则里包含了 smtpd_recipient_restrictions ，你得使Postfix 准许通过验证的用户使用转发服务，也就是将 permit_sasl_authenticated安插在限制条件里的适当位置（参阅第十一章）。比方说：

```
smtpd_recipient_restrictions =   permit_mynetworks
        reject_invalid_hostname
        permit_sasl_authenticated
        reject_non_fqdn_sender
        reject_non_fqdn_recipient
        reject_unknown_sender_domain
        reject_unknown_recipient_domain
        reject_unauth_pipelining
        reject_unauth_destination
        permit
```

如果你没设定 smtpd_recipient_restrictions参数（直接使用默认过滤规则），那么，你得在 *main.cf* 加入下列规则：

```
smtpd_recipient_restrictions = permit_mynetworks,
        permit_sasl_authenticated, reject_unauth_destination
```

设定验证机制

当客户端联机到 SMTP server 时，有哪些密码验证机制可使用，由 smtpd_sasl_security_options参数决定。完整的机制选项，取决于你的系统上有哪些机制可用以及你的 SASL 函数库支持哪些机制。如果不指定任何选项，默认值是接受包括明文密码（plaintext password）在内的所有可用机制，但匿名登录（anonymous login）除外。如果使用了 saslauthd daemon， 就必须接受明文密码，所以，默认值的设想是合情合理的。如果你指定了任何选项，则默认值无效，所以你的选项里必须包含 noanonymous。例如：

```
smtpd_sasl_security_options = noanonymous, noplaintext
```

以下是通用的机制选项：

noplaintext

此选项将 plaintext密码验证排除在外。这使得 SASL选择 challenge/response技术（不传送实际密码）来使用。如果你的安全政策不容许密码以明文形式流经网络，那就指定 noplaintext 选项吧，但是这也表示你不能使用 saslauthd。

noactive

此选项将可能遭受“主动攻击”（active attack）的密码机制排除在外。在“主动攻击”中，攻击者想办法将他们自己安插到 client与 server之间，所以，某些类型的主动攻击又被称为“中间人攻击”（man-in-the-middle attack）。攻击者可以读取或改变数据，让 client 或 server误以为数据真的是对方送来的。

nodictionary

此选项将可能遭受“字典攻击”（dictionary attack）的密码机制排除在外。在“字典攻击”中，攻击者使用预编的密码数据库，逐一测试哪个密码“恰好”能闯入你的系统。这类密码数据库通常由城市、团队、宠物、常见人名、字典词汇，加上各种明显的词汇变化等组成，所以称为“字典攻击”。

noanonymous

排除匿名登录。SMTP验证的最主要目的，就是要确认用户的身份，所以“匿名登录”对 SMTP server是没有意义的。Postfix 的默认行为是禁止匿名登录。如果你指定了其他选项（忽略了默认值），你必须明确设定 noanonymous。

mutual auth

要求使用互证机制 —— client与 server 双方都要提供证明自己身份的证据。指定 mutual_auth之后，会使得 Postfix 只声明具有互证能力的验证协议。

设定步骤汇整

在Postfix系统中添加 SASL功能所涉及的步骤有点繁复。总结本章前述的知识，以下是我们整理出来的设定步骤:

1. 决定你打算支持的验证机制与架构。
2. 安装SASL 函数据库,并重新编译Postfix,使其包含SASL。或者取得已经内含SASL的 Postfix包，以及相关的 SASL 验证机制。
3. 重新安装 Postfix。
4. 创建 */usr/local/lib/sasl2/smtpd.conf*配置文件，将 pwcheck_method参数设定为 saslauthd（如果使用 SASL saslauthd 与系统密码的话）或 auxprop（如果使用 SASL 的专属密码文件的话）。
5. 如果你选择的验证架构是 Unix系统密码，请启动 saslauthd daemon ,并且指出你的系统所用的验证方式；否则，使用 saslpasswd2命令在你的系统上创建SMTP client 的账户与密码。
6. 编辑 *main.cf*，启动 SASL验证功能，并指定验证方式。基本的设定至少需要下列两项参数：

```
smtpd_sasl_auth_enable = yes
smtpd_recipient_restrictions = permit_mynetworks,
        permit_sasl_authenticated, reject_unauth destination
```

7. 重新加载 Postfix，使我们在 *main.cf* 配置文件所做的改变生效：

```
# postfix reload
```

测试 SASL 验证配置

经过重重难关之后完成安装，要如何要确定所有步骤都没有出错，证明 Postfix确实真的已依照我们的要求来进行验证？等到用户旅行到外地了以后，看看他们会不会打长途电话回来抱怨，肯定不是好办法。最好的办法，是直接观察 SMTP server 的交互情况，实际体验验证过程，并立刻查阅日志文件留下的线索。

要想联机到 SMTP server，最容易的方法是使用 telnet 工具程序，然后与服务器进行SMTP对话（参考第二章的 SMTP对话范例）。最容易测试的是 PLAIN 机制，如果你禁用这项机制，建议你暂时先启用它，等完成测试实验之后再关掉它。

先让我们了解 PLAIN 机制的细节，然后再开始动手做实验。 PLAIN机制要求你在 AUTH 命令之后提供一个身份标识符串，此字符串必须编码成 base64格式。构成此字符串的各项数据的次序是“登录身份”，其后跟着一个 null 字符，然后是“密码拥有者的身份”，再跟着一个 null字符，最后是“密码”本身。通常，“登录身份”与“密码拥有者的身份”是相同的。举例来说，假设用户 *kdent* 的密码为 Rumpelstiltskin，而且其登录身份与密码拥有者身份相同，那么，他的身份标识符串是“`kdent\0kdent\0Rumpelstiltskin`”（编码前的格式）。

麻烦之处，在于如何将标识符串编码成 base64 格式且不包含字符串末端的 CR 字符。如果你的系统上有 `mmencode` 与 `printf` 命令，这步骤应该不难。`printf` 命令能显示出指定格式的字符串，但是不会像 `echo` 之类的命令那样，自动在字符串末端补一个换行字符。`mmencode` 命令能将输入字符串编码成各种 MIME格式，且其默认格式正好是我们所需要的 base64。

所以，我们可用下列命令来产生 base64 格式的标识符串：

```
$ printf 'kdent\0kdent\0Rumpelstiltskin' | mmencode
a2RlbnQAa2RlbnQAcnVtcGxlc3RpbHRza2lu
```

某些系统平台的 `printf` 可能不能正确地处理字符串中间的 null 字 符（\0）。要想知道自己系统上的 `printf` 有没有这个毛病，只要看base64 编码结果是否比原字符串短（译注 4）。

译注 4： base64 格式的字符串，比原字符串更长。

如果你的printf有问题，不妨使用 echo -n（“-n”表示不输出结尾的换行字符）来代替printf。如果你的系统上没有 mmencode，或是没有能正常工作的 printf或echo，我们提供了一个简单的Perl小程序来帮你解决问题。

encode_sasl_plain.pl

如果你的系统上没安装mmencode（或mimeencode),这里有一个简单的Perl脚本可帮你产生测试用的base64字符串。这个脚本需要用到MIME::Base64模块，如果你的系统没安装此模块，你可以从CPAN的任何一个镜像站免费取得。

```
#!/usr/bin/perl

use strict;
use MIME::Base64;

if ( $#ARGV != 1 ) {
        die "Usage: encode_sasl_plain.pl <username> <password>\n";
}

print encode_base64("$ARGV[0]\0$ARGV[0]\0$ARGV[1]");
exit 0;
```

让我们以同样的“ kdent\0kdent\0Rumpelstiltskin ”字符串为例，示范如何使用这个Perl脚本：

```
$ encode_sasl_plain.pl kdent Rumpelstiltskin
a2RlbnQAa2RlbnQAcnVtcGxlc3RpbHRza2lu
```

你可用“剪贴”的方法，将编码出来的base64字符串贴到你的Telnet对话内容中。下面的例子示范如何使用telnet联机到SMTP server，并展开ESMTP对话（黑体字代表需要手动输入的部分）：

```
$ telnet neon.oreilly.com.tw 25
Trying 192.168.0.10...
Connected to neon.oreilly.com.tw.
Escape character is '^]'.

220 mail.example.com ESMTP Postfix
EHLO test.ora.com
250-mail.example.com
250-PIPELINING
250-SIZE 10240000
250-ETRN
250-AUTH PLAIN LOGIN DIGEST-MD5 CRAM-MD5
250-AUTH=PLAIN LOGIN DIGEST-MD5 CRAM-MD5
250-XVERP
250 8BITMIME
AUTH PLAIN a2RlbnQAa2RlbnQAcnVtcGxlc3RpbHRza2lu
```

```
235 Authentication successful
quit
221 Bye

Connection closed by foreign host.
```

如果没有见到验证成功的信息，请检查邮件日志文件，看看 Postfix 回复了什么问题。由于牵涉的东西很多，所以问题症结可能不太容易追查，多花点时间吧！

当你使用 `telnet` 测试验证功能时，如果 Postfix 对于 `EHLO` 命令的响应中，不包括下面这段信息：

```
250-AUTH PLAIN LOGIN DIGEST-MD5 CRAM-MD5
```

很有可能是你忘了在 *main.cf* 配置文件里设定 `smtpd_sasl_auth_enable` 参数。如果你设定了该参数（而且没打错字），另一种可能是你的 Postfix 没链接 SASL 函数库，你最好重新编译 Postfix，确定编译程序确实已和 SASL 函数库链接。

如果日志文件告诉你它找不到 `db` 文件，请确定 `saslpasswd2` 所产生的密码文件确实位于 */etc* 目录下，而且该文件的权限模式足以让 postfix 账户访问。如果你的 Postfix 处于 chroot 环境下，请将 */etc/sasldb2* 复制到 */var/spool/postfix/etc/* 目录下。Cyrus SASL 包提供了一组检测工具，可能有助于你找出问题所在。请仔细阅读 `sasldblistusers2` 与 `saslpasswd2` 的说明文件，确定你没漏掉什么。

SMTP 客户端验证

本章前半段的讨论，着重于 Postfix 如何扮演 SMTP 验证的服务器端角色 ——验证远程 MUA 或 MTA 的身份，借此判断对方是否有权使用转发服务。现在，我们要换个角度，讨论 Postfix 如何扮演客户端角色 ——提出自己的身份证明，借此获得远程 MTA 的转发服务的使用权。

首先，你得提供一个密码文件给 Postfix，其中包含能通过远程服务器的证书数据。此密码文件的每一笔记录，各包含一个代表远程服务器的网域（或主机名称）以及一组能通过该服务器验证的账户与密码，格式如下：

```
destination  username:password
```

当 Postfix 要寄出一封邮件时，它先检查收件地址的网域部分，如果不能在密码文件中找到完全相符的 destination，再寻找打算要联机的主机名称。这种检查过程使得 Postfix 可以轻易连接共享相同账户数据库的多个 MX 主机。决定密码文件位置的参数是 `smtp_sasl_password_maps`。

决定客户端行为的是 smtp_sasl_security_options参数，其设定方法与服务器端的 smtpd_sasl_security_options参数相同，所以不再赘述。如果你没指定任何选项，则默认行为是容许系统上所有能找到的机制，唯独匿名登录除外。

SMTP客户端验证的设定过程

让我们举个例子来示范要如何设定 Postfix，才能使其在通过远程系统转发邮件时，提出自己的身份证明。在这个例子中，假设我们有一组可以通过 *ora.com* 网域的所有服务器主机的账户以及一组可以通过 *mail.postfix.org* 主机的账户。

1. 创建 */etc/postfix/sasl_passwd* 密码文件，并填入我们拥有的账户与密码，例如：

```
ora.com                 kdent:Rumpelstiltskin
mail.postfix.org        kyle:quixote
```

2. 使用 postmap产生数据库文件：

```
# postmap /etc/postfix/sasl_passwd
```

3. 编辑 *main.cf* 配置文件，启动客户端验证功能。请注意，这次要设定的参数是 smtp_sasl_auth_enable ，而不是 smtpd_sasl_auth_enable（后者用于启动服务器端验证）。接着，在 smtp_sasl_password_maps 参数中指出密码文件的位置：

```
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
```

4. 重新加载 Postfix，使我们所做的改变生效：

```
# postfix reload
```

现在，每当 Postfix SMTP client试图通过 */etc/postfix/sasl_passwd* 所列的任何网域或主机来转发邮件时，它就会提供对应的验证数据。比方说，如果你的 Postfix smtp client联机到 *mail.ora.com* 服务器，则它会使用 kdent与Rumpelstiltskin这组账户与密码。

第十三章

传输层安全协议(TLS)

从 SSL演变而来的“传输层安全协议”（Transport Layer Security，TLS），能以加密技术来保障TCP通信的私密性（信息不外泄）与完整性（能侦测数据在传输过程中是否遭到窜改)。RFC 3207为 SMTP制定了一种称为 STARTTLS的扩充机制，其主要用途在于保障点对点通信的私密性，也能确保你的邮件不会被送错地方，比方说，避免邮件被投递到伪装成收信地的非法系统。另一种有用的应用是与SASL结合，避免 PLAIN密码以明文(clear text）形式流经网络。

TLS令人赞赏的优点之一，在于两系统之间不必经过事先安排，就可以获得可靠的身份标识与私密性的保证，甚至连“强效验证”（strong authentication）也是可以办到的——如果用户的 MUA支持的话。借由客户端证书（client certificates)，邮件服务器可确信联机进来的客户端确实是对方自己所宣称的身份。“客户端证书”是一种以密码技术签名的标识符（参阅后面的“TLS证书概论”），可用来取代或搭配SASL验证（参阅第十二章），但是需要额外的控制管理工作，因为你得要为每一个客户端产生专用的证书，并且将这些证书“逐一”安装在每个客户端。不过，如果你只想利用 TLS来加密验证数据，那倒是相当容易设定。

有个很重要的概念一定要谨记在心： TLS不是用来保护邮件内容。当 client与 server双方以TLS进行通信时，所有东西（包括邮件本身）确实是被编码成密文形式，然而，TLS能保护的也仅止于两系统间的通信。服务器收下了邮件之后，仍可能以明文形式储存，当服务器要将邮件传递到下一站，或是收件人从服务器下载邮件时，邮件内容仍可能以明文形式流经网络。换言之，除非你能确定邮件从出发点到终点站之间的全程都受到加密保护，否则，就不能保证完全没有泄密之虞。要达成点对点的私密性，你需要客户端的解决方案（直接对邮件本身加密），像是 PGP或 S/MIME。

Postfix 与 TLS

Postfix 对于 TLS 的支持，由 Lutz Janicke 所写的一组修补文件（patch）提供。从 Postfix 主页的 Add-on Software 链接处可以取得这组修补文件，附录三示范如何使用 TLS patch 来加强你的 Postfix。如果你使用平台随附的预编译版 Postfix 包，请确定该版本确实包含 TLS patch；否则请参阅附录三的步骤，重新编译、安装支持 TLS 的 Postfix。

除了 Postfix本身要支持TLS之外，你还必须制作、设定TLS证书。你需要一个公钥（public key）与一个私钥（private key）。公钥是代表服务器身份的证书，你必须向具有一定信誉的认证中心（Certificate Authority，CA）提出申请，他们依据你的申请书向你求证之后，会以他们自己的数字签名签署一个公钥给你，只要客户端愿意相信CA的签名，就等于间接相信 CA核发的公钥所证明的系统 —— 你的服务器。除了你自己的证书之外，你还必须取得 CA的公钥（核发证书给你的那一家 CA）。

对电子交易而言，让买方能够证明卖方身份是很重要的。因此，网络交易商确实有必要花钱取得 CA 的签名。然而，就保密通信而言，服务器公钥是否来自有公信力的 CA，并不是绝对必要的事，这表示你可以自己扮演 CA 的角色，自己核发证书给自己。当客户端 MUA 连接到你的 TLS server时，如果不认识你使用的 CA证书，MUA 通常会提供一个机会给用户，让用户自己决定要不要相信这个 CA 以及是否将 CA 的证书纳入认同名单中。

TLS 证书

Postfix TLS patch 需要用到 OpenSSL 函数库。OpenSSL 包随附了一组管理证书的命令行工具，你可用这些工具来制作证书。就 Postfix而言，所有证书都必须是 PEM格式（主要数据以 base64编码，再加上一些额外的译码信息）。OpenSSL 工具的默认输出格式刚好就是 PEM，所以不用任何转换就可产生 Postfix 可用的证书。OpenSSL 工具的默认安装目录位于*/usr/local/ssl/*，管理证书所需的命令行工具是 `openssl`。

自己开设认证中心

服务器端证书必须要经过CA的签署才有效。你可以付费请有公信力的CA为你签署，不过，就保密用途而言，你并不需要花这笔钱。OpenSSL包提供了一个脚本，让你可以自己开设 CA，自己签署自己的证书。在 OpenSSL 的安装目录下，键入下列命令：

```
# misc/CA.pl -newca
```

然后回答所有问题，完成之后，它会在*./demoCA/* 目录下产生开设 CA 所需的全部文件，其中包括可用来签署证书的“CA 数字签名”（又称为“根证书”）。

TLS 证书概论

TLS 使用公钥加密技术，让 client/server 之间能够进行私密通信。此外， TLS 能保证没有人能够在传输中途窜改信息，或是冒名伪装成某一方，因为协议本身容许通信双方互相验证彼此的身份。然而，再次提醒 TLS 的好处仅限于 TLS联机两端，至于数据在传输之前发生过什么事，以及在传输到目的地之后会被怎样处理， TLS都无法保证。

公钥加密技术的原理，是运用一对互补的 key（译注 1），其中一个可以公开给众人知道，称为“公钥”（public key）；而另一个 key则只能由个人拥有，绝不可泄露，所以称为“私钥”（private key）或“密钥”（secret key）。以公钥加密的数据，只可由私钥予以解密，反之亦然。利用这种特性，公钥可用来让别人传输私密数据给你，而你可用私钥来证明自己的身份。当别人要传递机密数据给你时，对方可使用你事先提供的公钥来加密数据，由于只能用你自己的私钥予以解密，所以不怕机密数据泄露（除非你泄露自己的私钥）。私钥又如何用来证明自己的身份？你可用自己的私钥加密一份数据（这动作称为“签名”），如果对方能用你的公钥解开，表示数据一定来自于你，不可能是别人（除非你把自己的私钥泄露给别人）。因此，你的私钥可视为你个人的数字签名。一般而言，你应该尽可能将公钥散布出去，而私钥则必须不计一切代价予以保护。

当你收到别人的公钥时，你如何能够相信，公钥的拥有者确实是对方所宣称的那个人？实际上，公钥散布时，往往会同时散布一个代表公钥拥有者的标识符 —— 通常是服务器的主机名称。收到公钥的一方，可通过比对此标识符是否与 DNS 查出的主机名称（或是联机握手过程中得到的名称）吻合，借此确认公钥的拥有者确实是当前的联机对象。

有 CA 数字签名的公钥，称为证书（certificate）。CA 通常是交易双方都信任的第三方组织。理论上，有 CA 的证明，表示公钥拥有者的身份已经被 CA 调查并予以证实，取得公钥的人，可以相信该公钥确实属于其宣称的拥有者。换言之，证书的授信基础来自于你对 CA 的信任。值得注意的是， CA 只担保证书拥有者的身份（identity），而不是担保其信用（credit）。

在加密通信的过程中，公私钥仅用于联机初期，让双方互相确定对方的身份，并协

—待续—

译注 1： key：极大的质数。越大的质数（位数越多），就越难破解。因此， key 的长度也隐喻着破解的难度。

商出一个随机产生的session key（译注2），实质的通信内容，其实是由这个session key来加密与解密。一个 session key 只用于一次通信，在通信完毕之后， session key就可以丢掉了。

让我们看看 client与 server之间是如何进行秘密通信的。首先， client联机到 server，并提出秘密通信要求。对于 Web server，client使用 https 命令；对于 SMTP server，client 发出 STARTTLS命令。

如果 server 同意请求，便会返回一个由 CA 签署的证书，其中含有 CA 的数字签名以及一个代表 server的标识符。client检验 server的标识符是否符合预期，并检查该 CA 的数字签名是否在认同名单中。如果两项检查都过关， client 与 server双方便开始展开 session key 协商，决定后续通信内容要使用哪一种加密算法，并产生一个只用于该次通信的 session key 。接着，双方便使用协议出来的session key 与算法进行秘密通信。

产生服务器端证书

产生服务器端证书的第一步,是使用 openssl工具为服务器产生一对公钥与私钥，然后产生一个“证书签署请求” （certificate signing request，CSR），并将 CSR 与公钥交给 CA 签署。经过 CA签署的公钥证书可以广泛散布出去，但是私钥则必须被谨慎保管。事实上，有许多应用系统将私钥加密封存在一个特殊文件，在访问私钥之前，必须先提供密码（passphrase）才能解密，这种存储私钥的方法称为“密封”。然而， Postfix需要能够直接访问私钥，不能使用密封方法，因为访问私钥的动作发生在运行时，而此时你不可能实时提供密码。

OpenSSL包提供一组脚本可帮助你产生公私钥与 CSR，不过，它们所产生的key是“密封”的，所以，你得直接使用 openssl命令来产生公私钥：

```
$ openssl req -new -nodes -keyout mailkey.pem \
>    -out mailreq.pem -days 365
```

openssl的 -new选项表示你想产生公私钥与CSR，-nodes 选项表示不加密，-keyout 和 -out 分别指出私钥文件与 CSR 文件的名称。最后，-days 365指出证书的有效期限是一年。

译注 2: 公私钥属于“非对称式”系统，公钥加密的数据只有私钥能解密，缺点是加密与解密所需的运算量相当庞大。 session key属于“对称式”系统，也就是加密与解密都是使用相同的 key，其优点是运算速度快。

如果你使用第三方 CA（这表示你要付费),请按照 CA 的指示，提出你的 CSR（此例中的 *mailreq.pem* 文件）来要求他们签署。如果你自己扮演 CA 的角色，你可用下列命令来核发证书：

```
# openssl ca -out mail_signed_cert.pem -infiles mailreq.pem
```

此步骤所产生的 *mail_signed_cert.pem* 文件，就是 CA 所核发的证书。

你可能会想要将 Postfix/TLS 用到的所有证书文件复制到一个方便的位置。假设你完全依照上述的步骤，则请用下列命令将证书文件复制到 Postfix 的配置目录下：

```
# cp /usr/local/ssl/mailkey.pem /etc/postfix
# cp /usr/local/ssl/mail_signed_cert.pem /etc/postfix
```

*mailkey.pem*文件存有服务器的私钥（万万不可泄露！）， *mail_signed_cert.pem* 是 CA签署的公开证书（可放心散布）。由于 Postfix 不能使用密封的私钥文件，所以你应该以最严格的权限模式来保护私钥文件：

```
# chown root /etc/postfix/mailkey.pem
# chmod 400 /etc/postfix/mailkey.pem
```

上述命令将私钥文件的拥有权判给 *root* 账户，而且只有 *root* 能够读取。

安装 CA 证书

Postfix/TLS server 必须要能够访问 CA 的公开证书（也就是所谓的“根证书”），包括为你的服务器签名的那一个 CA 以及核发证书给你的用户的每一个CA。当然,如果双方使用同一个 CA ，则只需要安装一个根证书就够了。

如果你的服务器端证书是自己签发给自己的，请将先前的 `CA.pl` 脚本所产生的 *cacert.pem*文件复制到 Postfix 的配置目录：

```
# cp /usr/local/ssl/demoCA/cacert.pem /etc/postfix
```

如果你的服务器或任何客户端的证书是第三方 CA所签发的，你必须设法取得这些 CA 的根证书(PEM格式）；对于你不信任的 CA，自然没必要取得他们的根证书。请将搜集到的所有根证书集中在 */etc/postfix/cacert.pem* 文件里。

有两种方法可将新的 CA根证书安装到 Postfix/TLS 系统。第一种办法是将所有根证书集中在一个文件，并将 `smtpd_tls_CAfile` 参数指向此文件。你只需将新的根证书附加在现有文件末端即可。比方说，若原有的 CA根证书是存储在 */etc/postfix/cacert.pem* 文件中，而新 CA的根证书是存放在 *newCA.pem*文件中，那么，下列命令可将新的根证书纳入认同名单：

```
# cp /etc/postfix/cacert.pem /etc/postfix/cacert.pem.old
# cat newCA.pem >> /etc/postfix/cacert.pem
```

另一种方法是将每个CA的根证书全部集中一个专用目录下的个别文件里，并将smtpd_tls_CApath参数指向此目录。以后，每当需要安装新的CA根证书时，只要将新的证书文件存放在此目录下，然后执行一次OpenSSL的c_rehash命令即可。举例来说，假设*newCA.pem*是你想安装的新的证书文件，而根证书目录是*/etc/postfix/certs/*，则安装步骤如下：

```
# cp newCA.pem /etc/postfix/certs
# c_rehash /etc/postfix/certs
```

当你有许多CA根证书要处理时，这种方法可让维护工作稍微轻松些；不过，如果你的Postfix是在chroot环境下运行，则还需要将新的根证书文件复制到chroot环境下的对应目录，然后运行postfix reload，新证书才会有效。

设定 Postfix/TLS

Postfix TLS patch引进了一组关于TLS运作环境的参数。本小节只列出基本配置所需的关键参数，至于完整的TLS参数说明，请参阅TLS patch随附的配置文件样本。

smtpd_use_tls

启动TLS支持。如果没有设定此参数或是设定成no，Postfix的运行方式就像没有TLS patch一样。例如：

```
smtpd_use_tls = yes
```

smtpd_tls_key_file

指向服务器私钥文件。例如：

```
smtpd_tls_key_file = /etc/postfix/mailkey.pem
```

smtpd_tls_cert_file

指向服务器的PEM证书文件（必须经过CA签署）。例如：

```
smtpd_tls_cert_file = /etc/postfix/mail_signed_cert.pem
```

smtpd_tls_CAfile

指向CA根证书文件。该文件含有所有你愿意信任的CA的公开证书。例如：

```
smtpd_tls_CAfile = /etc/postfix/cacert.pem
```

smtpd_tls_CApath

指向CA根证书文件目录。该目录下的每个PEM文件，都含有一个你信任的CA的公开证书。例如：

```
smtpd_tls_CApath = /etc/postfix/certs
```

将上述参数设定到*main.cf*配置文件，并运行postfix reload之后，你的Postfix/TLS server将具备秘密通信的能力，并准备迎接客户端的STARTTLS命令。

Postfix/TLS的设定过程整理

总结前述的基本知识，在Postfix系统上设定TLS的大致过程如下：

1. 安装OpenSSL包（如果你的系统上没有的话），因为我们需要使用该软件包来产生TLS证书。
2. 使用TLS patch，并重新编译、安装Postfix（详细步骤请参阅附录三）。
3. 产生一份服务器端证书以及一个证书签署请求（CSR），然后将两者一并提交给有一定信誉的CA请求签署；或是自己扮演CA的角色，自己核发证书给自己。
4. 将所有证书文件（服务器的密钥、CSR、CA的根证书）安装在Postfix目录下。
5. 编辑*main.cf*，设定下列TLS参数：

   ```
   smtpd_use_tls = yes
   smtpd_tls_key_file = /etc/postfix/mailkey.pem
   smtpd_tls_cert_file = /etc/postfix/mail_signed_cert.pem
   smtpd_tls_CAfile = /etc/postfix/cacert.pem
   ```

   如果还需要设定其他TLS参数，现在正是时候（关于其他TLS参数，请参阅TLS patch的说明文件）。
6. 重新加载Postfix，使我们在*main.cf*所做的改变生效：

   ```
   # postfix reload
   ```

现在，每当有客户端要求秘密通信，Postfix/TLS server都能适当应对。

取得客户端证书

你可能会想要使用客户端证书来代替或加强其他SMTP验证技术。客户端证书是非常可靠的验证方法，因为其非常难以假造。

客户端证书必须由CA核发。如果你打算选择一个CA来核发证书给你的用户，你应该遵照该CA的申请过程来取得客户端证书。当然，你也可以使用OpenSSL包提供的工具，自己扮演CA的角色，自己签发证书给用户。

制作客户端证书

客户端证书的制作程序，其实与服务器端证书的制作过程很相似（参阅“产生服务器端证书”），唯一差别是多了一个步骤：将签署好的证书转换成 MUA 可用的格式。大部分流行的 MUA 都偏好 PKCS12 格式的证书文件，这种格式将签署好的证书与私钥封装在一起，并以一个密码保护。如果你使用第三方 CA，该 CA 应该能提供正确格式的证书文件来满足用户的 MUA。如果你自己签署证书，你应该制作 PKCS12 格式的证书文件来分发给用户。证书文件应该包含核发给用户个人的证书（公钥）、搭配于该证书的私钥以及你自己的 CA 根证书。

对于每一位你打算以证书来验证其身份的用户，你都必须分别产生一对专属的公私钥。你应该制定“辨别名称”(distinguished name，DN）的命名原则。一般而言，在产生证书时，你应该使用个人的邮件地址或是客户端机器的主机名称。举例来说，假设你决定以邮件地址为辨别名称，而现在要签发证书给 *kdent@ora.com* 这位用户，步骤如下：

1. 使用 `openssl` 命令产生一对公私钥。请注意，你自己的公钥也必须有 CA 的签名（很可能就是你自己）：

   ```
   $ openssl req -new -nodes -keyout kdentkey.pem \
   >    -out kdentreq.pem -days 365
   ```

   由于使用了 `-new` 选项，所以上述命令会产生一个私钥与一份 CSR。`-nodes` 选项要求 `openssl` 不要将私钥密封。`-keyout` 和 `-out` 分别指出私钥文件和 CSR 的文件名。最后，`-days 365` 表示证书的有效期是一年。

2. 签署证书。如果你与 CA 合作，请按照他们的申请流程，提出前一步骤所产生的 CSR（*kdentreq.pem* 文件），要求他们签署。如果你自己作为 CA，请用下列命令签署证书：

   ```
   # openssl ca -out kdent_signed_cert.pem -infiles kdentreq.pem
   ```

3. 将签署好的证书文件转换成适当格式。问清楚用户所用的 MUA 是哪一种，将 CA 签章的证书文件转换成该 MUA 能接受的格式（假设是 PKCS12）：

   ```
   # openssl pkcs12 -in kdent_signed_cert.pem \
   >   -inkey kdentkey.pem -certfile /etc/postfix/cacert.pem \
   >   -out kdent.p12 -export -name "kdent@ora.com"
   ```

   这命令会要求你提供一个密码来保护所产生的文件（*kdent.p12*），而你必须将这个密码告知用户。`-certfile` 选项指出你自己的 CA 根证书文件（以 `CA.pl` 脚本所产生的那个文件）。完成之后，就可将 *kdent.p12* 文件与脚本交付给用户。

   最后一步，请用户自己将证书文件安装到 MUA。大多数 MUA 都提供相当简便的步骤来导入证书文件，所以应该不成问题。如果你有不擅长操作计算机的用户，请给予适当的指导。

设定客户端证书验证

Postfix/TLS依据证书的“指纹”(fingerprint）来判别证书是否可接受。“指纹”是从证书中计算出来的值，不同的证书，不可能计算出相同的指纹。你必须将每个客户端证书的指纹都存放于一个标准的 Postfix查询表（参阅第四章）。每当有客户端出示其证书，Postfix/TLS 就计算该证书的指纹，然后检查该指纹是否已登记在查询表，借此决定是否要容许客户端使用转发服务。

你必须收集每一位获得授权的客户端的证书指纹。许多 MUA 都有显示证书指纹的功能，你可以要求用户将他们在 MUA 看到的证书指纹提供给你。如果他们的证书是你自己核发的，你可以用 `openssl x509`命令直接算出指纹：

```
$ openssl x509 -fingerprint -noout -in kdent_signed_cert.pem \
>  | cut -d= -f2
57:8E:95:63:67:CD:2B:96:7C:0A:3A:61:46:A5:95:EA
```

整个设定流程如下：

1. 分别取得每一位用户的证书指纹。你可以按照上述步骤自己计算，或是要求用户提供给你。
2. 将收集到的指纹集中在*/etc/postfix/clientcerts* 文件，并注明其对应的辨别名称（主机名称或邮件地址），参考格式如下：

   ```
   57:8E:95:63:67:CD:2B:96:7C:0A:3A:61:46:A5:95:EA    kdent@ora.com
   ```

   由于这是标准的 Postfix 查询表，所以对应值不可以省略 —— 即使不一定用到。
3. 将制作好的 *clientcerts* 查询表转换成数据库：

   ```
   # postmap /etc/postfix/clientcerts
   ```
4. 编辑 *main.cf* 配置文件，加入下列参数：

   ```
   relay_clientcerts = hash:/etc/postfix/clientcerts
   smtpd_tls_ask_ccert = yes
   smtpd_recipient_restrictions =
           permit_mynetworks
           permit_tls_clientcerts
           reject_unauth_destination
   ```

   注意，`smtpd_tls_ask_ccert`参数名称中那两个连在一起的字母 c，代表“client certificate”（客户端证书），不是印错字。如果你已经定义了 `smtpd_recipient_restrictions`参数，请将 `permit_tls_clientcerts` 加入你的过滤规则中。
5. 重新加载 Postfix，使我们在 *main.cf*所做的改变生效：

   ```
   # postfix reload
   ```

TLS/SMTP client的设定过程

既然SMTP/TLS server能要求客户端出示证书，那么，当 Postfix 扮演客户端角色时 —— 寄信到其他SMTP server ,或要求其他SMTP server转发邮件 —— smtp MDA 也可能被要求提供客户端证书。注意，一个Postfix系统只能有一个代表自己的客户端证书，除非你在 *master.cf* 中设定了其他 MDA，才有可能安装多个的客户端证书。

证明服务器端身份的证书，也可以用来证明客户端身份。不过，正式CA核发的证书，不见得能让你同时用在两种身份上。在这种情况下，你可能需要向CA 另外申请一个客户端证书。我们先前制作的自签服务器端证书没有这样的限制。不管你的客户端证书是怎么来的，其辨别名称都必须符合 `myhostname` 参数所指定的主机名称。

制作客户端证书的过程，其实与服务器端证书的制作过程完全一样，所以不再赘述。如果你打算使用同样的证书，那么，只要将几个TLS client参数指向 TLS server参数所用的相同文件即可。

以下是最基本的TLS client参数。关于其他 TLS参数，请参阅 TLS patch 随附的配置文件样本。

`smtp_use_tls`

启动 Postfix SMTP client的 TLS 支持。例如：

```
smtp_use_tls = yes
```

`smtp_tls_key_file`

指向客户端证书所对应的私钥文件。例如：

```
smtp_tls_key_file = /etc/postfix/mailkey.pem
```

`smtp_tls_cert_file`

指向客户端证书文件。例如：

```
smtp_tls_cert_file = /etc/postfix/mail_signed_cert.pem
```

`smtp_tls_CAfile`

签署客户端证书的CA的根证书文件，例如：

```
smtp_tls_CAfile = /etc/postfix/CAcert.pem
```

假设你打算让 smtp 使用与 smtpd 相同的证书，设定步骤相当简单：

1. 编辑 *main.cf*，设定下列参数：

```
smtp_use_tls = yes
smtp_tls_key_file = /etc/postfix/mailkey.pem
```

```
smtp_tls_cert_file = /etc/postfix/mail_signed_cert.pem
smtp_tls_CAfile = /etc/postfix/cacert.pem
```

如果你还想设定其他 TLS参数，请一并编辑完成。

2. **重新加载 Postfix，使 *main.cf* 的改变生效：**

```
# postfix reload
```

现在，每当 Postfix联机到一个要求出示客户端证书的 SMTP server，smtp便会提供必要的信息。

第十四章

内容过滤

内容过滤器（content filter）是一种能扫描邮件标题与正文的工具，扫描过程可能改变邮件内容，扫描结果可能会影响邮件的处理流程（拒收、代为回信、转移到他处等）。最知名的例子，莫过于病毒与垃圾邮件的过滤程序。病毒通常是藏身在邮件内容中，而垃圾邮件也不一定能依据传送方的IP地址或信封信息检测出来，因此，我们需要内容过滤器来抵制这些问题。

本章将解释如何在邮件服务器上进行内容过滤——虽然这不一定是最佳选择。只有全面性的过滤，才适合在MTA层运行。如果你需要让个别用户能够调整过滤方式，则应该在MDA或MUA层进行过滤。且让我们分析各层的优缺点：

MDA

诸如`procmail`、`sieve`之类的MDA，可让用户自己决定如何整理收下的邮件。一般而言，这类MDA的配置文件是放在邮件系统上（于用户个人的主目录下），所以，用户必须要有系统账户，而且具有编辑配置文件的能力（这可能需要相当程度的专业训练）。如果用户不能登录系统（没有系统账户）或是不会使用ssh、vi，你可能需要提供其他方式，让他们能够自己设定过滤规则。比方说，在邮件系统上安装Webmin包（*http://www.webmin.com*），提供一个web-based接口让用户能够调整自己的*.procmailrc*配置文件。

MUA

大部分流行的MUA都有过滤能力，供用户用于分类邮件。对于虚拟网域的用户（不能登录邮件系统），或是要满足个别用户的过滤需求，最理想的办法是在MUA层级进行过滤。因为服务器端不必负担扫描邮件所需的运算资源（CPU时间与内存），而是由运算资源较为宽裕的客户端承担。

*Postfix*的标题与正文检查

利用第十一章介绍的`head_checks`与`body_checks`参数，可以进行有限度的过滤。

这种方法的缺点，在于用户不能决定过滤规则，而且会加重服务器的负担，更重要的是，会影响到出入服务器的所有邮件。不过，这可能是最容易设定的方法。

在MTA与MUA两层的双管齐下，可互补彼此的优缺，兼顾广度与细度。以过滤垃圾邮件为例，在MTA层设置太严格的过滤规则很容易发生误判，太宽松又没有效果，如果要依个案设定，所耗的运算成本与资源又太高；另一方面，如果在MUA进行过滤，不一定每位用户都有能力设置过滤规则。因此，比较好的策略，是由MTA filter执行分析工作，在可疑邮件的标题里加上MUA filter可解读的标记，用户可利用标记来设定MUA filter的过滤规则，决定如何处理邮件（接受、拒绝或分类到适当的邮箱）。专用于与MTA合作的Spamassasin垃圾邮件分析软件，就是采用这种策略。关于Spamassassin软件，请参阅他们的网站（*http://spamassassin.org*）。

不过，如果要过滤病毒邮件，在MTA层是最理想的选择。因为病毒邮件的特征明显而固定（相较于垃圾邮件而言），误判机会不高，所以很适合由MTA将之挡在大门外，断绝其进入本地网络的机会，让用户有“干净”的邮箱。当你需要过滤每一封经过服务器的邮件时，这类的过滤动作最适合在MTA层进行处理。

Postfix的`body_checks`与`header_checks`是很有用的工具，它们能影响Postfix所经手的每一封邮件。不过，由于这两种检查一次只比对一行文字，所以，如果你设置了很复杂的过滤规则，将严重降低Postfix的工作效率，而且可能消耗大量的运算资源。此外，这两种检查的结果是决定性的，无法提供比“拒收”与“接受”更复杂的处理动作，这表示你不能将符合条件的邮件交给其他程序处理，或是转寄到其他地方。总而言之，除了简单的过滤动作之外，其他细节不应该由Postfix本身来执行，因为Postfix的主要用途是一般性的MTA，而非滤信软件。

Postfix如何与外部的过滤程序接轨？基本上，这要看过滤程序本身的运行方式：是实时启动的过滤命令还是常驻的daemon。对于前者，Postfix通过Unix管道传出邮件（过滤命令是从其标准输入端得到邮件），而且每处理一封邮件，过滤命令就会被启动一次，所以运算成本相当高。对于后者，Postfix通过SMTP或LMTP交出邮件，所以工作效率比较高，运算成本也相对较低，因为所需的系统资源比较少。虽然daemon方法的设定工作稍微复杂些，但是却能提供比较稳健的解决方案。

基于命令的过滤

本节介绍如何让Postfix在收下邮件后，启动一个外部过滤程序，通过管道（即pipe）将邮件传给该程序来进行过滤。过滤程序从自己的标准输入端得到邮件，完成检查之后，使用Postfix的sendmail命令将邮件传回队列系统。

为了方便讨论，我们假设过滤程序只处理来自 SMTP daemon（smtpd） 的邮件，而不处理本地自发的邮件（使用 sendmail 命令），这项假设使得过滤程序可使用 sendmail 将邮件传回 Postfix而不至于造成邮件循环。图 14-1 描述了 Postfix 与外部过滤程序整合在一起之后，邮件的处理流程。相较于正常的流程， Queue manager 这次不直接将邮件交给 MDA，而交给 pipe 所启动的外部过滤程序。

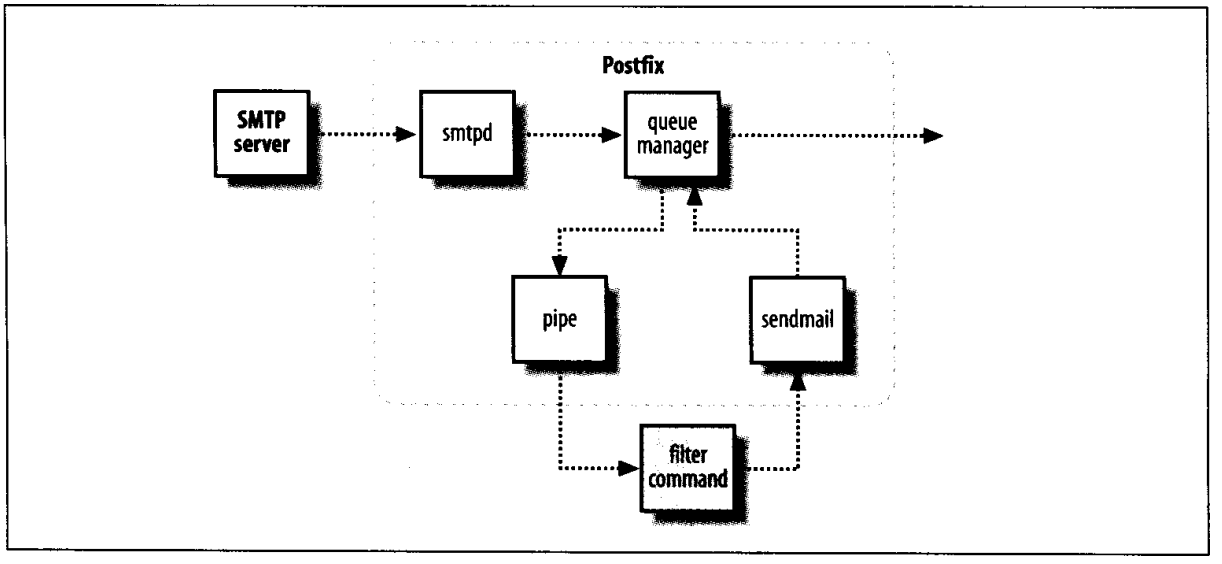

图 14-1 邮件过滤命令

外部过滤程序必须要能够从自己的标准输入端接收信息，以 Postfix 的 sendmail 命令传回处理过的信息。如果你的过滤程序不是依照这种方式来输入、输出信息，则必须另外写一个封装程序，使得 Postfix与过滤程序能顺利衔接。这样的封装程序应该不难写，用简单的 shell script 就可以完成了。Postfix 包随附的 *FILTER_README*说明文件有一个封装程序范例可供参考。

配置

当 Postfix启动外部过滤程序时，必须知道该程序应该继承哪一个系统账户的权限。你可以创建一个专供过滤程序使用的虚账户。

假设我们的过滤程序存放在 */usr/local/bin/* 目录下，其文件名为 simple_filt,而我们为此程序准备的虚账户是 filterman。第一步 ,我们必须编辑 *master.cf* 配置文件，加入下列内容:

```
filter    unix  -       n       n       -       -       pipe
    flags=Rq user=filterman argv=/usr/local/bin/simple_filt
    -f ${sender} -- ${recipient}
```

这三行其实是逻辑上的同一行（因为第二行与第三行的开头字符都是空格）。第一行包含 Postfix 组件的所有标准设定值，此行的最后一栏指出邮件应该交给 pipe MDA来处理。第二行与第三行是 pipe 的自变量，其中 flags=Rq 表示 pipe应该安插一个 Return-Path: 字段到标题，并处理 ${sender}与 ${recipient}地址中的特殊字符与空格，使其适合传给外部命令。关于其他选项，请参阅 pipe(8)在线说明书。

"user="选项指出过滤程序的运行身份，在此例中，我们将它设定为预先准备的 filterman 虚账户。"argv"选项指出过滤程序本身的命令行自变量。这里所给定的自变量行（-f ${sender} -- ${recipient}),就是封装程序在调用 sendmail 命令将邮件传回 Postfix时，所使用的命令行自变量。当然，不同的过滤程序可能需要不同的自变量,但是请务必提供 sendmail命令中将邮件传回 Postfix所需的自变量。pipe daemon 会将 ${recipient}变量展开成实际收件人地址，如果有多位收件人，则最多只展开到 filter_destination_recipient_limit参数所定义的上限。

除了增添新的组件之外，我们还得要修改 *master.cf*配置文件中关于 smtpd服务的设定值，使其将收到的邮件交给过滤程序来处理：

```
smtp      inet  n       -       n       -       -       smtpd
  -o content_filter=filter:
```

如你所见，相较于默认值，我们只是多加了第二行而已（再次提醒，开头的空格是必须的）。content_filter参数指向新设置的 filter服务。在 *master.cf*（而非 *main.cf*）设定此选项，是因为我们希望过滤"smtpd daemon 所经手的每一封邮件"，而非"进入 Postfix 系统的每一封邮件"。在你重新启动 Postfix之后，从 smtpd收进来的每一封邮件，都会流经 simple_filt过滤程序。

利用 pipe来启动外部过滤程序，虽然很容易设定，但是效率很低。因为每处理一封邮件，Postfix就要调用一次 shell或 interpreter （如果过滤程序本身以 Perl、Python 之类的语言所写成），而且过滤程序还必须调用一次 sendmail，将完成过滤的邮件传回 Postfix 系统。如果你的过滤程序遇到磁盘或内存空间不足的问题，将没有一个可靠的办法让 Postfix知道到底发生了什么问题。相对地，基于守护进程的过滤，提供了比较稳健、有效率的解决方案（注 1）。

基于守护进程的过滤

相较于实时启动的外部过滤程序，基于守护进程的过滤（daemon-based filtering)显然在 I/O 与 CPU使用量方面比较经济，因为每一封邮件都是以同一个进程来处理，而不像

注 1: 其实，效率的高低，绝大部分取决于过滤程序本身所要执行的动作。

外部过滤程序那样，每过滤一封邮件，系统就至少要多创建一个新进程。此外，基于守护进程的过滤与 Postfix 之间的交互架构也比较先进，双方可使用标准的 SMTP 或 LMTP 协议交换邮件。基于守护进程的过滤多半是独立的进程（由系统管理员负责启动），但也可以被设定成 *master.cf* 里的一项服务（由 Postfix 负责启动）。

假设我们希望使用基于守护进程的过滤器来处理全部邮件 —— 包括本机系统以 `sendmail` 寄出的邮件，以及 `smtpd` daemon从外界收进来的邮件，则我们必须在 *master.cf* 配置文件设置一个特殊的 `smtp` client （使其能将邮件传递给过滤程序）以及一个额外的 `smtpd` daemon （使其接收过滤程序传回的邮件）。图 14-2描绘了邮件通过 Postfix到达过滤程序，然后又回到 Postfix 的过程。在此图中，过滤程序在 localhost port 10025 接收我们设置的特殊 `smtp` client 传来的邮件，然后将过滤好的邮件传到位于 localhost port 10026的特殊 `smtpd` server，再回到 Postfix 系统。

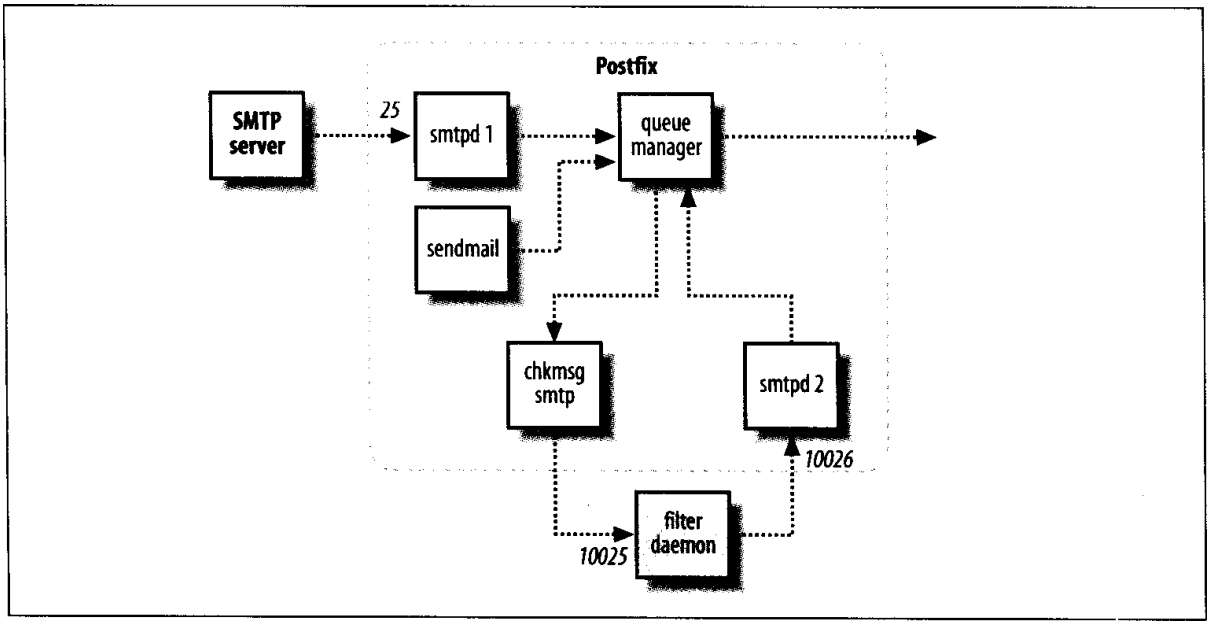

图 14-2 邮件过滤守护进程

如果过滤程序想拒收邮件，它会回复 Postfix 一个 SMTP 550 响应码，必要时，还可以加上一段描述拒绝原因的字符串；否则，它应该收下邮件，并执行它应该做的过滤工作，然后将处理好的邮件传回给 Postfix。当过滤程序拒收邮件时，Postfix 必须发出退信通知函给由过滤程序指定的寄件人地址。

配置

让我们举例示范 Postfix如何与基于守护进程的过滤联接。假设我们的过滤程序是一个独

立型服务器程序，它在特定的 TCP port等待其他程序使用SMTP 协议传送邮件进来。在完成处理之后，同样也是使用 SMTP协议将邮件传回到 Postfix。基本的设定步骤如下：

1. 为过滤程序创建一个专用的虚账户。
2. 安装并设定过滤程序。
3. 编辑 *master.cf*，增加两个额外的 Postfix 组件。
4. 编辑 *main.cf* ，设 定 `content_filter`参数。
5. 重新启动 Postfix ，使我们的改变生效。

设定基于守护进程的过滤程序时，请避免使用与 Postfix的 `myhostname` 参数值相同的主机名称，否则， Postfix SMTP client 会误认为发生了邮件循环，而拒绝将邮件送到过滤程序。接下来的几个小节，分别解释各步骤的细节与注意事项。

步骤一：创建一个虚账户

如同命令型的过滤程序，守护进程型过滤程序也需要一个虚账户。此账户的权限应该不能够访问系统上不该访问的资源。如果需要写文件，则必须为该虚账户准备一个专用目录。你应该以指定的身份来启动过滤程序或是设定该程序，使其在启动之后切换成指定的身份。在本例中，我们假设虚账户的名称是 filterman。

步骤二：安装内容过滤器

这一步骤的详细情况，请参考内容过滤程序的说明文件。在此例中，我们假设过滤器的服务接口是 loopback TCP port 10025 。完成过滤动作之后，过滤器应该透过 loopback TCP port 10026 将邮件传回到 Postfix系统。你的过滤程序应该容许你做类似的设定，如果使用不同的 TCP port 来接收或返回邮件，请依照实际状况修改。如果可能，请先测试你的过滤器是否能正确运作，然后再试着与 Postfix结合在一起。

步骤三：设定额外的 Postfix 组件

当你增加额外的 SMTP 组件时，很可能会遇到“ mail loops back to myself” 问题。解决办法之一，是让Postfix与过滤程序的SMTP组件各自使用不同的`myhostname`参数值。

编辑 *master.cf*，加入所需的新组件。新增的 `smtp`组件将会被用来传送邮件给内容过滤器。关于如何编辑 *master.cf* ，请参阅第四章的“ master.cf配置文件 ”。我们打算将额外的 `smtp`组件命名为 `chkmsg`：

```
chkmsg      unix  -       -       n       -       10      smtp
    -o myhostname=localhost
```

稍后在 *main.cf* 设定内容过滤器时，必须要求 Postfix 使用 chkmsg 组件将邮件传送到位于 TCP port 10025 的内容过滤器。

除了设置额外的 smtp client，我们还需要另一个 smtpd 服务，用以接收过滤器传回的邮件。这个 smtpd的设定方式略为不同于前一个，因为我们希望以不同的处理方式来对待来自外界与来自过滤器的邮件。以下是我们对于另一个 smtpd 服务的设定方式：

```
localhost:10026     inet  n     -     n     -     10     smtpd
        -o content_filter=
        -o local_recipient_maps=
        -o mynetworks=127.0.0.0/8
        -o smtpd_helo_restrictions=
        -o smtpd_sender_restrictions=
        -o smtpd_recipient_restrictions=
        -o smtpd_client_restrictions=permit_mynetworks,reject
```

如你所见，我们将这个 smtpd 服务放在 loopback 接口的 TCP port 10026，所以过滤器必须被设定成将处理好的邮件送回到此通信端口。此外，我们还设定了几个选项，这些选项的效力高于 *main.cf* 所定义的参数。以下是各选项的意义：

content_filter

由于我们稍后要在 *main.cf* 设定 Postfix 默认的 smtpd服务的 content_filter 参数，所以这里必须关闭第二个 smtpd 的内容过滤功能，以免造成邮件循环。

local_recipient_maps

当 Postfix 默认的 smtpd收下邮件后，有可能会依据查询表来转换收件人地址。因为当过滤器交回邮件时， Postfix有可能因不认识收件人地址而拒绝收信。将此选项设定成空白，以确定 Postfix 一定愿意收下过滤器交回的邮件。

mynetworks

由于我们的过滤器与 Postfix是在同一台机器上运行，所以两者可通过本机的 loopback 接口互相通信。loopback 本身是一个无关任何网络硬件的虚构设备，其 IP 地址固定为 127.0.0.1。由于此地址的第一个字节是 127，属于 A 级网络，所以 loopback 网络的掩码为 255.0.0.0（以 /8来表示）。将 mynetworks设定为 loopback 网络，同时限定 smtpd_recipient_restrictions 只接受来自 loopback 网络的服务请求，可使得我们的 smtpd只接受过滤器的联机，而不受外界网络的（恶意）干扰。

smtpd_helo_restrictions, smtpd_sender_restrictions, smtpd_recipient_restrictions

你可以关闭初始 smtpd 所设定的任何限制条件。当然，如果你的 *main.cf* 原本就没设定任何限制条件，就不必关闭它们。

smtpd_client_restrictions

最后，要求 smtpd 只接受来自 loopback 接口的联机请求。

步骤四：启动过滤功能

在 *master.cf* 做了必要的修改之后，我们还得设定 Postfix 将所有收到的邮件都传给过滤器。请编辑 *main.cf* 配置文件，加入以下内容：

```
content_filter = chkmsg:[127.0.0.1]:10025
```

此参数要求 Postfix 使用我们先前在 *master.cf* 设置的 chkmsg 服务，将所有邮件交给内容过滤器。我们必须指出过滤器所在的 IP 地址与通信端口，而这两项设定值都必须符合过滤器的实际配置。一切都完成之后，请重新启动 Postfix，它才会开始将收下的邮件交给内容过滤器。

守护进程型过滤器实例

本小节以 Central Command 公司出品的 Vexira AntiVirus 软件为例，具体示范如何设定一个守护进程型过滤器。Vexira 是一套付费软件，但是你可从他们的网站下载试用版（*http://www.centralcommand.com/*）。Vexira AntiVirus for Mail servers 系列产品可结合多种 MTA，包括 Postfix 在内。这套软件所支持的系统平台包括 Linux、FreeBSD 与 OpenBSD。如果你使用其他 daemon-based 的反病毒产品，其设定过程也应该很类似于此处所展示的步骤。

1. 依照 Command Central 的说明书，将 Vexira 软件安装到 Postfix 服务器上。接下来的步骤假设 Vexira 的配置文件是放在 */etc/* 目录下。
2. 设定 Vexira 的服务接口。假设我们选定的接口是 loopback:10024，请编辑 */etc/vamailarmor.conf* 配置文件，将其 ListenAddress 参数设定成下面这样：

   ```
   ListenAddress localhost port 10024
   ```

3. 接着设定 ForwardTo 参数，使其通过 loopback:10025 将邮件传回 Postfix：

   ```
   ForwardTo SMTP: localhost port 10025
   ```

4. 重新启动 Vexira（详细步骤请参考 Vexira 的说明书）。
5. 编辑 Postfix 的 *main.cf* 配置文件，将所有邮件交给 Vexira daemon 进行病毒检查：

   ```
   content_filter = smtp:[127.0.0.1]:10024
   ```

6. 编辑 Postfix 的 *master.cf* 配置文件，增加另一个 SMTP daemon 来回收 Vexira 处理过的邮件：

   ```
   localhost:10025   inet  n   -   n   -   10   smtpd
           -o content_filter=
   ```

```
-o local_recipient_maps=
-o mynetworks=127.0.0.0/8
-o smtpd_helo_restrictions=
-o smtpd_client_restrictions=
-o smtpd_sender_restrictions=
-o smtpd_recipient_restrictions=permit_mynetworks,reject
```

7. 重新启动 Postfix，使我们所做的改变生效：

```
# postfix reload
```

其他考虑事项

必要的话，你可以同时运行多个过滤程序，诀窍是将它们串接在一起。比方说，我们有一个反病毒的与一个反垃圾邮件的内容过滤器，我们可让病毒过滤器检查第一手的邮件，经过消毒的邮件不立刻转回 Postfix ，而是交给垃圾邮件过滤程序继续检查。在这种情况下，Postfix 同样只需设置一组出入口即可，由最后接手的过滤器将邮件传回给 Postfix。

请留意发生在过滤器收到邮件之前的任何邮件地址转换。当过滤器交回邮件时，如果改写后的地址不在有效收件人列表里， Postfix 将予以拒收。你可能需要关闭 Postfix 默认的 SMTP server 的地址改写动作，改由回收过滤邮件的那个 SMTP server 来进行地址改写。

某些过滤器建议你将它们设置在正常 MTA之前，也就是说，过滤器本身能扮演 SMTP server 的角色，而它们会将处理完毕的邮件交给 MTA。就 Postfix 而言，我们并不建议你这样做，因为 Postfix 是特地为险恶网络环境而设计的 MTA，而过滤器擅长的是处理邮件内容，它们不一定能妥善应付来自外界的潜在攻击。类似地，某些过滤器能将邮件投递到最终目的地，而不回交给 Postfix。我们也不建议这样做，因为 Postfix 的最终投递程序提供了许多灵活性与安全性，如果改由其他软件进行最终投递，结果可能得不偿失。

第十五章

外部数据库

在处理电子邮件的过程中，经常需要执行各式各样的转换与查表操作，如检查收件地址是否有效、将别名改写成规范名称等。就应付各式各样的查表操作而言， Postfix 本身的查询表已经相当完美，不但容易使用，而且效率也令人满意。然而，在某些情况下，将信息存放在 Postfix 之外的独立数据库会比较方便些。举例来说，有许多系统或网络服务都需要同样的全局性信息，诸如账户、密码、个人数据等，将这类信息集中在一个独立的数据库，有助于信息的集中管理与维护。此外，数据库也可以用来存储配置信息。对于备用的 Postfix 系统或是由多人共同维护的 Postfix 系统而言，将配置信息集中在数据库是最理想的。

相较于平常的 DB 索引文件，外部数据库会降低 Postfix 的效率。一般而言，如果不是非得用数据库不可，最好还是使用 Postfix 的标准查询表。或者，你可以采取折衷方案：将信息集中存放在数据库，另外写一个 cron script 来定期读取数据库，并更新 Postfix 的查询表。如此可享受信息集中控制管理的好处，却又不至于让 Postfix 的效能受到数据库系统的影响。不过，如果你的环境需要能够立刻访问修改过的数据而不容许任何时间差存在，那么，直接使用外部数据库可能是你唯一的选择。

本章将示范 Postfix 如何与 MySQL 与 LDAP 合作（未来的 Postfix 2.1 可能会支持 PostgreSQL）。Postfix 的默认配置不支持 `mysql` 与 `ldap` 查询方式，除非你在编译 Postfix 的过程中，与额外的支持库链接。关于如何建构出一个支持 MySQL 或 LDAP 的 Postfix，请参阅附录三。如果你使用别人预先编译好的 Postfix 包，请确认你的版本支持 MySQL 或 LDAP。

要想知道自己的 Postfix 是否支持 LDAP 与 MySQL，只要观察 `postconf -m` 命令的输出信息中是否包含 `mysql`、`ldap` 字样即可：

```
$ postconf -m
static
pcre
regexp
mysql
environ
proxy
ldap
btree
unix
hash
```

虽然被用来搭配 Postfix 的数据库可能含有各式各样的信息，但不管是基本的“索引键一对应值”配对概念或是在用途上，外部数据库并无异于一般的 Postfix 查询表。你同样可将收件人地址之类的数据当成索引键，并期待得到转寄地址之类的对应值。接下来的两节，分别解释如何对 MySQL 与 LDAP 这两种截然不同的数据库系统进行查询。

在开始使用外部数据库之前，建议先使用 Postfix 的标准查询表来验证查询工作是否正确无误，然后再改用 MySQL 或 LDAP 来查询，并确定两者都能返回同样的结果。在大部分情况下，Postfix 希望每次查询只会返回一个结果，所以，你得确定数据库不会返回多个结果。

MySQL

MySQL 是一套“关系数据库管理系统”（Relational Database Management System，RDBMS），让用户能以“结构化查询语言”（Structured Query Language，SQL）来查询、管理数据。即使你不懂 SQL，也不妨碍你在 Postfix 里使用 MySQL，但是，如果你懂 SQL，将有助于理解 Postfix 与 MySQL 之间是如何交互的。一般来说，想到使用 MySQL，是因为 Postfix 所需的数据早已经被存放在现有的数据库中。比方说，用户的个人数据（姓名、邮件地址、住址、电话号码、账户名称等）已经全部都存放在数据库里了，在这种情况下，直接让 Postfix 查询数据库，显然比浪费一些空间来存储相同数据要好得多。当然，前提是数据库确实能提供 Postfix 完成特定工作所需的一切信息。外部数据库的常见用途之一，是提供邮件别名与本地账户名称之间的对应关系，对于这种用途，数据库至少要有一个存储邮件别名的字段以及一个存储本地账户名称的字段。Postfix 能以邮件收件人地址为索引键，从你的数据库查出其对应的本地账户名称，借此决定邮件应该被投递到哪一个邮箱。

MySQL 配置

任何可使用 Postfix 查询表的参数，都可以用 MySQL 数据库来达成相同效果，前提是你必须知道哪些字段含有你所需的信息。 MySQL map 的指定方式，就像任何其他类型的

查询表一样，形式上唯一的差异是你必须将 hash（或 static、pcre、regexp）换成 mysql，例如：

```
alias_maps = mysql:/etc/postfix/mysql-aliases.cf
```

不过，这里指定的 *mysql.aliases.cf* 文件所包含的不是直接可查的数据，而是描述如何查询 MySQL 数据库的配置信息。下一小节解释此配置文件需要设定哪些参数。

MySQL 参数

MySQL 参数提供了 Postfix 向 MySQL server 查询数据时所需的配置信息，包括如何联机到 MySQL server 以及如何构造出适当的 SQL 语句来查询所需的数据。MySQL 参数必须集中在 MySQL map 所指定的配置文件，此配置文件的格式与 Postfix 的 *main.cf* 一样，所以不再赘述。你可以同时设置多个 MySQL 配置文件，就像你可以在 Postfix 中同时使用许多查询表一样。每一个 MySQL 配置文件，都要含有图 15-1 所述的全部参数，唯一例外是 additional_conditions。

图 15-1 显示了各个 MySQL 参数与 SQL 语句结构的对应关系。

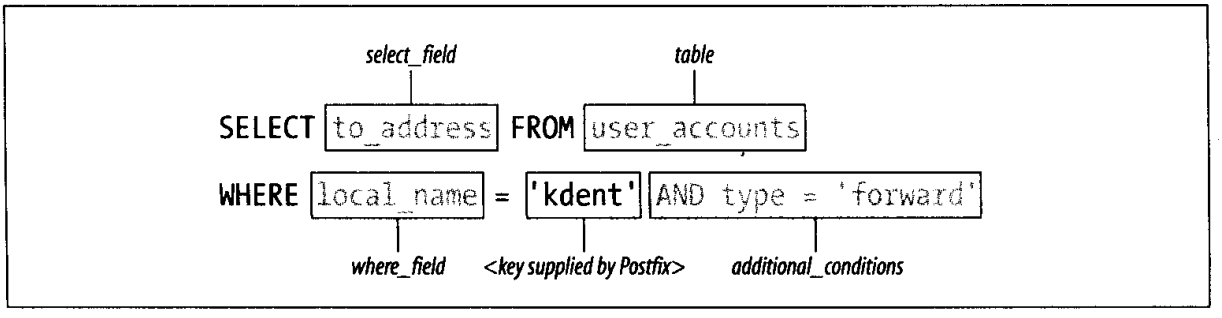

图 15-1：SQL 语句的结构

hosts

MySQL 所在的主机名称或 IP 地址。如果 MySQL 与 Postfix 位于同一台机器，你可以使用 unix 指出 Unix-domain socket 的完整路径。例如：

```
hosts = unix:/tmp/mysql.sock, db.example.com, 192.168.150.15
```

除非你有多台备用的 MySQL server，否则列出一个位置就应该够了。如果你列出多个位置，Postfix 将依照你列出的顺序逐一查询，直到查询成功为止。

user

用于登录 MySQL server 的账户名称。此账户必须是有效的 MySQL server 账户（不是 MySQL 所在主机的系统账户），而且要有足够的权限来查询所需数据。

password

用于登录 MySQL server 的密码。

dbname

要被查询的数据库的名称。

table

要被查询的数据表的名称。

select field

含有“对应值”的字段的名称。

where field

含有“索引键”的字段的名称。

additional_conditions

额外的筛选条件。如果设定此参数，当 Postfix 构造 SQL 语句时，会将你指定的参数值当成 WHERE 子句。因此，当你设定此参数时，必须假设自己是在延续 SQL 语句，例如：

```
additional_conditions = and mail_type = 'local'
```

你必须了解 SQL，才能知道如何使用此参数。

MySQL 范例

让我们以具体实例示范如何让 Postfix 能够查询 MySQL 数据库。假设 *example.com* 网域的所有用户信息都是集中在一个 MySQL 数据库，此数据库包含了多个数据表，分别记录该网络每一位用户的各种信息，包括姓名、电话号码、邮件地址等。其中，记录邮件地址的数据表是 E-mail_address，此表含有设定 Postfix 所需的相关信息。以下是 E-mail_address 数据表的结构：

```
+-----------------+-------------+------+-----+---------+-------+
| Field           | Type        | Null | Key | Default | Extra |
+-----------------+-------------+------+-----+---------+-------+
| localpart       | varchar(15) |      | PRI |         |       |
| type            | varchar(15) | YES  |     | NULL    |       |
| to_address      | varchar(65) | YES  |     | NULL    |       |
| password        | varchar(65) | YES  |     | NULL    |       |
| last_changed_by | varchar(15) | YES  |     | NULL    |       |
+-----------------+-------------+------+-----+---------+-------+
```

此表的 localpart 字段，可供 Postfix 用来判别收件地址是否有效。某些用户的主要邮件账户是位于其他系统，而他们在 *example.com* 的地址其实是主要邮件账户的别名而已。type 字段供 Postfix 用来判别 localpart 是否为其他账户的别名，如果 type 字段的值为 forward，表示 Postfix 应该将收下的邮件转寄到 to_address。

在开始设定 Postfix之前，应该先搜集关于如何访问数据库服务器的信息。表 15-1 是 Postfix访问此 MySQL server 所需的配置信息。

表 15-1：Postfix 访问 MySQL数据库所需的信息

项目（参数名称）	值
主机名称（hosts）	mysql.example.com
数据库名称（dbname）	user_accounts
数据表名称（table）	E-mail_address
数据库账户（user）	kdent
数据库密码（password）	Rumpelstiltskin

除了要搜集表 15-1所列的信息之外，你还必须判断哪个MySQL 数据表拥有足够的字段，可用来取代 Postfix 的特定查询表。在此例中，我们希望使用 E-mail_address 数据表作为 local_recipient_maps 与 alias_maps参数的数据源，因为该表的 localpart、type 与to_address三个字段恰好含有 Postfix所需的信息。例 15-1 是 E-mail_address数据表的其中一笔记录：

例 15-1：E-mail_address 数据表的其中一笔记录

```
+------------+----------+-------------------+
| localpart  | type     | to_address        |
+------------+----------+-------------------+
| kdent      | forward  | kyle.dent@ora.com |
+------------+----------+-------------------+
```

设定 local_recipient_maps

Postfix依据local_recipient_maps参数所提供的列表来判断收件人地址是否有效，借此决定是否应该收下邮件。在默认情况下，此参数指向系统账户文件与别名文件，如此，SMTP server 才能拒收那些送给不明用户的（垃圾）邮件。 local_recipient_maps 所指的查询表，最大的与众不同之处在于它不要求每笔记录都必须成对，因为它只在乎收件地址是否存在于查询表里而已。

在我们的例子中，所有合法本地邮件账户都记录在 MySQL数据库。因此，第一步应该是将 local_recipient_maps 参数指向一个 MySQL 查询配置文件，好让 Postfix 能利用该文件提供的信息，从 MySQL server取得邮件用户的列表。假设我们选定的查询配置文件的名称为 *mysql-local.cf*：

```
local_recipient_maps = mysql:/etc/postfix/mysql-local.cf
```

接着，将表 15-1 的信息填入 *mysql-local.cf* 配置文件，并设定 select_field 与 where_field 分别对应的字段：

```
#
# mysql-local.cf - local recipients for mail server
#
hosts = mysql.example.com
user = kdent
password = Rumpelstiltskin

dbname = user_accounts
table = E-mail_address

select_field = localpart
where_field = localpart
```

select_field 与 where_field 两者应该都同样指向 localpart 字段。对于 local_reipient_maps 参数而言，其实 select_field 并不重要，因为该参数不需要返回值。此处不需要设定 additional_conditions参数，因为我们想查询的对象，是 E-mail_address 数据表里的每一笔记录。在重新启动 Postfix 之后，它就能依据 *mysql-local.cf* 提供的信息来访问 MySQL数据库，取得有效账户的列表，拒绝收件地址没有出现在 E-mail_address 数据表中的邮件。

使用 postmap 命令可测试我们的 MySQL配置是否正确：

```
$ postmap -q 'kdent' mysql:/etc/postfix/mysql-local.cf
kdent
```

"-q"选项要求 postmap 使用指定的索引键来查表，如果出现问题， postmap 会回复查询过程遇到的状况。

设定 alias_maps

某些 *example.com* 的用户并非从 *example.com* 的邮件系统收信，他们的邮件必须被转寄到另一个邮件地址。将 alias_maps 指向另一个 MySQL查询配置文件（假设是 *mysql-alias.cf*），我们可查出哪些用户具有别名以及他们的转寄地址。首先，将 alias_maps参数指向 *mysql-alias.cf* 配置文件：

```
alias_maps = mysql:/etc/postfix/mysql-alias.cf
```

mysql-alias.cf 查询配置文件含有下列参数：

```
#
# mysql-alias.cf - forwarding aliases
#
```

```
hosts = mysql.example.com
user = kdent
password = Rumpelstiltskin

dbname = user_accounts
table = E-mail_address

select_field = to_address
where_field = localpart

additional_conditions = and type = 'forward'
```

在此例中，select_field 参数应该指向 to_address 字段，因为该字段含有 alias_maps 所需的转寄地址。此外，我们还必须使用 additional_conditions 参数将查询范围局限在 type栏的值为‘ forward ”的记录上，也就是排除没有别名的账户。重新启动 Postfix 之后，它就应该能使用 MySQL 提供的信息来判断别名与转寄地址。

架设虚拟网域

代管许多虚拟网域的大型服务器系统,经常将网域名称、账户数据等信息记录在MySQL数据库。我们最后的 MySQL 范例，要示范如何使用 MySQL来支持虚拟网域。本节只讨论关于 MySQL 配置的部分，至于如何架设虚拟网域，请参阅第八章。

为了支持虚拟网域，我们建了一个 customer数据库，并将所有虚拟地址都记录在此数据库中的 E-mail_address 数据表中。构成此表的字段如下：

domain
: 虚拟网域名称。相当于收件地址的“ 网域部分 ”。

mail address
: 用户的公关邮件地址。 从外界寄到此地址的邮件，会被投递到虚拟邮箱。

mailbox
: 虚拟邮箱的相对路径（相对于 virtual_mailbox_base所指的路径）与文件名。如果想使用 maildir 格式的邮箱，请在文件名末端加一个“ /” 符号。

例 15-2是 E-mail_address数据表中的一笔记录：

例 15-2： 支持虚拟 网域的数据表样本

```
+------------+---------------+---------------+
| domain     | mail_address  | mailbox       |
+------------+---------------+---------------+
| ora.com    | kdent@ora.com | ora.com/kdent |
+------------+---------------+---------------+
```

假设所有虚拟邮箱（无论哪一个网域）的投递工作，都是以vmail:vmail这组系统账户与组来进行的，而且其UID与GID同样都是1003。如果你需要以不同的权限组合来自不同网域或用户的邮件，你可能需要在E-mail_address数据表里增添两个字段，分别记录投递工作所用的UID与GID。

第一步，先设定已知固定值的参数，包括虚拟邮箱的基础目录（virtual_mailbox_base）以及用于执行虚拟邮箱投递操作的身份：

```
virtual_mailbox_base = /usr/local/vmail
virtual_uid_maps = static:1003
virtual_gid_maps = static:1003
```

第二步，设定需要从MySQL取得信息的参数。包括虚拟网域列表以及邮件地址与虚拟邮箱的查询表：

```
virtual_mailbox_domains = mysql:/etc/postfix/virtual_domains.cf
virtual_mailbox_maps = mysql:/etc/postfix/virtual_mailboxes.cf
```

以下是*virtual_domains.cf*配置文件的内容，由于Postfix只需知道收信网域是否存在而已，所以select_field与where field都是指向domain字段：

```
hosts = mysql.example.com
user = kdent
password = Rumpelstiltskin
dbname = customer
table = E-mail_address
select_field = domain
where field = domain
```

*virtual_mailboxes.cf*配置文件要让Postfix能从收件地址（mail_address字段）查出对应的虚拟邮箱（mailbox字段）：

```
hosts = mysql.example.com
user = kdent
password = Rumpelstiltskin

dbname = customer
table = E-mail_address
select_field = mailbox
where_field = mail_address
```

LDAP

LDAP是用于访问目录（directory）信息的协议。LDAP目录是一种层次结构式的树状数据结构，通常用于存储人事信息。你必须了解LDAP的运作原理以及你的LDAP目录的组织结构，才能了解如何在Postfix使用LDAP。目前已经有越来越多的网络开始使用

LDAP来记录人事信息，因此，直接让 Postfix 从 LDAP 取得用户列表或有效地址列表，可避免同样的信息放于两地，减轻系统管理员的工作负担。如果你的公司有现成的 LDAP 目录，你可以要求 Postfix 查询现成的信息。

LDAP 配置

如同 `mysql:` 一样，任何可以使用 `hash:` 查询表的参数，都可以用 `ldap:` 来代替。然而 `,mysql:`是指向一个含有 MySQL 参数的文件，而 `ldap:` 却是指向一个代表某 LDAP 参数组合的配置名称。 LDAP 的配置参数并非放在各自独立的文件中，而是全部都集中在 *main.cf* 配置文件。不同组的 LDAP 配置，各有一个代表该配置的名称。你得为每一个 LDAP 配置起一个名称，并以此名称作为该组参数的名称的前缀。举例来说，假设某个 LDAP 配置的名称为 `ldapaliases`：

```
alias_maps = ldap:ldapaliases
```

定义 LDAP server 主机名称的参数是 *name*`_server_host`，所以，此参数在 `ldapaliases` 配置里的名称是 `ldapaliases_server_host`：

```
ldapaliases_server_host = ldap.example.com
```

以下是最重要的几个 LDAP 参数。如果你需要完整的参数列表，请参考 Postfix 包随附的 *LDAP_README* 文件。

name `search base`

搜索起点的辨别名称（DN）。你得知道目录的命名环境（naming context），才可能知道目标数据的共同起源。如果你真的不知道搜索起点，那就从目录的“根”开始找。范例：

```
ldapaliases_search_base = dc=example, dc=com
```

name`_scope`

搜索范围。有三种可能范围可选择：`sub`、`base` 与 `one`。选择哪一种范围，要依你的目录的结构而定。`base` 只搜索起点本身，`sub` 搜索起点以下的整棵子树，`one` 只搜索直接子节点（搜索起点的下一层）。`name_scope` 参数的默认值是 `sub`。范例：

```
ldapaliases_scope = one
```

name`_query_filter`

构成搜索条件的属性与值。变量 `%s` 用来代表收件人邮件地址。范例：

```
ldapaliases_query_filter = (mailType=forward)
```

*name*_result_attribute

含有搜索值的属性名称。你可以按照优先级列出多个属性。范例：

```
ldapaliases_result_attribute = E-mail, rfc822Mailbox.
```

LDAP范例

LDAP在Postfix系统上最重要的应用，莫过于核验收件地址的有效性，以及提供邮件地址与收信主机之间的对应关系。举例来说，假设*example.com*是一个大型网域，该网络上的网关上有一台Postfix server，其内部有多个邮件系统，分别服务不同组的用户。此网域的所有用户的人事数据都收集在同一个LDAP目录里，而且所有邮件系统都可从这个LDAP目录取得一致的人事信息。在这种情况下，网关上的Postfix server可将自己的local_recipient_maps参数指向LDAP目录，借此判断外来邮件所指定的收件人是否确有其人（以免遭受垃圾邮件发送者的字典攻击）。此外，它也可将transport_maps参数指向LDAP目录，使其能将收到的邮件转送到正确的内部邮件服务器。

假设用户的公共邮件地址是记录在LDAP目录里的mail属性，而其内部收信主机是记录在mailHost属性（网关系统必须将用户的邮件转送到mailHost所指的服务器）。以下是某人在LDAP目录里的人事数据样本：

```
dn: uid=kdent,ou=people,dc=example,dc=com
uid: kdent
cn:  Kyle D. Dent
mail: kyle.dent@example.com
uidNumber: 1001
gidNumber: 1001
mailHost: mail1.example.com
homeDirectory: /home/kdent
mailType: forward
objectClass: people
userPassword: {crypt}hidden
accountStatus: active
```

表15-2是Postfix需要从LDAP目录取得的数据项。在你开始设定Postfix之前，你应该先查出LDAP server的主机名称，以及目标信息的搜索起点（base DN）。

表15-2：在Postfix设定LDAP目录所需的信息

名录信息	值
主机（Host）	ldap.example.com
搜索起点（Base DN:）	dc=example,dc=com

就 local_recipient_maps 参数的作用而言，它只需要知道收件地址的有效性，所以只要查询邮件地址是否存在于 mail 属性中即可；另一方面， transport_maps参数则需要 mailHost 属性的值，才能将邮件转递到正确的收信主机。

设定 local_recipient_maps

Postfix依据 local_recipient_maps 参数所提供的列表，决定是否要收下外来的邮件，此参数的默认值是指向本地系统的账号文件与别名文件。在此例中，有效邮件地址全部都是记录在 LDAP目录，所以 local_recipient_maps应该指向一个 LDAP配置的自定义名称。假设我们取的名称为 ldaplocal：

```
local_recipient_maps = ldap:ldaplocal
```

定义此配置的 LDAP参数，都必须冠上 ldaplocal 前缀，如下：

```
ldaplocal_server_host = ldap.example.com
ldaplocal_search_base = dc=example, dc=com
ldaplocal_query_filter = (&(mail=%s)(accountStatus=active))
ldaplocal_result_attribute = uid
```

上述 LDAP参数的解释如下：ldaplocal_server_host与 ldaplocal_search_base参数分别指出LDAP server 的主机名称与目录搜索起点。 ldaplocal_query_filter参数定出的搜索条件为：收件人地址（ %s ）必须符合目录里的 mail 属性，而且其 accountStatus属性的值必须等于 active。如果能找到符合此条件的记录，则返回该记录的 uid属性。

其实，就 local_recipient_maps 的用途而言，返回的 uid属性值并没有任何实质作用，但是 Postfix 需要一个非空白的值，才能判断收件人邮件地址是否有效。

在重新启动Postfix之后，每当收到外来邮件时，它将依照 ldaplocal 配置的指示，向 LDAP server查询收件人地址的有效性，如果查询结果为空白（表示找不到符合条件的数据），则拒收邮件。

使用 postmap 命令可验证刚设定好的 LDAP配置是否正确：

```
$ postmap -q 'kdent' ldap:ldaplocal
kdent
```

-q 选项要求 postmap 使用指定的索引键（此例中的“ kdent”）来进行查询。如果查询过程发生任何问题，当场便可见到错误信息。

设定 transport_maps

网关上的Postfix收下邮件之后，还必须将邮件传送到正确的内部邮件服务器。提供“收件人邮件地址一收信主机”对应关系的参数是transport_maps。为了实现这项查询服务，我们需要另外设置一个LDAP配置，假设此配置的名称为ldaptransport：

```
transport_maps = ldap:ldaptransport
```

由于LDAP目录只返回收信主机的名称，而transport_maps 参数需要我们设定一个传输值(transport:nexthop)，因此，我们得使用name_result_filter参数来规范查询结果的格式：

```
ldaptransport_result_filter = relay:%s
```

以下是ldaptransport配置的LDAP参数：

```
ldaptransport_server_host = ldap.example.com
ldaptransport_search_base = dc=example, dc=com
ldaptransport_query_filter = (&(mail=%s)(accountStatus=active))
ldaptransport_result_attribute = mailHost
```

此处ldaptransport_query_filter参数所规定的搜索条件，如同先前的ldaplocal_query_filter 参数，所以不再赘述。唯一的差别是这次我们希望返回符合条件的记录的mailHost属性值（用户的收信主机），而且返回值必须被改写成ldaptransport_result_filter所指定的格式。

设定完毕，请记得重新启动Postfix，这样新设定的ldap transport传输表才会开始生效。

附录一

配置参数

本附录按照字母顺序，列出可在 *main.cf* 配置文件里设定的参数，并简略说明各参数的意义或用途。本附录的用意在于提醒读者各个 Postfix 参数的作用，而非完整地说明它们，因为最权威、最完整的信息 —— Postfix 包随附的在线说明与配置文件样本，已经在你手上了。因此，本附录只能算是 Postfix 参数的“速查参考”，如果想要知道各参数的确实用法，请参阅本书正文或是在线文件。

对于每一个参数，我们都会列出其可能值或是设定值的类型，大部分类型的设定值都没有异义，只有一些类型的值需要解释：

逐项列出

参数需要一个或多个可能值。至于某特定参数可能需要哪些值，请参阅在线说明。

查询表

当参数指向查询表时，必须指出表的类型与名称，两者之间以一个冒号隔开。例如：

```
transport_map = hash:/etc/postfix/transport
```

路径名称

一个文件的完整路径。

格式样板

某些参数的设定值，是由包含宏的字符串所组成，例如：

```
smtpd_banner = $myhostname ESMTP $mail_name
```

宏（此例中的 `$myhostname` 与 `$mail_name`）是在运行时才展开的对应的参数值。至于哪些参数的样板可接受哪些宏，请参阅在线说明文件。

计时单位

Postfix 有许多涉及时间的参数，例如：

```
queue_run_delay = 1000s
```

为了方便描述时间，Postfix拟定了一组通用的计时单位简写，如表A-1所示。如果没注明计时单位，每个时间参数都有自己默认的计时单位。至于特定时间参数的默认计时单位是什么，请参阅在线说明。

表A-1：计时单位

单位	简写	范例
秒	s	1s
分	m	15m
时	h	4h
日	d	5d
周	w	2w

所有参数都有一个默认值（虽然某些默认值是空白）。只有实际设定值不同于默认值的参数时，才需要被列在 *main.cf* 中。本篇附录虽然列出了参数的默认值，但是不保证新版本的Postfix一定不会改变默认值。使用 `postconf`命令的 `-d`选项，可以查出特定参数的默认值。

```
$ postconf -d alias_maps
alias_maps = hash:/etc/aliases, nis:mail.aliases
```

Postfix 参数速查参考

2bounce_notice_recipient

可能值：邮件地址　　默认值：postmaster

“2bounce”是多种可能的错误类型之一。每一类错误，都可由你决定是否要发出通知函。`2bounce_notice_recipient` 指定“2bounce”错误类型的通知对象。

范例：　`2bounce_notice_recipient = postmaster`

access_map_reject_code

可能值：响应码　　默认值：554

Postfix因为访问表条件限制而拒收邮件时，返回给客户端的SMTP响应码。

范例：　`access_map_reject_code = 554`

alias_maps

可能值：别名表　　默认值：hash:/etc/aliases, nis:mail.aliases

列出 local MDA 所使用的别名数据库。

范 例：　alias_maps = hash:/etc/aliases, nis:mail.aliases

allow_mail_to_files

可能值：逐项列出　　默认值：alias,forward

当 local MDA 在展开别名文件时，禁止或容许 local MDA 将邮件投递到外部文件。

范例：　allow_mail_to_files = alias, forward

allow_percent_hack

可能值：yes或 no　　默认值：yes

在 DNS 尚未普及之前，寄信方可用特殊格式的收件地址来影响邮递路径，这种技术称为“percent hack”或“sender specific routing”。现在，DNS 与邮递路径的选择已经很可靠了，但是 Postfix 仍然持续支持这项技术。若想要关掉这项功能，可将此参数设定为 no。

范例：　allow_percent_hack = no

alternate_config_directories

可能值：目录　　默认值：无

此参数要求 postqueue 和 postdrop 命令从指定的目录读取 Postfix 配置文件。你打算使用的每一个非标准目录，都必须列在此参数中。

范例：　alternate_config_directories = /usr/local/postfix/conf

append_at_myorigin

可能值：yes或 no　　默认值：yes

是否以 myorgin 的值补齐只含人名部分的不完整邮件地址。比方说，将 user 改成 *user@host.example.com*。

范例：　append_at_myorigin = yes

authorized_verp_clients

可能值：主机名称或网域名称　　默认值：$mynetworks

VERP 是供 MLM 用来处理退信的功能。此功能以一个特殊的分隔字符，将“列表拥有者地址”与“原收件人地址”结合在一起。 authorized_verp_clients列出可使用此功能的网域名称或客户端的 IP地址。

范例：　　authorized_verp_clients = $mynetworks

berkeley_db_read_buffer_size

可能值：字节　　默认值：131072

读取 Berkeley DB （hash:）或 btree 查询表时，所用的缓冲区数量。

范例：　　berkeley_db_read_buffer_size = 131072

biff

可能值：yes 或 no　　默认值：yes

biff 是一个能在收到新邮件时，发出通知给本地用户的小程序。如果你的用户不会登录邮件系统，建议你关掉 biff 通知，因为这有助于提升邮件系统的效率。

范例：　　biff = no

body_checks_size_limit

可能值：字节　　默认值：51200

限制 body_checks 的检查范围。注意，不是大小超过此数值的邮件就完全不检查，而是不检查超出此数值范围的部分。因此，如果有一封邮件的正文大小是 100 KB，则只有前半部会被检查。

范例：　　body_checks_size_limit = 51200

bounce_service_name

可能值：master.cf 定义的服务名称之一　　默认值：bounce

无法寄出邮件时，master daemon 将状态信息填入日志文件的服务的名称。通常没必要修改此参数。

范例：　　bounce service name = bounce

canonical_maps

可能值：各类型的查询表　　默认值：无

指出规范映射表的类型与位置。规范映射表定义“内部地址格式”与“规范地址格式”的对应关系， Postfix 依据规范映射表，决定如何改写寄件人的邮件地址。

范例：　　canonical_maps = hash:/etc/postfix/canonical_maps

command_directory

可能值：目录　　默认值：/usr/sbin

Postfix 的命令行管理工具（postcat、postqueue 等）的存放目录。

范例：　　command_directory = /usr/sbin

command_time_limit

可能值：计时单位　　默认值：1000s

当 local MDA 将邮件交给外部命令后， Postfix 可容许外部命令有多长的运行时间。

范例：　　command_time_limit = 1000s

content_filter

可能值：传输服务　　默认值：无

将邮件传给内容过滤器的传输服务。 Postfix 只负责将邮件传给指定的传输服务，该传输服务要负责将邮件传给内容过滤器。内容过滤器可以将邮件传给其他过滤器，或是交回给 Postfix 系统。

范例：　　content_filter = myfilter

daemon_timeout

可能值：计时单位　　默认值：18000s

Postfix daemons 处理一次服务的限制时间。如果处理时间超过限制，这些 daemon 会自动结束。

范例：　　daemon_timeout = 18000s

debug_peer_list

可能值：主机名称或网域名称　　默认值：无

当 Postfix 与特定的主机交互时，可特别提升日志记录的详细程度，以提供足够的调试信息。debug_peer_list 可列出一个或多个你想要检查的主机、网域或正则表达式。信息详细程度的提升定义在 debug_peer_level 参数中。

范例　　debug_peer_list = example.com, mail.ora.com

default_destination_concurrency_limit

可能值：纯数值　　默认值：20

传输服务（定义于 *master.cf*）对于同一目的地同时投递量的上限。如果没明确规定上限，则以 default_destination_concurrency_limit 为默认值。请注意，此参数的限制对象是“目的地”，而非“传输服务”本身。决定各个传输服务同时投递量的上限的参数是 default_transport_concurrency_limit 。

范例：　　default_destination_concurrency_limit = 20

default_extra_recipient_limit

可能值：纯数值　　默认值：1000

当 queue manager 以优先度较高的传输服务排挤优先度较低的 MDA 时，该传输服务可处理的收件人上限。

范例：　　default_extra_recipient_limit = 1000

default_process_limit

可能值：纯数值　　默认值：100

任何传输服务可同时启动的进程上限。如果你不要刻意限制各个传输服务的进程数，则以 default_process_limit 参数的值为限制。请注意，此参数的限制对象是各个“传输服务”,如果要针对个别‘目的地’做限制,应该使用 default_destination_limit 参数。

范例：　　default_process_limit = 100

default_recipient_limit

可能值：纯数值　　默认值：10000

限制 queue manager 为各个传输服务保留在内存里的收件人数量。

范例：　default_recipient_limit = 10000

default_verp_delimiters

可能值：字符　　默认值：+=

VERP 是供 MLM 用来处理退信的技术。此技术以一个特殊的分隔字符，将“列表拥有者地址”与“原收件人地址”结合在一起。 default_verp_delimiters 参数决定哪些字符可供 VERP用来构造要被传回的地址。

范例：　default_verp_delimiters = +=

defer_service_name

可能值：master.cf定义的服务名称之一　　默认值：defer

暂时无法寄出邮件时， master daemon用来交付状态信息并维护日志文件的服务。你通常不需要改变此参数。

范例：　defer_service_name = defer

delay_notice_recipient

可能值：邮件地址　　默认值：postmaster

“delay”是多种可能的错误类型之一。每一类错误，都可由你决定是否要发出通知函。此参数决定“ delay”错误类型的通知对象。

范例：　delay_notice_recipient = postmaster

deliver_lock_attempts

可能值：纯数值　　默认值：20

当 Postfix 执行投递操作时，最多可以尝试几次取得邮箱文件的独占锁定。

范例：　deliver_lock_attempts = 20

disable_dns_lookups

可能值：yes或 no　　默认值：no

当 Postfix 决定邮件的递送目的地时，通常先查询收信网域的 DNS MX 记录。如果 disable_dns_lookups成立，则 Postfix不查询 MX记录而直接将邮件递送到收信网域的 A 记录所指的 IP 地址。

范例：　disable_dns_lookups = no

disable_mime_output_conversion

可能值：yes 或 no　　默认值：no

当远程系统没有声明支持 8-bit MIME时，Postfix 会自动将 8-bit MIME格式转换成 7-bit 格式。将 disable_mime_output_conversion 设定成 yes，将取消这项转换。

范例：　disable_mime_output_conversion = nc

disable_vrfy_command

可能值：yes或 no　　默认值：no

通常 Postfix容许客户端使用 SMTP VRFY命令。将disable_vrfy_command设定成yes，可避免外界使用 VERY 命令来侦测收件地址的有效性（垃圾邮件发送者常用这种技巧来收集邮件地址）。

范例：　disable_vrfy_command = no

double_bounce_sender

可能值：邮件地址　　默认值：double-bounce

无法将退信通知函寄达原寄件人时，系统就会产生“双退信通知”，并将此通知函寄到 double_bounce_sender 参数指定的地址。该指定地址不应该作任何其他用途，因为寄到该地址的邮件，一律会被偷偷丢弃。

范例：　double_bounce_sender = double-bounce

empty_address_recipient

可能值：邮件地址　　默认值：MAILER-DAEMON

无法递送出含有空寄件人地址（即<>）的邮件时，退信通知函的收件地址。例如，发送

退信通知函时（这类邮件的寄件人地址是空的），如果无法寄达，则将该通知函改寄到empty_address_recipient所指定的地址。

范例：　empty_address_recipient = MAILER-DAEMON

error_service_name

可能值：master.cf定义的服务名称之一　　默认值：error

无法递送邮件时，master daemon用来产生错误报告的服务。通常你不需要改变此参数。

范例：　error_service_name = error

export_environment

可能值：环境变量　　默认值：TZ MAIL_CONFIG

要被导出到外部进程（比方说，启动 pipe服务或外部命令时）的所有环境变量。

范例：　export_environment = TZ, MAIL_CONFIG

fallback_relay

可能值：主机名称或网域名称　　默认值：无

如果找不到或无法到达正常的目的地，则将邮件转交给此参数列出的 IP地址、主机或网域。

范例：　fallback_relay = example.com

fast_flush_domains

可能值：主机名称或网域名称　　默认值：$relay_domains

速清服务（fast flush）让 queue manager可以应管理员要求，立刻尝试送出特定网域的累积邮件。 fast_flush_domains参数列出适合速清服务的 IP地址、主机以及网域。

范例：　fast_flush_domains = $relay_domains

fast_flush_refresh_time

可能值：计时单位　　默认值：12h

速清服务（fast flush）让 queue manager 可以应管理员要求，立刻尝试送出特定网域的累积邮件。fast_flush_refresh_time 参数定义自动速清操作之间的间隔时间。

范例：　fast flush refresh time = 12h

fork_attempts

可能值：纯数值　　默认值：5

Postfix 要 fork （衍生）出一个新进程时，最多可以尝试几次。

范例：　fork_attempts = 5

forward_expansion_filter

可能值：字符　　默认值：（参阅范例）

设定路径名称给 forward_path参数时，可以使用 $user这种由 Postfix展开的宏，它随状况决定当时邮件的投递路径。 forward_expansion_filter参数列出展开宏时所容许的字符。不容许的字符，会被替换成下划线字符（_）。

范例：　forward_expansion_filter =
1234567890!@%-_=+:,./abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ

hash_queue_depth

可能值：纯数值　　默认值：1

为了组织队列文件，Postfix为每一个队列都建 ſ一个子目录结构。hash_queue_depth 参数代表子目录结构在队列目录下的层数。

范例：　hash_queue_depth = 1

header_address_token_limit

可能值：纯数值　　默认值：10240

限制标题里的地址最多有多少个记号（tokens）可被 Postfix 改写。超过的记号一律予以丢弃。注意：依据 RFC 2822 的定义，单词（word）、“@”与“.”符号，都是记号。

范例：　header_address_token_limit = 10240

header_size_limit

可能值：字节　　默认值：102400

限制邮件标题可包含的字符（字节）数量。超过的部分会被丢弃。

范例：　header_size_limit = 102400

home_mailbox

可能值：路径名称　　默认值：无

Postfix通常是将邮件投递到邮箱目录下的邮箱文件。此参数可将邮箱文件的位置，改成用户主目录下的指定位置。home_mailbox 的值，是以用户主目录为起点的相对路径，若在邮箱文件名称末端附加一个／符号，表示你想要使用 maildir 格式的邮箱。

范例：　home_mailbox = Mail/mbox

ignore_mx_lookup_error

可能值：yes 或 no　　默认值：no

当 Postfix 向 DNS server 查询 MX 记录却得不到响应时，它会隔一段时间之后再试一次。将 ignore_mx_lookup_error 设定成 yes，将让 Postfix 在第一次失败之后，立刻直接查询 A 记录。

范例：　ignore_mx_lookup_error = no

in_flow_delay

可能值：计时单位　　默认值：1s

Postfix 每收下一封邮件之后，至少停顿多长时间之后才可以再接收新信。如果你的服务器遭遇严重的效率问题，或是外来邮件的量实在太大，你可以适度增加此参数的值，让服务器系统有多一点时间消化刚刚收下的邮件。

范例：　in_flow_delay = 1s

initial_destination_concurrency

可能值：纯数值　　默认值：5

对于同一目的地，初始的投递进程数量。

范例：　initial_destination_concurrency =

ipc_idle

可能值：计时单位　　默认值：100s

内部通信渠道之间的闲置时间上限。一旦超过时限， Postfix 组件将主动切断渠道。

范例：　　ipc_idle = 100s

line_length_limit

可能值：纯数值　　默认值：2048

邮件中任何文字行的长度上限。超过限制的文字行，其超出的部分，在递送时期会被放入下一行。

范例：　　line_length_limit = 2048

lmtp_connect_timeout

可能值：计时单位　　默认值：0s

限制 LMTP client 完成一次 TCP 联机最长可以等待的时间。将此参数设定为 0，表示取消逾时计算。

范例：　　lmtp_connect_timeout = 0

lmtp_data_init_timeout

可能值：计时单位　　默认值：120s

限定 LMTP client在送出 LMTP DATA命令之后，LMTP server至少要在多少时间内响应。

范例：　　lmtp_data_init_timeout = 120s

lmtp_lhlo_timeout

可能值：计时单位　　默认值：300s

限定 LMTP client在送出 LMTP LHLO 命令之后，LMTP server 至少要在多少时间内响应 。

范例：　　lmtp_lhlo_timeout = 300s

lmtp_quit_timeout

可能值：计时单位　　默认值：300s

限定LMTP client 在送出 LMTP QUIT命令之后，LMTP server 至少要在多少时间内响应。

范例：　　`lmtp_quit_timeout = 300s`

lmtp_rset_timeout

可能值：计时单位　　默认值：300s

限定 LMTP client 在送出 LMTP RSET命令之后，LMTP server至少要在多少时间内响应。

范例：　　`lmtp_rset_timeout = 300s`

lmtp_tcp_port

可能值：通信端口编号　　默认值：24

当 *master.cf* 没有定义 `lmtp` 服务时，哪一个 TCP port 可供 LMTP联机之用。

范例：　　`lmtp_tcp_port = 24`

local_destination_concurrency_limit

可能值：纯数值　　默认值：2

设定同一个本地收件人最多可同时有几个投递进程。

范例：　　`local_destination_concurrency_limit = 2`

local_recipient_maps

可能值：查询表　　默认值：proxy:unix:passwd.byname $alias_maps

含有所有本地邮件地址的查询表。 SMTP server借此拒收外界寄给不明用户的邮件。

范例：　　`local_recipient_maps = unix:passwd.byname $alias_maps`

luser_relay

可能值：邮件地址　　默认值：无

外界寄给不明用户的邮件，全部转寄到 luser_relay 所指定的邮件地址（可能会收到大量垃圾邮件）。

范例：　luser_relay = info

mail_owner

可能值：系统账户名称　　默认值：postfix

Postfix 以此参数指定的系统账户，作为队列文件的拥有者以及Postfix daemon进程的运行身份。

范例：　mail_owner = postfix

mail_spool_directory

可能值：目录　　默认值：（随系统而异）

邮箱目录（存放邮箱文件的目录）的完整路径。

范例：　mail_spool_directory = /var/mail/spool

mailbox_command

可能值：路径名称　　默认值：无

用于将邮件投递到邮箱的外部命令，最常用的外部命令大概是 procmail。

范例：　mailbox_command = /usr/local/bin/procmail

mailbox_delivery_lock

可能值：逐项列出　　默认值：（随系统而异）

当 Postfix 将邮件投递到文件时，所使用的锁定方法。

范例：　mailbox_delivery_lock = fcntl, dotlock

mailbox_transport

可能值：传输服务　　默认值：无

用于执行邮箱投递操作的传输服务。

范例：　mailbox_transport = cyrus

manpage_directory

可能值：目录　　默认值：（随系统而异）

Postfix 在线说明书（ manpage）的存放目录，

范例：　manpage_directory = /usr/share/man

masquerade_domains

可能值：网域名称　　默认值：无

在邮件离开网关系统之前，将寄件人地址的内部主机名称部分替换成此参数所指定的网域名称。masquerade_domains 参数可指定一系列用于地址伪装的网域名称。

范例：　masquerade_domains = example.com

max_idle

可能值：计时单位　　默认值：100s

除了queue manager 之外的 Postfix服务器进程，等待新服务请求的闲置时间上限。

范例：　max_idle = 100s

maximal_backoff_time

可能值：计时单位　　默认值：4000s

在每次因故无法递送出邮件时， queue manager 便会将下次递送的时间延后，延长后的间隔时间不得超过 maximal_backoff_time。

范例：　maximal_backoff_time = 4000s

message_size_limit

可能值：字节　　默认值：10240000（10M Bytes）

限制单封邮件的最大长度。

范 例：　message_size_limit = 10240000

mime_header_checks

可能值：查询表（模式表）　　默认值：$header_checks

列出用于比对外来邮件的每个 MIME 标题字段的模式表。模式表的索引键，是描述 MIME标题字段特征的正则表达式（ regular expression,即“ 模式 ”),对应值是要对符合条件的邮件采取的处理动作。

范例：　mime_header_checks = regexp:/etc/postfix/mime_header_checks

minimal_backoff_time

可能值：计时单位　　默认值：1000s

每次因故无法递送出邮件时， queue manager 便会延长下次的投递时间，延长后的间隔时间不得低于 minimal_backoff_time。

范例：　minimal_backoff_time = 1000s

mydomain

可能值：网域名称　　默认值：（随系统而异）

邮件系统的网域名称。

范例：　mydomain = example.com

mynetworks

可能值：网络地址　　默认值：（随系统而异）

列出可通过本邮件系统寄出邮件的网络地址或 IP 地址。mynetworks 或mynetworks_style 两参数都可以用来界定转发服务的服务对象，但是 mynetworks 的效力高于 mynetworks_style。

范例：　mynetworks = 192.168.15.32/26

myorigin

可能值：网域名称　　默认值：$myhostname

要附加到只含人名部分（localpart）的不完整邮件地址的网域名称。

范例：　myorigin = $myhostname

newaliases_path

可能值：路径名称　　默认值（随系统而异）

用于重建别名数据库的 newaliases 程序的完整路径。（这是一个刻意为了保持对 Sendmail 的兼容性而设计的程序。）

范例：　newaliases_path = /usr/bin/newaliases

notify_classes

可能值：bounce、2bounce、delay、policy、protocol、resource、software的任何可能组合
默认值：resource,software

列出要触发发送通知函的错误类型。通知对象的邮件地址，定义在各错误类型的 *class*_notice_recipient 参数中。

范例：　notify_classes = resource, software

parent_domain_matches_subdomains

可能值：yes、no或任何使用查询表的参数名称　　默认值：（参阅范例）

此参数决定查询表里的 *domain.tld* 格式的网域名称，是否能涵盖其下的所有子网域（*subdomain.domain.tld*、*host.sub1.domain.tld*...）。如果设定为 no，则只有 *.domain.tld* 格式的网域名称才算涵盖其下的所有子网域。

范例：
```
parent_domain_matches_subdomains = debug_peer_list,
fast_flush_domains, mynetworks,
permit_mx_backup_networks,
qmqpd_authorized_clients, relay_domains,
smtpd_access_maps
```

pickup_service_name

可能值：master.cf 定义的服务名称之一　　默认值：pickup

供 master daemon 用于撷取本地回复邮件的服务。通常你不需要改变此参数。

范例：　pickup_service_name = pickup

process_id_directory

可能值：目录　　默认值：pid

master daemon 用于存放其锁定文件（进程本身的 PID 文件）的目录。指定的路径是在相对于 Postfix 邮箱目录（由 mail_spool_directory 参数决定）之下。

范例：　process_id_directory = pid

proxy_interfaces

可能值：IP地址　　默认值：无

如果运行 Postfix server 的机器是位于 Proxy 或 NAT 后台的内部网络上，而且是被当成网域的备用 MX 主机，那么当主 MX 系统离线时，它有可能遭遇邮件循环。 proxy_interfaces 列出哪些网络接口会从 Proxy或 NAT 网关收进邮件，让 Postfix 能够避免邮件循环。

范例：　proxy_interfaces = 192.168.15.23

qmgr_clog_warn_time

可能值：计时单位　　默认值：300s

当特定目的地的邮件阻塞于活动队列（ active queue）时，至少每隔多长时间要发出一次警告。设定值 0表示取消这项警告。

范例：　qmgr_clog_warn_time = 300s

qmgr_message_active_limit

可能值：纯数值　　默认值：20000

限定活动队列最多可容纳多少封邮件。

范例：　qmgr_message_active_limit = 20000

qmgr_message_recipient_minimum

可能值：纯数值　　默认值：10

每封邮件至少有多少个收件地址可以被存储在内存里。

范例：　qmgr_message_recipient_minimum = 10

qmqpd_error_delay

可能值：计时单位　　默认值：1s

QMQP 服务为一组邮件主机提供一个共同的集中式邮件队列。 qmqpd_error_delay 定义 QMQP server在送错误响应给客户端之前，可以停顿的时间。这项延迟是为了缓和行为错乱的客户端的混乱程度。

范例：　qmqpd_error_delay = 1s

queue_directory

可能值：目录　　默认值：/var/spool/postfix

Postfix 队列的主目录。

范例：　queue_directory = /var/spool/postfix

queue_run_delay

可能值：计时单位　　默认值：1000s

每隔多久扫描一次等待队列，重新投递等待邮件。

范例：　queue_run_delay = 1000s

rbl_reply_maps

可能值：查询表　　默认值：无

指出定义“ RBL网域名称 ”与“ RBL 回复信息模板 ” 对应关系的查询表。使用 reject_rbl 或 reject_rhsbl 来拒收垃圾邮件时，Postfix 依据本参数所指的查询表来解读 RBL 查询结果的意义。如果查询表里没列出 Postfix所使用的 RBL网域，则使用 default_rbl_reply 提供的样板来解读查询结果。

范例：　rbl_reply_maps = hash:/etc/postfix/rbl_reply

recipient_canonical_maps

可能值：查询表　　默认值：无

指出收件地址的正格表。操作流程与 canonical_maps 相似，只不过被改写的是收件人地址，而非寄件人地址。recipient_canonical_maps 的优先程度高于 canonical_maps。

范例：　recipient_canonical_maps = hash:/etc/postfix/canonical

reject_code

可能值：响应码　　默认值：554

基于客户端限制条件（smtpd_client_restriction)而拒收邮件时，返回给客户端的 SMTP 响应码。

范例：　reject_code = 554

relay_domains_reject_code

可能值：响应码　　默认值：554

因为客户端不符合使用转发服务的资格，而拒绝传信要求时， Postfix server 所使用的 SMTP 响应码。

范例：　relay_domains_reject_code = 554

relay_transport

可能值：传输服务　　默认值：relay

用于递送转发邮件的传输服务。

范例：　relay_transport = relay

relocated_maps

可能值：查询表　　默认值：无

定义已移出的邮件地址或网域的新位置。

范例：　relocated_maps = hash:/etc/postfix/relocated

resolve_dequoted_address

可能值：yes或 no　　默认值：yes

决定 Postfix 是否要处理人名部分含有“ 寄信方指定的邮递路径 ”的邮件地址。设定成 yes 时，Postfix会严格遵守 RFC 822 的规定，标记含有特殊符号（例如 @）的人名部分。

范例：　resolve_dequoted_address = yes

sample_directory

可能值：目录　　默认值：/etc/postfix

存放 Postfix 配置文件样本的目录。在这些样本文件里可以找到大部分 Postfix 参数的说明与范例。

范例：　sample_directory = /etc/postfix

sendmail_path

可能值：路径名称　　默认值：（随系统而异）

Postfix 的 sendmail 兼容程序的完整路径。sendmail 主要是供脚本或命令行用来寄送邮件。

范例：　sendmail_path = /usr/sbin/sendmail

setgid_group

可能值：组的名称或 GID 值　　默认值：postdrop

供 Postfix 用来提交邮件或管理队列的组标识符（Group ID）。你的系统应该设置一个专门给 Postfix 使用的组（通常命名为 postdrop）。

范例：　setgid_group = postdrop

showq_service_name

可能值：master.cf 定义的服务名称之一　　默认值：showq

用于回复 Postfix 队列状态的服务。通常不需要改变此参数。

范例：　showq_service_name = showq

smtp_bind_address

可能值：IP 地址　　默认值：无

当 SMTP client 要联机到远程的邮件服务器时，应该使用的网络接口的 IP 地址。只有在 multihomed 系统上（安装多张网卡，而且分别连接不同的网络），才有必要设定此参数；对只有单一实际网络接口的系统，此参数是多余的。

范例：
```
smtp_bind_address = 192.168.15.22
```

smtp_data_done_timeout

可能值：计时单位　　默认值：600s

在 SMTP client 送出“.”表示信息内容结束之后，最多可以等待 SMTP server 在多长时间内的响应。

范例：
```
smtp_data_done_timeout = 600s
```

smtp_data_xfer_timeout

可能值：计时单位　　默认值：180s

SMTP client 在传送信息的过程中，可以容忍的等待时间。如果联机阻塞时间超过设定值，SMTP client 会主动切断联机。

```
smtp_data_xfer_timeout = 180s
```

smtp_destination_recipient_limit

可能值：纯数值　　默认值（参阅范例）

限制通过 SMTP client 传出的每封邮件，最多可以有多少位收件人。

范例：
```
smtp_destination_recipient_limit =
$default_destination_recipient_limit
```

smtp_helo_timeout

可能值：计时单位　　默认值：300s

SMTP client 在送出 SMTP HELO 命令之后，等待服务器响应的限制时间。

范例：
```
smtp_helo_timeout = 300s
```

smtp_mail_timeout

可能值：计时单位　　默认值：300s

SMTP client 在送出 SMTP MAIL FROM命令之后，等待服务器响应的限制时间。

范例：　　smtp_mail_timeout = 300s

smtp_pix_workaround_delay_time

可能值：计时单位　　默认值：10s

若 SMTP server 是位于某些有瑕疵的老式 Cisco PIX 防火墙之后，当 SMTP client 送出的“ .”与“ CR/LF ”信息结束信号，是分别放在两个不同的封装包时，这类防火墙可能会干扰 SMTP 通信。Postfix 能自动侦测出此问题，并借由延迟送出“ .”与“ CR/LF ”的时间，来避开这问题的发生。 smtp_pix workaround_delay_time参数指出 SMTP client在传送完邮件内容之后，必须要等待多长时间（让 socket的输出缓冲区可以清空），然后才可送出“ .CRLF” （如此可确保这三个字符会被编在同一个封装包）。

范例：　　smtp_pix_workaround_delay_time = 10s

smtp_quit_timeout

可能值：计时单位　　默认值：300s

SMTP client 在送出 SMTP QUIT 命令之后，等待服务器响应的限制时间。

范例：　　smtp_quit_timeout = 300s

smtp rcpt timeout

可能值：计时单位　　默认值：300s

SMTP client 在送出 SMTP RCPT TO 命令之后，等待服务器响应的限制时间

范例：　　smtp_rcpt_timeout = 300s

smtp_skip_5xx_greeting

可能值：yes或 no　　默认值：yes

当远程的 SMTP server 返回 5xx 响应码，Postfix 可以直接退信给原寄件人或是试着将邮件寄到目标网域的其他MX主机（如果有的话）。smtp_skip_5xx_greeting决定 Postfix 应该采取何种行为。设定为 no 时，Postfix 会继续尝试寄信到另外的 MX 主机。

范例：　　smtp_skip_5xx_greeting = yes

smtpd_banner

可能值：格式样板　　默认值：（参阅范例）

跟在 220 状态码之后的 SMTP 问候信息。如果你改变此参数，请遵守 RFC 的要求，以 $myhostname 为信息的开头。

范例：　smtpd_banner = $myhostname ESMTP $mail_name

smtpd_data_restrictions

可能值：UBE 限制条件组合　　默认值：无

客户端送出 SMTP DATA 时，所要检查的 UBE 限制条件组合。

范例：　smtpd_data_restrictions = reject_unauth_pipelining

smtpd_error_sleep_time

可能值：计时单位　　默认值：1s

当客户端引发错误时， Postfix 的初始等待时间。当错误次数超过 smtpd_soft_error_limit 的限制时，Postfix 在每次错误之后，都会延长一秒等待时间。

范例：　smtpd_error_sleep_time = 1s

smtpd_expansion_filter

可能值：字符　　默认值：（参阅范例）

SMTP server 展开宏时，所容许的字符。

范例：　smtpd_expansion_filter = \t\40!"#$%&'( )*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_` abcdefghijklmnopqrstuvwxyz{|}~

smtpd_helo_required

可能值：yes或 no　　默认值：no

决定 Postfix 是否要求客户端必须先送出 HELO/EHLO 命令，才能开始正式开始 SMTP对话。

范例：　smtpd_helo_required = no

smtpd_history_flush_threshold

可能值：纯数值　　默认值：100

限制 SMTP server 命令历程暂存区可容纳的行数。超过此限之后，旧命令历程将被清理掉 。

范例：
```
smtpd_history_flush_threshold = 100
```

smtpd_noop_commands

可能值：逐项列出　　默认值：无

列出 Postfix 应该接受、但是不动作的 SMTP命令。对于这些无动作命令， Postfix 总是响应“ 250 Ok”。

范例：
```
smtpd_noop_commands = vrfy, expn
```

smtpd_recipient_limit

可能值：纯数值　　默认值：1000

对于每封邮件，客户端最多可以在 RCPT TO命令指定多少个收件地址。如果超过此限，Postfix 将拒绝 RCPT TO 命令。

范例：
```
smtpd_recipient_limit = 1000
```

smtpd_restriction_classes

可能值：自定义规范等级的名称　　默认值：无

列出管理员自己定义的所有规范等级。每一组规范等级，都是由一系列 UBE限制条件组成。

范例：
```
smtpd_restriction_classes = myruleset_a, myruleset_b
myruleset_a = reject_invalid_hostname,
reject_unknown_sender_domain
myruleset_b = permit
```

smtpd_soft_error_limit

可能值：纯数值　　默认值：10

客户端在发生多少次错误之后， Postfix 才开始执行再出现错误就延迟一秒的操作。

范例：
```
smtpd_soft_error_limit = 10
```

soft_bounce

可能值：yes或 no　　默认值：no

设定成 yes 时，原本应该退信的动作，会改成将邮件放回队列，等待下次递送；此外，原本应该响应永久错误码（5xx）的状况，一律改成以暂时性错误码（4xx）响应。当你想测试新配置的效果，却又担心发生误判时（比方说，拒收不应该拒绝的重要邮件），可利用此参数避免造成无法弥补的错误。

范例：　soft_bounce = no

strict_7bit_headers

可能值：yes或 no　　默认值：no

决定 Postfix 是否应该严格遵守 RFC的规定，只接受标头全部都是 7-bit ASCII字符的邮件。在默认情况下， Postfix 接受标头里含有 8-bit 文字的邮件（译注 1）。

范例：　strict_7bit_headers = no

strict_8bitmime_body

可能值：yes或 no　　默认值：no

决 定 Postfix应该接受还是拒绝含有 8-bits文字、但是却没编码成适当的 MIME格式的邮件 。

范例：　strict_8bitmime_body = no

strict_rfc821_envelopes

可能值：yes或 no　　默认值：no

依照 RFC 821 的规定，信封地址必须放在一对尖角括号之内，而且不能含有多余信息。此参数决定 Postfix 是否要求客户端严格遵守这项规定。

译注 1: 依照 RFC 的规定，MUA 要负责将标头里的 8-bits（或 16-bits）编码字符转换成只含 7-bits 字符的 MIME特殊格式。例如，当你寄出一封主题为“ 中文主题 ”的邮件时， MUA 安插在邮件标头里的 Subject ：字段应该是‘ Subject: =?Big5?B?pKSk5aVEw0Q=?=”，而非“ Subject ：中文主题 ”。支持多国文字的 MUA 应该都能将其正确转换成 MIME 格式，但是很多专门用来滥发垃圾邮件的软件并未严格遵宁 RFC 的规定。因此，将 strict_7bit_headers 设定成 yes，可以挡掉不少垃圾邮件，不过，我不知道误判合法邮件的风险有多高。

范例：　　strict_rfc821_envelopes = no

swap_bangpath

可能值：yes或 no　　默认值：yes

UUCP使用“！”字符来指定邮件递送路径。swap_bangpath参数决定 Postfix是否应该将“！”换成“@”符号，使其成为适合在 Internet上传递的邮件。

范例：　　swap_bangpath = yes

syslog_name

可能值：字符串　　默认值：postfix

Postfix 在邮件日志文件里自称的进程名称。

范例：　　syslog_name = postfix

transport_retry_time

可能值：计时单位　　默认值：60s

每隔多久时间，Postfix 重新尝试传递先前未能顺利送出的邮件。

范例：　　transport_retry_time = 60s

undisclosed_recipients_header

可能值：字符串　　默认值：（参阅范例）

当标头里的 To:、Resent-To:、Cc: 等字段后全部都没有收件人地址时，应该放在这些字段之后的字符串。

范例：　　undisclosed_recipients_header = To: undisclosed-recipients:;

unknown_client_reject_code

可能值：响应码　　默认值：450

由于 reject_unknown_client限制条件而拒绝客户端时，所使用的 SMTP响应码。

范例：　　unknown_client_reject_code = 450

unknown_local_recipient_reject_code

可能值：响应码　　默认值：550

当客户端试图寄信给不存在的本地网域用户时， Postfix 用于拒绝客户端的 SMTP 响应码。

范例：　unknown_local_recipient_reject_code = 550

unknown_virtual_alias_reject_code

可能值：响应码　　默认值：550

当客户端试图寄信到Postfix控制管理的虚拟别名网域，但是所指定的收件人不存在于该网域时， Postfix 使用此参数指定的 SMTP 响应码来拒绝客户端。

范例：　unknown_virtual_alias_reject_code = 550

verp_delimiter_filter

可能值：字符　　默认值：-=+

VERP是供 MLM处理退信的技术。此技术以一个特殊的分隔符，将“ 列表拥有者地址 ”与“ 原收件人地址 ”结合在一起。 verp_delimiter_filter参数列出可当成分隔符的合法字符。

范例：　verp_delimiter_filter= -=+

virtual_alias_maps

可能值：查询表　　默认值：无

此参数指向含有“ 虚拟别名地址 ”与“ 实际收件地址 ”对应关系的查询表。

范例：　virtual_alias_maps = hash:/etc/postfix/virtual_alias

virtual_mailbox_base

可能值：目录　　默认值：无

虚拟邮箱文件的相对路径起点。所有虚拟邮箱文件都是放在此目录之下。

范例：　virtual_mailbox_base = /usr/local/virtual_mail

virtual_mailbox_limit

可能值：字节　　默认值：51200000

虚拟邮箱文件的容量上限。对于maildir 格式的邮箱而言，此参数只能限制每个文件（只含一封邮件）的大小，而非整体邮箱的大小。此参数值不得低于 message_size_limit

范例：　　virtual_mailbox_limit = 51200000

virtual_mailbox_maps

可能值：查询表　　默认值：无

此参数指向含有“虚拟邮箱地址”与“邮箱文件”对应关系的查询表。邮箱文件的路径，以 virtual_mailbox_base所指的路径为起点。

范例：　　virtual_mailbox_maps = hash:/etc/postfix/virtual_mailbox

virtual_transport

可能值：传输服务　　默认值：virtual

用于投递邮件到虚拟邮箱地址的默认传输服务。

范例：　　virtual_transport = virtual

附录二

Postfix 支持工具

Postfix 包提供了一系列命令行工具，本附录大略说明各工具的用途，让读者对各个工具的作用有个大概印象。关于详细的用法与选项，在线说明文件中已经有很详细的解释了，当你需要查阅某工具的用法时，查询 man 会比翻书更快，所以这里不再赘述。

postalias
: 创建或查询别名数据库。

postcat
: 显示出队列文件的内容，让管理员可观察滞留在队列里的邮件内容。

postconf
: 显示或改变 Postfix 参数。可一次显示一个参数，或是显示出所有参数。

postdrop
: 将邮件放回到 maildrop／目录，由 Postfix 重新进行投递操作。

postfix
: 启动或停止 Postfix 系统，或重新读取配置文件。也可以用于其他维护工作，包括检查系统配置，以及清空队列。

postkick
: 对特定 Postfix 服务发出请求。此工具的作用，主要是给 shell scripts 提供一个能够与 Postfix 沟通服务的管道。

postlock
: 锁定特定文件，确保能够独占访问。此工具的作用，主要是让 shell scripts 能使用兼容于 Postfix 的锁定方式。

postlog

将指定的信息记录到系统日志文件中。这是支持 shell scripts 的工具，使其能以类似于 Postfix 的样式来记录信息到日志文件。

postmap

创建查询表的 DB 数据库或查询查询表内容。Postfix 有许多配置信息都是记录在 postmap 所建的查询表数据库中。

postqueue

让一般的用户能够有限度地访问 Postfix 队列。可能改变队列的访问方式需要有管理员特权才能进行，而这方面的访问能力由 postsuper 命令提供。

postsuper

供管理员访问 Postfix 队列。管理员可删除邮件、扣留邮件（搬到 hold 队列）、取回邮件（将邮件从 hold 队列搬回 active 队列），必要时，还可以修复队列目录结构。

附录三

Postfix 的编译与安装

从源程序建构出一套 Postfix，大致的步骤是：取得软件包、解压缩、编译、安装。这些步骤所需的工具包括：`gzip`、`tar`、`make` 与一套 C 编译器，在各个 Unix 系统上都应该能够找到。Postfix预设你的系统上会有 GNU `gcc` 编译器，不过，你也可以使用自己平台上的原始编译器，但其必须支持 ANSI C。

取得 Postfix

在 Postfix的正式网站（*http://www.postfix.org/*）可找到一个 download 链接，点击进去之后可见到世界各地的映像站，你应该选择地理位置离你最近的映像站。你可以选择 Official 或 Experimental 版本（参阅第一章）的 Source Code。本附录假设你下载的文件为 *postfix-2.0.18.tar.gz*。如果你取得的版本不同于本附录的引例，请依实际情况适当调整。

Postfix 编译过程入门

在示范Postfix的实际建构步骤之前，我们先为没有程序设计经验的读者介绍，关于C语言程序建构过程的基本概念。

Postfix是以 C语言程序写成的软件，其源程序由许多 *.c* 与 *.h* 文件构成。建构过程大致分成编译（compiling）、链接（linking）两大阶段。在编译阶段，编译器（通常是 GNU gcc）将 *.c* 与 *.h* 文件转换成“目标文件”（*.o*）；在链接阶段，链接器（通常是 `ld`）将前阶段产生的 *.o* 文件与必要的函数库整合在一起，产生出“可执行文件”（executable）。

然而，哪些 *.c*、*.h* 文件可产生哪些 *.o* 文件？哪个可执行文件又需要哪些 *.o* 文件？用到哪

些函数库？运行 gcc 与 ld 时，又分别需要哪些选项？这些错综复杂的关系，只有程序设计者才搞得清楚。为了简化编译、链接的操作，程序设计者将各文件之间的依赖性、编译与链接过程的选项，都写在一个 *Makefile* 文件中，make 工具程序依据 *Makefile* 提供的信息，自动执行必要的建构动作，产生我们最后想要的可执行文件。

你不必担心如何编辑 *Makefile* 文件，因为 Postfix 包已经提供了该文件，而且提供一套操作程序，让你用几个简单命令既可产生适合你的环境的 *Makefile*。因此，你不应该擅自编辑 *Makefile*，因为你所做的任何改变，可能会在后续的过程中被改写。

Postfix 的 *Makefile* 需要一些环境变量，例如 CCARGS。包随附的 *INSTALL* 文件说明了所有需要的环境变量，我们只解释其中比较常用的几项变量。

下列环境变量用于设定编译期的选项。如果变量值含有空格符或其他的 shell 特殊字符，整个变量值必须放在一对双引号之间。

AUXLIBS

指出位于标准位置之外的额外函数库。假设你需要链接 SASL、MySQL、LDAP、或任何额外附加的函数库，而且它们不是放在标准位置（*/usr/lib/* 目录），则必须在 AUXLIBS 变量指出这些函数库的路径。

CC

指出所要使用的特定编译器。如果你不想使用 Postfix 预先选择的编译器，请在此变量中指出你想使用的编译器。原则上， Postfix 优先选择 GNU gcc，除非该系统平台上有已知的更佳选择。你可以检查 *makedefs* 文件，看看 Postfix 在你的系统平台上预选的编译器为何。

CCARGS

提供额外选项给编译器。如果你的编译器支持特殊选项，或是所需的支持文件不是放在默认目录下，请在此变量中指出这些特殊选项。

DEBUG

构造 Postfix 可执行文件时，编译器所使用的调试等级。等级越高，调试工具（Debugger）就能够得到更详细的调试信息。当你建构用于实际系统的 Postfix 时，建议你彻底关闭调试支持。

OPT

构造 Postfix 可执行文件时，编译器可进行何种程度的最优化。超过建议程度的优化措施或许能提升些许效率，但代价是需要更多内存（以及更长的编译时间）。建议你接受 Postfix 为你的系统平台所选择的默认优化程度。

编译器选项

额外的编译选项是设定在CCARGS环境变量中的，C源程序(*.c*) 需要的特定函数与变量另外定义在标头文件（*.h*)。标头文件的标准位置位于 */usr/include/* 目录。如果你的标头文件位于其他地方，你得让编译器知道要去哪里找到它们。编译器选项 -I 用于指出额外标头文件的存放目录。如果你的 Postfix需要其他软件包的函数库，而它们的标头文件不是放在标准位置，你的CCARGS 变量应该包含 -I选项，指出额外标头文件的存放目录。一般而言，额外安装的软件开发包，它们的标头文件多半被安装在 */usr/local/include/* ，如果你的 Postfix要链接它们，则 CCARGS变量应该这样设定：

```
CCARGS='-I/usr/local/include/'
```

如果额外标头文件的存放位置不只一处，那么对于每一个额外目录，应使用一个额外的 -I选项指出其完整路径。编译器依照你指定的顺序，逐一寻找所需的标头文件。

Postfix包定义了一系列宏，这些宏各自代表在系统上找到的特定资源以及你预先设定的选项。Postfix 依据这些宏来进行条件式编译；换言之， Postfix 的编译过程可随系统实际拥有的资源与函数库而调整。编译器选项 -D供用户定义编译期宏，对于每一个你想要编译进Postfix 的外挂包，你都应该定义一个代表该包的宏。举例来说，若你希望 Postfix 具备查询 MySQL的能力，你应该定义 HAS_MYSQL宏：

```
CCARGS='-DHAS_MYSQL'
```

链接器选项

链接器选项是设定在 AUXLIBS变量中的。在 Postfix被编译成目标文件（*.o*) 之后，还必须与相关的函数库链接在一起，才能产生可执行文件。系统函数库的标准位置是位于 */usr/lib/* . -L选项告诉链接器从哪些目录中可以找出额外函数库：

```
AUXLIBS='-L/usr/local/lib'
```

除了指出函数库存放目录之外，还必须用 -l 选项让链接器知道函数库的名称。函数库文件必须被放在标准位置或 -L指出的目录。函数库文件的命名遵循一定的法则：文件名前三个字母固定为 *lib*，其后跟着函数库本身的名称，最后是扩展文件名。从扩展文件名可看出该函数库文件适用的链接方式，*.a*（代表 archive）用于静态函数库 ,*.so*或*.sl*（分别代表 shared objects与 shared libraries ）用于动态链接。要特别注意的是， -l 指出的是“函数库的名称”（在*lib*与扩展文件名之间的部分),而非‘函数库文件名”。举例来说，若要链接 */usr/local/lib/* 目录下的 MySQL client library （其函数库文件名为 *libmysqlclient.a*)，AUXLIBS 变量里应该包含的 -L与 -l选项如下：

```
AUXLIBS='-L/usr/local/lib -lmysqlclient'
```

如果函数库同时提供动态与静态两种版本，大多数的链接器会挑选动态版本。所谓动态链接，是说程序在运行时被加载到内存的过程中，才从函数库中取得所需的目标码，构成一个完整的程序。如果是在编译期就将函数库里的目标码集成进可执行文件，则称为静态链接。静态链接的好处是它可以独立运行，但缺点是比较浪费内存，产生出来的可执行文件也比较庞大。动态链接的优缺点刚好与静态链接相反：可执行文件体积较小、较节省内存，但是不能独立运行 —— 动态函数库必须存在于运行时环境中。如果你总是将动态函数库安装在 */usr/lib/* 标准目录下，你的系统就应该能在运行时找到函数库。不过，并非所有软件包的函数库都安装在运行时可找到的位置。不同的系统，使用不同的方法来找出动态函数库的位置。有些系统以特定的环境变量来表示动态函数库的存放路径，有些系统则是将动态函数库的路径写入可执行文件本身。请检查相关环境变量包含了动态函数库的安装路径，或是确定这些函数库都安装在系统的标准目录下。你也可以在建构过程中，明确指出特定函数库的实际路径。

决定运行时动态函数库搜索路径的链接器选项，随实际使用的链接器与系统平台而定。以 GNU linker（Linux、FreeBSD、SGI IRIX ）为例，此选项称为 `-rpath`，Solaris则是使用 `-R`，而 HP-UX为 `+b`。如果你不知道自己系统的链接器使用的是哪一种，请参阅 `ld(1)` 在线说明。

以 SSL函数库为例，假设你的 *libssl.so* 文件是位于 */usr/local/lib/* 目录下，而你是在 FreeBSD（或任何使用 `-rpath` 的系统）上建构 Postfix，那么，`AUXLIBS` 环境变量应该像下面这样设定：

```
AUXLIBS='-L/usr/local/lib -rpath/usr/local/lib -lssl'
```

gcc 与无法识别的链接器选项

某些版本的 `gcc` 不见得能识别所有链接器选项（`-rpath` 就是其中一例），因而在编译过程中发生错误。在这种情况下，你会看到这样的错误信息：`gcc: unrecognized option '-rpath'`。由于 `-rpath` 是链接器的选项，所以我们并不需要 `gcc` 真的能识别它。这个问题有个简单解法：使用 `gcc` 的 `-Wl` 自变量指出应该交给链接器的选项，`gcc` 本身不解读 `-Wl` 所指定的选项。当我们希望 `gcc` 将 `-rpath` 选项直接传给链接器时，我们可在 `-rpath` 字样之前加上 `-Wl,`：

```
AUXLIBS='-L/usr/local/lib -Wl,-rpath,/usr/local/lib -lssl'
```

关于细节，请参阅 `gcc(1)` 在线说明。

若你的Postfix 需要与外部函数库链接，而你的系统上安装了同一函数库的多个版本，你得确定Postfix 的链接对象是正确的版本。此外，由于函数库与标头文件有相当紧密的连

带关系，请确定实际链接的函数库版本与编译期引入的标头文件版本相符。版本不匹配是最常见的编译错误原因。有时候，编译器不见得发出警告，而你也能顺利建构出Postfix，但是你会在运行时遇到一些非常难以追查的奇怪问题，这时候，你应该怀疑标头文件与函数库的版本是否匹配。

建构 Postfix

从 Postfix 映像站下载回来的源文件，封装成俗称为“ tarball ”的格式：

```
[me@andy /usr/src/tarballs]$ wget ftp://postfix.linuxaid.com.cn/pub/postfix/
official/postfix-2.0.18.tar.gz
```

你必须先以 `gzip` 解压缩，然后使用 tar 解开封装，才可以开始进行编译：

```
[me@andy /usr/src/tarballs]$ cd ..
[me@andy /usr/src]$ gzip -d tarballs/postfix-2.0.18.tar.gz
[me@andy /usr/src]$ tar -xf tarballs/postfix-2.0.18.tar
```

如果你的系统上有 GNU tar，则解压缩与解封装的动作可以一气呵成：

```
[me@andy /usr/src]$ tar xzvf tarballs/postfix-2.0.18.tar.gz
```

解开封装之后，你的工作目录下会产生一个 *postfix-2.0.18/*子目录，请将工作目录切换到此子目录：

```
[me@andy /usr/src]$ cd postfix-2.0.18
[me@andy /usr/src/postfix-2.0.18]$ ls
...
```

如果你接受 Postfix 的默认编译条件，那么，只要在 *postfix-2.0.18/* 目录下运行 `make`，就可以完成整个建构过程：

```
[me@andy /usr/src/postfix-2.0.18]$ make
```

`make`会先检查你的系统环境，产生一个适当的 *Makefile* ，然后依据这个 *Makefile* 建构出一个适合你的系统的 Postfix。如果你真的不需要改变默认编译条件，现在就可以跳到“ 安装 ”,开始进行安装。

调整编译条件

Postfix 在各系统平台上的编译条件定义在 *makedefs* 文件里。如果你好奇，可以打开此文件，看看 Postfix 在你的系统平台上的默认编译条件为何。 *makedefs* 的作用是产生适合当前系统环境的 *Makefile*。当你下达 `make` 命令时，如果当时的工作目录下没有 *Makefile*,它便依照 *makedefs*的指示来识辨你的系统环境，并将估算出来的宏与定义写

入 *Makefile*。接着，这个 *Makefile* 会被 make 用来完成实际的建构过程。上述所有动作都是自动完成的，所以你其实用不着理会 *makedefs* 文件里到底写了些什么。

但是，你如果想要改变默认编译条件，则必须分两步骤来完成建构过程。第一步是使用 make makefiles 命令，并在命令行中指定你想改变的编译或链接条件（参阅“编译器选项”与“链接器选项”），它会依照你的意思产生一个新的 *Makefile* 。举例来说，如果在 HP-UX 系统上，你不想使用 Postfix 预选的编译器，你可以像下面这样指出正确的编译器：

```
$ make makefiles CC="/opt/ansic/bin/cc -Ae"
```

当然，除了指出编译器的正确路径之外，你还可以加上任何必要的选项。比方说，如果有些标头文件是放在 */usr/local/include/* 目录中，你的 CCARGS 变量应该包含该目录：

```
$ make makefiles CCARGS="-I /usr/local/include/"
```

不同的变量，可以仿照下列方式结合在一起：

```
$ make makefiles CC="/opt/ansic/bin/cc -Ae" \
>    CCARGS="-I /usr/local/include"
```

修改 Postfix 默认值

Postfix 的配置文件（*master.cf* 与 *main.cf*）提供了高度的灵活性，几乎所有的运行时参数 —— 包括它使用的各种目录，都可以在配置文件里设定。唯一的例外是配置文件本身的位置。如果我们希望 Postfix 在 */usr/local/etc/postfix/* 目录下去寻找它的配置文件（而非默认的 */etc/postfix/* 目录），你可以在 CCARGS 变量里重新定义 DEF_CONFIG_DIR 的值，借此改变配置文件的默认位置：

```
$ make makefiles CCARGS='-DDEF_CONFIG_DIR=\"/usr/local/etc/postfix\"'
```

这个命令的语法有些复杂，需要稍加解释。首先，“ = ”后面的值必须放在一对引号之间。但是这里有两个“值”，一个是 DEF_CONFIG_DIR 宏本身的值（即 */usr/local/etc/postfix*），另一个是 CCARGS 环境变量的值（即 -DDEF_CONFIG_DIR="/usr/local/etc/postfix"）。为了避免 shell 误解，在单引号之间的双引号，前面必须加上 \ 符号（变成 '\"'），这样，shell 才不会将双引号误解为特殊字符。

想要改变多项编译条件时，必须在同一个 make makefiles 中做改变，而不能分别以多次 make makefiles 命令来改变单项编译条件。因为，每次运行 make makefiles 命令时，都会覆盖前一次产生的 *Makefile*。当你的命令行开始变得复杂时，建议你写一个简单的 shell script。本附录最后的“总结”有这么一个 shell script 供你参考。

一旦使用 `make makefiles` 与适当选项产生出合适的 *Makefile* 之后，就可以用 `make` 来建构你的 Postfix：

```
$ make
```

安装

成功完成建构过程之后，就可以开始将 Postfix 安装到你的系统上。你需要 *root* 权限才能顺利运行安装程序，因为 Postfix 的程序文件与配置文件必须被放在系统上的适当目录下，而这些目录只有 *root* 才有写权限。

首先，你必须设置一个新的虚账户，以便充当 Postfix 队列与大部分进程的拥有者。这个虚账户不应该具备登录系统的能力，所以不需要 login shell，也不需要主目录。所有 Unix 系统都提供创建新账户的管理工具，请使用你惯用的工具来创建新账户。你可以将密码设定为 *（没密码），并将其主目录与 shell 指向无效路径（例如 */bin/false* 或 */dev/null*）。习惯上，供 Postfix 系统使用的虚账户的名称是 *postfix*，但这不是硬性规定，你可以使用任何你喜欢的名称，唯一要求是 UID 不能与其他账户重复。建好新账户之后，其 */etc/passwd* 里的记录应该类似下面这样：

```
postfix:*:1001:1001:postfix:/no/where:/bin/false
```

除了专用账户之外，还必须另外设置一个专用组，该组不应该包含任何成员，包括先前新建的 *postfix* 虚账户也不例外。习惯上，该组的名称应该是 *postdrop*，在大部分系统上，你可以编辑 */etc/group* 文件，直接加入下列内容：

```
postdrop:*:1007:
```

由于 Postfix 被刻意设计成为 Sendmail 的替代品，而为了保持兼容性， Postfix 会以它自己的 *sendmail* 可执行文件取代系统上原有的同名文件。你可能会想要修改原版文件的名称，以免被 Postfix 的版本覆盖。一般而言， Sendmail 包的 *sendmail* 可执行文件若不是放在 */usr/sbin/sendmail* 就放在 */usr/lib/sendmail*（随系统平台类型而定）。如果你不确定，请使用 whereis 或 which 查出 *sendmail* 可执行文件的实际路径：

```
# whereis sendmail
```

它可能会列出多个文件，而那个没有任何扩展文件名的文件，才是你要找的对象（译注 1）。找到之后，将它修改成一个不会被 Postfix 覆盖的名称：

译注 1：RedHat 7.3 以后的版本，同时提供了 Sendmail 与 Postfix 这两个包，并以巧妙的链接与目录安排，使得两个包可以共存于同一个系统上。在这类系统上，你可以使用 redhat-switch-mail-nox（这是一个 TUI 程序）来选择要使用哪一个组件。在你按照本附录的步骤进行安装之前，建议你先切换成 Postfix 模式。

```
# mv /usr/sbin/sendmail /usr/sbin/sendmail.orig
```

除了 *sendmail* 可执行文件之外，还有两个会被 Postfix 替换掉、而你想要改名的文件：*mailq* 与 *newaliases*。你应该可以在 */usr/bin/* 目录下找到这两个文件，如果必要的话，你也可以使用 `whereis` 找出实际的路径。在某些系统上，*mailq* 与 *newaliases* 可能只是 *sendmail* 的符号链接而已。

```
# mv /usr/bin/mailq /usr/bin/mailq.orig
# mv /usr/bin/newaliases /usr/bin/newaliases.orig
```

现在，你可以开始进行真正的安装，将建构出来的程序文件与相关支持文件存放到系统上的适当目录。如果这是你第一次安装 Postfix，请运行：

```
# make install
```

这个命令会启动 postfix-install 脚本，该脚本会先检查要被安装的文件是否完备，然后询问你几个关于重要系统目录的问题：

```
install_root: [/]
```

install_root 是你的系统的根目录，如果改变它，Postfix 的所有文件都会被安装在你所指定的新目录下的相关子目录。比方说，若你将 *install root* 改成 */tmp/testbuild*，则 *main.cf* 会被安装在 */tmp/testbuild/etc/postfix/* 目录下。一般而言，你不应该修改 *install_root*，除非你想要制作一个可直接安装到其他同类系统上的包。你会被问到的第二个问题是：

```
tempdir: [/tmp]
```

tempdir 是 postfix-install 脚本用来存放临时文件的目录，在完成安装之后，这些临时文件会被清除干净，因此，你通常不必修改 *tempdir*。接下来的问题是一系列关于重要文件的存放路径：

```
config_directory: [/etc/postfix]
daemon_directory: [/usr/libexec/postfix]
command_directory: [/usr/sbin]
queue_directory: [/var/spool/postfix]
sendmail_path: [/usr/lib/sendmail]
newaliases_path: [/usr/bin/newaliases]
mailq_path: [/usr/bin/mailq]
```

通常，你应该接受 Postfix 建议的默认值。你只要确定 Postfix 建议的位置，刚好能覆盖系统上原有的 *sendmail*、*newaliases* 与 *mailq* 等可执行文件即可。如果其建议路径与实际情况有所出入，请在 postfix-install 提示相关目录时予以修正。下一个问题：

```
mail_owner: [postfix]
```

mail_owner的默认值为 *postfix*，而且它假设你已经按照先前叙述的准备步骤，事先在系统上创建好此虚账户。如果当初你建的账户使用另一个名称，请在此修正。接下来的问题是：

```
setgid_group: [postdrop]
```

setgid_group的默认值为 *postdrop* 。照例，postfix-install 假设你的系统上已经存在此群组。如果你使用其他组名，请在此修正。下一个问题：

```
manpage_directory: [/usr/local/man]
```

这个问题的答案决定 Postfix在线说明书的安装位置，你应该可以接受建议值；否则，你应该提供 manpage命令所列的目录之一。

```
sample_directory: [/etc/postfix]
```

配置文件样本提供了各项Postfix参数的解释与范例，你应该将这些样本文件存放在你方便查阅的目录下。Postfix的默认值（*/etc/postfix/*）并不是一个很好的选择，除非你平常用来登录系统的账户有权访问该目录。

```
readme_directory: [no]
```

Postfix包里随附了好几个*README*文件，在这些文件里可以找到特定附加软件（SASL、MySQL等）的额外信息。虽然它们的重要性不如配置文件样本，但是，我们仍建议你找在平常容易查阅的地方来安装它们。当然，如果不安装它们，你仍可以在源程序的目录下找到这些文件。

答复完上述问题之后，postfix-install 会依据你的答案，将文件安装到相关目录。

升级

如果你时常注意Postfix 的发展动态，可能会发现它的改版速度相当快。这表示每隔几个月，你就会想要升级自己系统上的Postfix 版本。新版 Postfix 的编译过程没什么特殊之处，按照“建构 Postfix”描述的步骤就可以完成。编译好新版 Postfix之后，你应该先停用目前正在运行中的 Postfix，然后再进行升级。升级时它不会问你任何问题，因为升级脚本可从现有的 *main.cf*中查出所有实际安装目录。

升级步骤如下：

```
# postfix stop
# make upgrade
# postfix start
```

当你运行 make upgrade 时，它先检查文件的交互情况，然后以新编译出来的版本取代旧版文件。在重新启动 Postfix 之后，你应该检查日志文件，看看新版的 Postfix 是否能够顺利运作。

编译附加包

本书的第十二章、十三章与十五章，分别介绍 Postfix 如何发挥 Cyrus SASL、TLS、MySQL 与 LDAP 的功能。这些功能不是 Postfix 最初就有的，而需要靠相关附加包提供，换言之，Postfix 在建构过程中必须与相关附加包联接，才有你需要的功能。本节分别示范如何在 Postfix 的建构过程中加入各项附加包的支持能力。

在开始实际进行编译之前，应该先用 make tidy 清理编译过程余留下来的文件。请在 Postfix 的源程序目录下执行这个命令：

```
[me@andy /usr/src/postfix-2.0.18]$ make tidy
```

现在，你的源程序目录应该和初始时一样。接下来的四个小节，都只说明如何产生适当的 *Makefile*，在那之后，只要用 make 命令就可重建你的 Postfix 包：

```
[me@andy /usr/src/postfix-2.0.18]$ make
```

重建成功之后，直接升级目前安装好的 Postfix 即可：

```
# make upgrade
```

当然，如果你的系统从没安装过 Postfix，应该使用 make install 代替。

Cyrus SASL

关于 Postfix 与 Cyrus SASL 的详细信息，请参阅第十二章。从 Carnegie Mellon 大学的网站（*http://asg.web.cmu.edu/sasl/*）可以免费下载 Cyrus SASL 函数库的源程序。Cyrus SASL 有两个重要版本，本书假设你使用比较新的 2.x 版。

编译 Cyrus SASL 的过程中，有一个影响 Postfix 的因素，那就是要不要支持某些旧版的 Microsoft client，因为这类软件使用非标准的验证机制。标准的 plain-text 验证机制规定客户端必须先发出 PLAIN 命令，但是某些旧版的 Microsoft MUA 软件却错误地使用 LOGIN 命令来代替。如果你的用户中还有人继续使用这类蹩脚软件，当你使用 ./configure 设定 Cyrus SASL 的编译条件时，记得要加上一 enable-login 选项。以下是建构 Cyrus SASL 函数库的步骤（仅供参考）：

```
[me@andy /usr/src/tarballs]$ wget \
```

```
> http://ftp.andrew.cmu.edu/pub/cyrus-mail/cyrus-sasl-2.1.17.tar.gz
[me@andy /usr/src/tarballs]$ cd ..
[me@andy /usr/src]$ tar xzvf tarballs/cyrus-sasl-2.1.17.tar.gz
[me@andy /usr/src]$ cd cyrus-sasl-2.1.17/
[me@andy /usr/src/cyrus-sasl-2.1.17]$ ./configure --enable-login \
> --enable-ntlm --enable-sql --mandir=/usr/share/man
[me@andy /usr/src/cyrus-sasl-2.1.17]$ make
[me@andy /usr/src/cyrus-sasl-2.1.17]$ sudo make install
```

安装好 Cyrus SASL 函数库之后，请记下它们的安装位置。标头文件的默认安装目录是 */usr/local/include/*，函数库本身则是放在 */usr/local/lib/* 目录下。以下的示范假设你使用默认安装目录。如果你选择其他位置，请依实际情况调整。

要建构出一个支持 SASL 的 Postfix，你必须定义 USE_SASL_AUTH 宏，并指出函数库与标头文件的位置，而且必须与 *libsasl2.so* 函数库文件链接（该函数库的名称为“sasl2”）。请使用下列选项来产生你的 *Makefile*：

```
$ make makefiles CCARGS='-DUSE_SASL_AUTH -I/usr/local/include/sasl'
>  AUXLIBS='-L/usr/local/lib -lsasl2'
```

注意，如果你的编译器要求你提供动态函数库的位置给链接器，则 AUXLIBS 变量必须包含正确的运行时搜索路径：

```
$ make makefiles CCARGS='-DUSE_SASL_AUTH -I/usr/local/include/sasl' \
>   AUXLIBS='-L/usr/local/lib -lsasl2 -rpath /usr/local/lib'
```

倘若你的链接器并非使用 -rpath 来表示函数库搜索路径，请将上述的 -rpath 改成适当选项。

本小节描述的步骤，在 Postfix 包随附的 SASL README 文件里也找得到。如果你需要更进一步的说明，请参阅该文件。

TLS

关于 Postfix 与 TLS 的详细信息，请参阅第十三章。 Postfix 对于 TLS 的支持，并非由外界函数库提供，而必须修改 Postfix 本身的源程序才办得到。从 Postfix 正式网站的 Add-on Software 网页可以找到 TLS 修补文件的下载链接（在“ STARTTLS Support ”的下方）。由于这是‘ 修补文件 ”,你选择的版本必须与 Postfix 本身的版本相符。以本附录所示范的 Postfix 2.0.18 版为例，你应该下载 PFIXTLS 0.8.16 版：

```
[me@andy /usr/src/tarballs]$ wget \
> ftp://ftp.aet.tu-cottbus.de/pub/postfix_tls/pfixtls-0.8.16-2.0.18-0.9.7c.tar.gz
```

由于 PFIXTLS 需要用到 OpenSSL 函数库，所以你应该先安装该函数库。请参阅 PFIXTLS

随附的说明文件，确定你有正确版本的OpenSSL（译注 2）。我们假设你的 OpenSSL 函数库是安装在 */usr/local/ssl/lib/*，其标头文件是位于 */usr/local/ssl/include/* 目录下。如果你使用其他安装目录，请依实况调整。以下是我们的 OpenSSL 函数库安装步骤，仅供读者参考：

```
[me@andy /usr/src]$ tar xzvf tarballs/openssl-0.9.7d.tar.gz
[me@andy /usr/src/openssl-0.9.7d]$ ./config
(如果在 Sparc 64平台上的话，在 ./Configure linux64-sparcv9)
[me@andy /usr/src/openssl-0.9.7d]$ make
(这一步骤需较长时间)
[me@andy /usr/src/openssl-0.9.7d]$ sudo make install
```

PFIXTLS 的主要文件是 *pfixtls.diff*，此文件描述了 Postfix 的源程序应该怎样修改，才有支持 TLS 的能力。*pfixtls.diff* 文件是原作者在修改完 Postfix 源程序之后，使用 diff 命令比较原版与修订版之间的差异所产生的，俗称“差异文件”。与 diff 互补的命令是 patch，它能读取 diff 产生的“差异文件”，并依据该文件描述的差异之处，将原版文件改成修订版。在你解开 PFIXTLS 包之后，应该可以找到 *pfixtls.diff* 文件。请将 Postfix 与 PFIXTLS 的源文件都放在同一级目录下（例如 */usr/src/*），然后将工作目录切换到该处，也就是说，使用 `ls` 可以看到含有 Postfix 与 PFIXTLS 源文件的子目录：

```
[me@andy /usr/src]$ pwd
/usr/src
[me@andy /usr/src]$ ls -ld postfix* pfixtls*
drwxr-xr-x  5  me me  4096 Jul  8  2003 pfixtls-0.8.16-2.0.18-0.9.7c/
drwxr-xr-x  15 me me  4096 Jan 22 21:45 postfix-2.0.18/
```

就在 */usr/src/* 目录下，像下面这样使用 patch：

```
[me@andy /usr/src]$ patch -p0 < pfixtls-0.8.16-2.0.18-0.9.7c/pfixtls.diff
patching file postfix-2.0.18/Makefile.in
patching file postfix-2.0.18/conf/master.cf
...
patching file postfix-2.0.18/src/util/sdbm.c
patching file postfix-2.0.18/src/util/sdbm.h
[me@andy /usr/src]$
```

`patch` 会随安装过程的进行报告它正在修改什么文件，修改完毕后会回到命令行。只要 PFIXTLS 与 Postfix 的版本互相匹配，而且你的工作目录与 `-p0` 自变量都正确，修改过程应该不会出现任何问题。

现在，我们可以切换到 Postfix 包目录下，建构出一个支持 TLS 的 Postfix。在编译器选项方面（CCARGS 变量），我们必须定义 `HAS_SSL` 宏，并指出 OpenSSL 的标头文件的

译注 2：请使用 OpenSSL 0.97d 以后的版本，不要使用它原本建议的 0.97c 版，因为该版本有安全漏洞。（0.97d 是在 PIXTLS 0.8.16 版推出之后才出现的）

存放目录。在链接器选项方面（AUXLIB变量），我们得要提供 OpenSSL 函数库的存放路径，以及要链接的函数库名称。TLS需要用到 *libssl.so*（或 *libssl.a*）以及 *libcrypto.so*（或 *libcrypto.a*）这两个函数库，所以函数库名称分别为 ssl 与 crypto。总结以上信息，我们应该使用下列命令来产生 *Makefile*：

```
$ make makefiles CCARGS='-DHAS_SSL -I/usr/local/ssl/include' \
>   AUXLIBS='-L/usr/local/ssl/lib -lcrypto -lssl'
```

注意，如果你必须提供函数库路径给运行时链接器，记得 AUXLIB 变量应该包含正确的运行时搜索路径自变量（-rpath）：

```
$ make makefiles CCARGS='-DUSE_SSL -I/usr/local/ssl/include' \
>  AUXLIBS='-L/usr/local/ssl/lib -lcrypto -lssl -rpath /usr/local/ssl/lib'
```

倘若你的链接器并非使用 -rpath 来表示函数库搜索路径，请将上述的 -rpath 改成适当选项。

MySQL

关于 Postfix 与 MySQL 的详细信息，请参阅第十五章。要让 Postfix 具备查询 MySQL 数据库的能力，Postfix 本身必须与 MySQL client 与 zlib 这两个函数库链接。因此，你的系统上必须先安装这两个函数库。以下的示范过程，假设 MySQL client 函数库是安装在 */usr/local/lib/mysql/*，其标头文件位于 */usr/local/include/mysql/*；而 zlib 函数库是安装于 */usr/lib/*。如果你的函数库的安装位置不同于本例的假设，请依实际情况自行调整。

对于 MySQL 的支持能力，在编译器选项方面（CCARGS 变量），我们必须定义 HAS_MYSQL 宏，并指出 MySQL client 函数库的标头文件位置。在链接器选项方面（AUXLIB 变量），Postfix 必须与 *libmysqlclient.so*（即 MySQL client 函数库）、*libz.so*（zlib 函数库）与 *libm.so*（数学函数库，这是 Unix 系统上的标准文件）链接。总结以上信息，我们应该使用下列命令来产生 *Makefile*：

```
$ make makefiles 'CCARGS=-DHAS_MYSQL -I/usr/local/include/mysql' \
>  'AUXLIBS=-L/usr/local/lib/mysql -lmysqlclient -lz -lm'
```

注意，如果你的运行时链接器需要函数库搜索路径，记得 AUXLIB 变量应该包含正确的路径自变量（-rpath）：

```
$ make makefiles 'CCARGS=-DHAS_MYSQL -I/usr/local/include/mysql' \
>   'AUXLIBS=-L/usr/local/lib/mysql -lmysqlclient -lz -lm \
>   -rpath /usr/local/lib/mysql'
```

倘若你的链接器并非使用 -rpath 来表示函数库搜索路径，请将上述的 -rpath 改成适当选项。

Postfix 包随附的 *MYSQL_README* 文件，提供了如何建构一个支持 MySQL 的 Postfix 的更详细信息，如果你遇到困难，请参考该文件。

LDAP

关于 Postfix 与 LDAP 的详细信息，请参阅第十五章。要让 Postfix 具备查询 LDAP 目录的能力，Postfix 本身必须与 LDAP 函数库链接。在 OpenLDAP 网站（*http://www.openldap.org/*），可以购买到商业发行的版本，也可以免费下载 Open Source 版本。以下的示范程序，假设 OpenLDAP 的标头文件与函数库分别被安装在 */usr/local/include/* 与 */usr/local/lib/* 目录下。

对于 LDAP的支持能力，在编译器选项方面（CCARGS 变量），我们必须定义 HAS_LDAP 宏，并指出 LDAP 函数库的标头文件位置。在链接器选项方面（AUXLIB变量），Postfix 必须与 *libldap.so* 与 *liblber.so* 函数库文件链接。总结以上信息，产生 *Makefile* 的命令如下：

```
$ make makefiles CCARGS='-I/usr/local/include -DHAS_LDAP \
>  AUXLIBS='-L/usr/local/lib -lldap -L/usr/local/lib -llber'
```

提醒你，如果你的运行时链接器需要函数库搜索路径，记得 AUXLIB 变量应该包含正确的运行时搜索路径自变量（-rpath）：

```
$ make makefiles CCARGS='-I/usr/local/include -DHAS_LDAP' \
>  AUXLIBS='-L/usr/local/lib -lldap -L/usr/local/lib -llber \
>  -rpath /usr/local/lib'
```

倘若你的链接器并非使用 -rpath 来表示函数库搜索路径，请将上述的 -rpath 改成适当选项。

Postfix 包随附的 LDAP README 文件，更详细地说明了如何建构一个支持 LDAP 的 Postfix 系统，如果你遇到困难，请参考该文件。

常见问题

当你遇到编译问题时，应该先查阅 Postfix 随附的一系列 README 文件。通常，这些文件会告诉你，你可能会遇到怎样的问题，如何避免或解决。当然，如果你找到关于你的系统平台的 README 文件，那更是非看不可。本节针对 Postfix 建构过程中比较常见的问题提出解决的方法。

编译期的常见问题

“没有这样的文件或目录”

请确认编译器的路径是否正确。如果你的编译器路径来自 CC环境变量（例如`make makefiles CC="/path"`），请再检查一次你键入的路径；如果编译器的路径是来自 Postfix 的 *makedefs* 文件，请用正确的 CC 变量修正回来：

```
$ make makefiles CC="/path/to/your/compiler"
```

另一种办法，是由系统环境为 Postfix 找出编译器。请确定 PATH环境变量所列的搜索路径中，包含编译器本身所在的目录，然后将 CC 变量中的路径部分拿掉，只剩编译器本身的文件名：

```
$ make makefiles CC="cc"
```

“不能开启……文件”

请确定标头文件的路径是否正确。标头文件通常位于 */usr/include/*,如果你的系统因为特殊原因而使用不同的路径，则必须在 `CCARGS` 变量里的 `-I`选项指出正确位置：

```
$ make makefiles CCARGS="-I/path/to/include"
```

如果你已经用 `-I` 指出标头文件的路径，请再检查一次路径是否完全正确，看是否漏列了哪个路径。

“未解（或未定义）的符号”

这个问题通常是因为找不到函数库所造成的。请检查 `-L`选项指出的函数库路径是否正确，`-l` 选项是否逐一列出了每个要被链接的函数库名称（不是文件名）。

“来自 某标头文件的警告”

如果编译器是在处理 *mail_conf.h* 之类的标头文件时提出警告或错误信息，很有可能是你没使用符合 ANSI C标准的编译器。几乎所有系统平台配备的编译器，都被刻意设定成用于重新编译系统内核，但是这类编译器不见得完全合乎 ANSI C标准。你可能要接洽你的系统的厂商来取得 ANSI C编译器；几乎每一种系统平台都支持的 GNU `gcc` 编译器，也是个很好的选择。如果你使用 HP-UX 的原生编译器，记得使用 `-Ae`标记将它切换到 ANSI 模式：

```
$ make makefiles CCARGS="-Ae"
```

“不知道如何……”

你的 *Makefile* 可能不见了，或是你的 *Makefile* 不曾存在过。使用下列命令可产生一个全新的 *Makefile*：

```
$ make -f Makefile.init makefiles
```

完成之后，试着再重新建构一次看看。

运行时的常见问题

“无法加载共享函数库”

这问题的成因，在于运行时链接器找不到所需的动态函数库。而找不到的原因，一是函数库真的不存在，二是路径信息错误。请确定建构 Postfix 时，你指定了正确的 `-rpath` 或 `-R` 选项，而且所给的路径也都正确。如果你不确定自己的系统平台上的运行时链接器到底使用哪种选项，请查阅 `ld(1)`。

总结

建构 Postfix 的过程中，最烦人的步骤莫过于冗长的 `make makefiles` 命令。如果只是想添加一个外挂软件，偶尔为之或许还算可以接受；但如果想要建构一个同时支持 Cyrus SASL、TLS、MySQL 与 LDAP 的 Postfix 系统，事情将会有点棘手。其实我们可将所有编译条件整合在一起，并且写一个简单的 shell script 来自动产生 *Makefile*。以下是我们提供的 shell script：

```
# genmake.sh
# 产生用于建构 Postfix 的 Makefile
#

# 如果要清理掉前次编译留下来的多余文件，
# 请将下面这一行的注释符号拿掉：
# make tidy

# 或者，在运行本脚本之前，
# 自己动手运行一次make tidy，这样或许反而比较容易。
#
# 请依实际环境调整：
#
make makefiles \
 CCARGS='-DUSE_SASL_AUTH -DHAS_SSL -DHAS_MYSQL -DHAS_LDAP \
  -I/usr/local/include/sasl -I/usr/local/ssl/include \
  -I/usr/local/include/mysql -I/usr/local/include' \
 AUXLIBS='-L/usr/local/lib -L/usr/local/ssl/lib \
  -L/usr/local/lib/mysql -L/usr/local/lib \
  -lsasl2 -lcrypto -lssl -lmysqlclient -lz -lm -lldap -llber
  -rpath /usr/local/lib/mysql -rpath /usr/local/lib \
  -rpath /usr/local/ssl/lib'
# End of genmake.sh
```

有了这个 *genmake.sh* 脚本，只要三个简短的命令，就可以建构出一个全功能的 Postfix：

```
$ make tidy
$ sh genmake.sh
$ make
```

第一个命令清除前次编译过程留下来的多余文件，第二个命令产生符合我们需求的 *Makefile*，最后是执行实际的建构动作。如果顺利，就可以开始安装或升级旧版的 Postfix：

```
$ make upgrade          (升级旧版 Postfix)
```

或

```
$ make install          (第一次安装 Postfix)
```

附录四

问题集

我似乎收不到信。这段错误信息是什么意思？ *"<test@example.com>: mail for example.com loops back to myself?"*

当DNS的查询结果告诉Postfix它自己就是邮件的最终目的地，但是Postfix却没被设定成要收下该网域的邮件，这时候就会出现上述错误信息。只有列在mydestination、relay_domains、virtual_mailbox_domains、virtual_alias_domains中的网域，以及IP地址被列在inet_interfaces或proxy_interfaces中的网域，Postfix才会收下其邮件。你的网域必须被列于上述参数的其中之一。

当我改变配置文件或是修改查询表时，我需要重新加载*Postfix*吗？

这要看你所改变的文件的本质。原则上，如果被改变的文件是Postfix在启动之初就需要读入内存，那就需要重新加载。例如main.cf、master.cf以及任何含有正则表达式的查询表。至于DB或DBM文件这种在运行时才会被读取的文件，在你改完它们之后，它们就立刻生效了，所以不需要重新加载Postfix。

***main.cf*里有没有任何类似"*include*"作用的指令？**

没有。如果你控制管理的配置文件太复杂，复杂到需要将不同类型的参数分开在不同的文件里，一般的解决办法是写一个*Makefile*，在必要时将所有文件整合在一起。比方说，假设你有三个配置文件，那么你可以写一个像下面这样的*Makefile*：

```
main.cf: basic_conf adv_conf antispam_conf
        cat basic_conf adv_conf antispam_conf > main.cf.new
        mv main.cf main.cf.old
        mv main.cf.new main.cf
```

每当你需要重建配置文件时，只要键入make main.cf就可以了。事实上，同样的技巧也可以运用在其他例行性的管理工作上。例如，若你在修改了查询表或别名表之后，时常忘记使用*postmap*或*postalias*命令来重建数据库文件，这时候就很适合写一个*Makefile*来帮你完成这些事。

我如何确定邮件已经送达目的地？

Postfix 目前没有这样的功能。

如何附加一段声明稿（或其他文字）到每一封从我的邮件服务器寄出去的信？

按照 Internet邮件系统的设计，这不是 MTA 的工作，所以 Postfix 也没提供这种功能。不要把这个问题想得太简单了，虽然附加一段额外信息到普通的纯文本文件或 HTML邮件不是难事，但是如果将 MIME与数字签名考虑进去将困难重重。首先，MIME格式非常复杂，需要专门的分析、解读、编码程序。再者，数字签名保证了邮件的完整性，随便改动一个位，就必须重新签名才有效，这表示 Postfix得要有管理密钥的能力。MTA唯一有权修改的部分在于邮件的标题，不过，大部分的用户不会去注意标题的内容，所以将声明稿加在标题里是白费心机。真正的解决方案，应该在 MUA 层做这件事。

或许你会想到，使用内容过滤器将声明稿加到邮件末端应该是可行的办法。确实如此，不过，这样的内容过滤器并不容易写 —— 如果你要将 MIME与数字签名的问题都考虑进去的话。

我希望每一封流经服务器的邮件，都在某特定邮箱留下一份副本，可能吗？

可以的，只要将 `always_bcc` 参数指向一个邮件地址即可。每一封流经 Postfix 的邮件，都会以“密件抄送”的方式寄到指定邮箱。在一般的邮件系统上使用此功能有潜在的道德风险，请谨慎为之。毕竟，窥伺他人的邮件，不仅有违邮件系统管理员的职业道德，也违背社会道德标准。如果真的有必要这样做，请先告知用户。不过，如果 Postfix 是用来当成 MLM server （参阅第十章),那另当别论。

如何限制用户的邮箱容量以及单封邮件的大小？

单一邮件的大小上限为 `message_size_limit`，单一邮箱的容量上限为 `mailbox_size_limit`。请注意，如果使用 maildir格式的邮箱，`mailbox_size_limit` 参数限制的是个别邮件文件的容量上限，而非整个邮箱目录的容量上限，因为Postfix 是借助操作系统的文件功能来限制邮箱的容量。事实上，邮箱容量最好是由 Mail Store 或 POP/IMAP server 自己来控制，虽然 Mail Store 与 POP/IMAP server 多半也是靠操作系统提供的服务来施行管制。

当 *Postfix* 发出退信通知时，它告诉原寄件 者：*“For further assistance, please send mail to <postmaster> 。”* 但是，我希望信息中的*<postmaster>* 包含我的网域名称，也就是 *<postmaster@example.com>*，我该怎么办？

退信通知的用意，是要求用户向他们自己的邮件管理员求助，而原寄信方的本地邮件管理员是最有可能的人选。试想，如果你寄信到某个外国网域，而被对方退信了，你应该找自己的邮件管理人员帮忙，还是应该要找对方的邮件管理人员？如果你真的坚持，唯一的选择是修改 Postfix程序源代码。不过，事先警告你，当你真的这样做时，全世界的垃圾邮件发送者都会知道你的邮件地址。

当我寄信到一组别名地址时，似乎只有第一个地址能收到信，其他人只有在直接寄到他们的正式地址时才能收到信。但是当他们是别名的一部分时，邮件似乎不会进入他们的邮箱。

如果你使用外部程序来进行投递操作，它有可能无法一次处理多个地址。例如，maildrop 就有这种现象。若要确定 Postfix 一次只交付一封邮件给 MDA，请将 *main.cf* 的 `transport_destination_recipient_limit` 参数值改为 1，其中的 `transport` 是 MDA 在 *master.cf* 的传输服务名称。如果你使用 maildrop，其参数如下：

```
maildrop_destination_recipient_limit = 1
```

我的邮件系统有多个网络接口，我如何让 *Postfix* 只绑定 *(bind)* 其中之一？

将 Postfix 要绑定的接口的 IP 地址设定给 `inet_interfaces` 参数。

使用 *Sendmail* 时，如果邮件没于四小时内寄出，它会给我一封通知函。我如何在 *Postfix* 做到这一点？

此行为由 `delay_warning_time` 参数控制，其默认值为 0，表示不通知。

我想要测试别名表，检查哪个地址是从哪个别名展开出来的。使用其他 *MTA* 时，我习惯使用 *SMTP* 的 *EXPN* 命令来得到完整的有效收件人列表，但是 *Postfix* 似乎不理会这个命令。

没错，Postfix 确实不支持 EXPN，而且是刻意不支持。因为要实现出 EXPN 命令的效果，`smtpd` 需要能够访问别名表，但是由于 Postfix 本身的架构设计以及基于安全性的考虑，没有特权身份的 `smtpd` 无法访问别名表。在投递期间，负责将别名展开成正式地址的是 local MDA，也就是说，真正具有特权身份的是 local，不是 `smtpd`。如果你使用 MLM，MLM 应该会有一个能提供完整列表的命令，不然，就请检查邮件系统上的别名表吧！

谁能告诉我 `mailbox_transport` 与 `mailbox_command` 的差别？

`mailbox_transport` 是指向 *master.cf* 所设置的服务之一；而 `mailbox_command` 是指向外部程序，也就是你可以在邮件服务器的文件系统上找到的具体独立程序，例如 procmail。可能影响邮箱投递操作的参数有好几个，依照效力高低，这些参数依序是 `mailbox_transport`、`mailbox_command_maps`、`mailbox_command` 与 `home_mailbox`。

我们内部网络上的所有机器，都是通过同一台邮件网关寄出邮件。有没有办法在邮件出门之前，消除掉标头中涉及内部主机名称或 *IP* 地址的 *Received:* 字段？

以 `header_checks` 找出符合条件的标头字段，并使用 IGNORE 动作来处理它们。

我如何要求 *Postfix* 将外界寄给不明用户的邮件全部转寄到特定邮箱？

简单。将 `luser_relay` 参数指向该邮件地址，然后取消 `local_recipient_maps`：

```
luser_relay = info
local_recipient_maps =
```

不过，这样做会有严重后果，请三思而后行。以垃圾邮件无所不在的本质，你指定的邮箱很快就会被垃圾邮件塞满。

依照我的配置，*Postfix* 应该回复永久性错误码 *(554)*，但是它却总是响应暂时性错误码 *(454)*。怎么回事？

你或许打开了 soft_bounce。

我的队列里堆积了一大堆邮件，而我确定这些邮件都是垃圾。有没有任何办法可以一次清空队列里的所有信息？

```
# postsuper -d ALL
```

注意，*ALL* 关键字必须全部大写，因为这个命令会删除掉队列中的全部邮件。

哪里可以找到 *Postfix* 的日志文件？

Postfix 是通过本地系统的 syslogd daemon 来记录日志信息的。各个操作系统的日志文件位置都不太一样，但多半是在 */var/log/* 目录下。详情请参阅你的系统说明书，必要时，检查 */etc/syslog.conf* 的内容是否正确。

Postfix 似乎总是忽略 *MX* 记录，而直接将邮件递送到 A 记录所指的系统。这样算正常吗？

正常。如果你的 *main.cf* 是这样设定：

```
disable_dns_lookups = yes
```

另一种可能是你设置了传输表，而收信网域的名称被放在一对方括号里（这会使得 Postfix 将邮件直接递送到该系统）：

```
example.com      smtp:[mail.example.com]
```

我收到一大堆含有空白回信地址的垃圾邮件（译注 *1*），我如何挡掉这些邮件？

使用 head checks 就可以办到。不过，你不应该这样做，因为 RFC 标准规定你必须收下这类的邮件。这一技术是用于避免错误通知函造成邮件循环。你应该依据其他准则来过滤垃圾邮件。

我使用 header checks 和 body_checks 来阻挡垃圾邮件，但是，不管过滤条件再怎么谨慎，偶尔仍会发生误判合法邮件的事件。*Postfix* 有没有提供任何类似“白名单”*(whitelist)* 之类的机制，好让某些邮件可以躲过标题检查与正文检查？

没有。header_checks 和 body_checks 会影响所有经过 Postfix 的邮件，它们只用于简单的、统一的检查。如果你需要更细腻的控制，或许内容过滤器是比较理想的选择。

译注 1：即为 Return-Path:<>。

索引

本篇索引可以帮助读者迅速找出特定信息在本书中的页码。所有项目都是依照字母顺序排列，所以读者可以像使用英文字典那样使用本索引。符号“……”代表主关键词，例如：

access map（访问表）
以……检查客户端，151, 151-155
检查……之后的动作， 152
配置范例，153
正则表达式， 153

在此例中，“以……检查客户端， 151-155”表示读者可在 151 到 155 页找到关于“以访问表检查客户端”的信息，其中 152 页描述“检查访问表之后的动作”，而 138 页提供了范例。在 158 页可找到如何在访问表里使用正则表达式。

为了方便检索，也为了编排上的考虑，原则上，顶层关键词不翻译，只加注中文解释（如果有的话），对于比较简单的字词，甚至不加注中文解释。第二层与第三层的关键词则不保留原文或不翻译，视情况而定。

希望本节的说明有助于读者使用本篇索引。如果读者对本公司书籍的索引编排方式有任何意见，请用 E-mail 告诉我们，好让我们知道如何改进。

欢迎提出改进索引方面的建议。请发邮件至 *info@mail.oreilly.com.cn*。

符号

* （星号），于活动队列中的邮件，75
\（反斜线），延续 Unix 的过长命令行，20
% （命令提示符），19
$ （美元符号）
  命令提示符，19
  配置宏变量，42
!（感叹号）
  标示保留队列中的信息， 75
  避免改写域名部分，66
?（问号），标示唤醒时间，62
"（双引号）
  别名定义，50
  查询表与……，44
  参数值与……，41
（root 提示符），19
／（斜线）
  表示文件路径，42
  标示正则表达式， 48
| （垂直线），作为别名目标的命令，50

数值

2bounce_notice_recipient 参数，220

A

A 记录，82
  没有……的网域，88
  ……与 MX主机，85
  ……与 MTA 的邮件路由，84
access map（访问表）
  以……检查客户端，156，156-159
    检查……之后所采取的动作， 157
    配置范例，158
  描述正则表达式的 ……，158
access_map_reject_code 参数，220
account name（账户名称）
  避免伪装特定……的邮件地址， 66
active attack（主动攻击），181
active queue（活动队列）， 31， 35，71， 72
  加注 * 的邮件，75
additional_conditions 参数， 210-212
address classes（邮件地址分类），31
  全部伪装，66
address（邮件地址）
  关闭自动补齐，65
  作为别名目标，50
  阻挡来自特定……的垃圾邮件（参阅 "spam"）
  client-based 过滤规则，……限制条件，157
  cleanup daemon 的自动修正动作，35
  制作邮件列表文本文件，130
  删除队列中特定……的邮件，76
  ……于邮件标头中的格式 RFC 2822），22
  trivial-rewrite daemon 的处理操作，29
  辨识来自……的垃圾邮件，146
  被垃圾邮件发送者假冒的合法回信地址，144
  改写，64-68
    使用 canonical_maps 查询表，44
    规范地址，64
    伪装主机名称，66
    更换其他…的用户， 68
    不明用户，68
  寄件人与收件人，在 SMTP 对话过程中，151
administration （管理），54-59
  日志记录，56
  供服务器进程使用的虚账户，19
  队列管理（参阅 "queue manager"）
  ……与 root权限，19
  在开机时启动 Postfix，58-59
    写出一个启动脚本，58
  启动、停止、重新加载 Postfix，57
administrator E-mail,（管理员的邮件地址，postmaster）， 21
agents for E-mail （邮件系统代理程序）
  各种 ……， 21
  （参阅 MDA、MTA、MUA）
alias file （别名文件），49
  建构别名数据库文件，49
  投递到外部命令或是……所指定的文件，95
  ……的格式，50
  重要的别名，51
  别名文件的位置，49
  别名目标的限制，51

alias database 参数，50，131
增加 Mailman 别名文件到……，140
增加 Majordomo 别名文件到……，136
alias（别名），22
local MDA 的检查，33
主机名称，……的规范名称（CNAME），82
MX 记录的主机，85
mailing list（邮件列表）
创建，129
拥有者，129
传送邮件到测试列表，132
Mailman 创建存储……的文件，140
制作 Majordomo 的别名文件，136
……的系统数据库、正确格式，39
virtual（虚拟），107
……无限别名地址，109
……查询表，115
virtual alias（虚拟别名）
地址，33
查询表，105
alias_maps 参数，49，129，220
增加 Mailman 别名文件到……，140
用于邮件列表的额外别名文件，131
编辑……以增加 Majordomo 别名文件，136
LDAP 设定，215
allow_mail_to_command，51
allow_mail_to_files 参数，51，221
allow_percent_hack 参数，221
alternate_config_directories 参数，221
ANONYMOUS 验证机制，175
anonymous login（匿名登录），182
append_at_myorigin 参数，221
关闭地址自动补齐，65
append_dot_mydomain 参数
关闭地址自动补齐，65
approve 命令（Majordomo），133
architecture（结构），Postfix，28-37
组件，28
邮件如何进入 Postfix 系统，29-31
转寄（forward），31
通知函，31
提交本地邮件，29
网络邮件，30
mail delivery（投递），31-35
local（本地），33
其他 MDA，34
转发邮件，33
虚拟别名邮件，33
虚拟邮箱邮件，33
queue manager（队列管理器），31
邮件在……中的处理流程，35-37
arrival time（抵达时间），在队列中的邮件，75
ASCII 字符，于邮件正文中，23
attack（攻击）
主动，181
字典（参阅 dictionary attack）
分布式 DoS 攻击，使用受胁持系统进行……，146
AUTH SMTP 命令，183
authentication（验证）
certificate（证书），56，187
（参阅 TLS）
客户端证书，193-195
framework（架构），174
选择，176
在 Postfix 指定 SASL 使用的……，177-178
identity（身份），183
IMAP server，Cyrus SASL 函数库，100
mechanism（机制），174
选择，174
SASL（参阅 SASL 验证）
SMTP，185
转发控制，55
authoritative domain nameserver（权威域名服务器），82
authorization identity（授权身份），183
authorized_verp_clients 参数，221
automatic reply program（自动回信程序），111-113
在虚拟网域上架设……，114-116
auxiliary property plug-in（辅助性专属外挂），178

B

backup MX（备用交换器），118-121
base64 编码法，验证数据，183

Berkeley DB, Cyrus IMAP, 100
berkeley_db_read_buffer_size 参数，221
biff，222
bin 与 daemon （虚账户），19
binary 格式，alias 文件，39
BIND（DNS服务器软件），83
blacklisted site （黑名单）
DNS-based Blacklist（DNSBL），146
检查实时黑名单，156
实时黑名单
基于客户端的检查条件，162
body of E-mail message （邮件正文），22
在垃圾邮件辨识过程中检查……，151
使用 body_checks进行内容过滤，198
body_checks 参数，48，165
模式匹配，167
body_checks_size_limit 参数，167，222
bounce daemon，31
bounced message（退信），21
mailing list（邮件列表），132
寄给不明用户的垃圾邮件，144
bounce_service_name 参数，222
bounce_size_limit 参数，72
broken_sasl_auth_clients 参数，180
btree（查询表数据库），45
building Postfix（建构 Postfix），16，253
customizing your build，255
修改默认值，255
（参阅 compiling Postfix）

C

C 程序，编译，250
CA（参阅 Certificate Authority）
canonical address（规范地址），64
canonical domain（规范网域），103
canonical names for hostname alias（主机别名的规范名称），82
canonical_maps 参数，44，46，64，222
指定对查询给……，105
Carnegie Mellon University Cyrus IMAP，98
catchall address（无限别名地址），109
virtual alias（虚拟别名），109
virtual mailbox（虚拟邮箱），109
Certificate Authority（CA），188
……签署的客户端证书，193
公钥的数字签名，189
代表……的公共证书（根凭证），192
certificate signing request（CSR 证书签署请求），190
certificates（证书），187
依据……验证，56
CA 安装……，191
TLS，188
自己开设 CA，188
客户端，193-195
公钥密码学，……的用途，189
check_client_access 限制条件，157
check_helo_access 限制条件，157
check_recipient_access 限制条件，157
check_sender_access 限制条件，157
chroot，15，69
执行适合你的系统的正确脚本，69
于 master.cf 文件，61
将 Postfix 局限在……环境下运行，DNS 文件与……，89
class（类）
address（地址），31
error（错误），74
class_notice_recipient 参数，74
class_transport 参数，34
cleanup daemon，29，30
错误信息，检查，31
修正邮件地址，35
queue manager 收信通知，31
client certificate （客户端证书）
辨识名称，195
制作，193
……的指纹，194
client error （客户端的错误），逐渐频繁，63
client access文件，159
client-based spam detection，146
设定……的过滤规则，149
以限制条件类定义过滤规则，150
DNS-based 黑名单，146
过滤规则，设定限制条件到……，150
访问表，156-159
DNS 限制条件，160
一般限制条件，163
限制条件的运作原理，153
限制条件列表，152

其他限制条件， 159
实时黑名单，162
限制条件定义， 156
SMTP对 话，150
SMTP 过滤规则与限制条件， 152
严格语法限制条件， 160
测试新的限制条件， 154
垃圾邮件发送者的手段， 148
clientcerts 文件， 194
client/server验证，认同使用同一种……机制，175
client-side 证书验证（参阅 certificate）
CNAME记 录，82，85
command_directory 参 数，222
command-line tools （命令行工具）
管理证书的……， 188
队列，75-80
删除邮件，76
显示邮件，78
清空邮件， 80
保留邮件，77
列出邮件， 75
重排邮件， 78
解析 网域 名 称，86
command （命 令）
作为别名目标，50
内容过滤，199，199-201
配 置，200
投递到……，110-117
自动回信程序， 111-113
设定虚拟网域的自动回信程序，114-116
设定虚拟网域的 MLM，116
运行你自己的 pipe daemon，62
Postfix，248
默认目录，38
在 alias文件中指定……，投递到……，95
command_time_limit 参数，222
comment（注 释，于 main.cf文件），41
common name （辨识名称）
客户端证书， 195
公钥，189
compile time error（编译期错误），264
compiling Postfix
附加包，259
Cyrus SASL，260
LDAP，263
MySQL，262
TLS，260
常见问题，263
编译选项，251
链接选项，252
入门，250
用于产生Makefile 的脚本，266
（参阅 building Postfix）
component（组 件），Postfix，28
concurrent delivery attempt（同时投递），73
configuration（配置），38-70
chroot 环境，69
默认值，如同 Unix mail server，38
默认目录，38
MTA标识， 52
mydestination 参 数，54
myhostname 与 mydomain 参数，52
myorigin 参数，52
参数，219-247
Postfix反垃圾邮件范例， 171
Postfix/TLS，191
设定步骤汇整，192
接收限制，63
relay control （转发控制），54
动态 IP 的解决方案，55
限制转发访问，54
SMTP验证，55
relay （MDA） 控制
证书验证，56
改写地址， 64-68
规范地址，64
伪装主机名称， 66
更换地址的用户，68
不明用户，68
垃圾邮件检查，149-172
垃圾邮件判别依据的分类，149
客户端判别规则，150-164
内容检查，165-168
自定义限制条件组合，168
严格语法参数， 164

启动 Postfix，第一次，39
aliases 文件，系统， 39
主机名称，39
（参阅 configuration file）
configuration files（配置文件），40-44
alias（别名文件），49
alias 数据库，建构， 49
……的格式， 50
重要别名，51
别名文件的位置， 49
别名目标的限制， 51
BIND 网域，83
Cyrus，100
默认值目录，38
/etc/postfix/ 目录，40
查询表，44-48
数据库格式，45
……的格式，44
其他格式，48
使用列表的参数，47
正则表达式，使用， 47
比对顺序，46
main.cf 配置文件， 41
配置变量，42
延续前一行，42
多值参数， 42
majordomo.cf，135
master daemon（main.cf与 master.cf）， 28
master.cf（参阅 master.cf文件）
mysql-local.cf，211
样本， 41，70
content filtering（内容过滤），198-206
地址改写的影响，205
command-based，199-201
……的配置，200
设定外部过滤程序，199
daemon-based，201-205
……的配置， 202-204
……的范例， 204
mail delivery agent（MDA）， 198
mail transfer agent（MTA）， 198
mail user agent（MUA）， 198
MTA 与 MUA filter，结合，199
Postfix 的 body 与 header检查，198
串联多层过滤器，205
content filter（内容过滤器），198
检查客户端访问表之后发送邮件到……，158
在内容检查之后发送邮件到……， 167
用于侦测垃圾邮件，165
content-based spam detection，148
模式匹配，167
设定 Postfix 进行……， 150
内容检查动作，166
内客检查配置，165
内容检查参数，165
以内容过滤器标示垃圾邮件， 149
垃圾邮件发送者的手段，148
content filter参数，201， 223
启动 daemon-based filtering，204
continuation of lines （跨行）
邮件标头字段，23
在查询表中，44
在 main.cf 配置文件中，42
corrupt queue（毁损队列）， 31， 71， 74
credential（证书），183
用于交换……的验证机制， 174
以 TLS 加密，187
OTP验证，用于 SMTP client，175
cryptography （密码学），公钥，189
客户端证书， 193
产生服务器端证书， 190
CSR（certificate signing request）， 190
cur/ 目录，93
Cyrus IMAP
Postfix与……，98
范例，100
服务器，98
Cyrus SASL 函数库，100，173，176
saslauthd 验证服务器，177

D

daemon（虚账户），19
daemon_directory 参数，62
daemon，28
chrooted，使所有资源可在……环境下取得，69

内容过滤，199，201-205
配置，202-204
……的范例，204
…… 的默认 Postfix 目录，38
master daemon control by，40
……的选项，62
daemon_timeout 参数，223
DATA 命令（SMTP），25，151
database format（数据库格式），查询表，45
database（数据库），外部，207-218
LDAP，215-218
配置，215
Postfix/LDAP 配置范例，216
MySQL，208
配置，208
MySQL/Postfix 配置范例，210
days（d） 62
dbm（查询表数据库），45
dbname 参数，209
debugging（调试）
域名解析，86
产生用于……的信息， 62
调查 chroot 环境下的错误，70
debug_peer_list 参数，223
default_database_type 参数，45，50
default_destination_concurrency_limit 参数，73，223
default_destination_recipient_limit 参数，74
default_extra_recipient_limit 参数，223
default_privs 参数，51
default_process_limit 参数，62，73，224
default_recipient_limit 参数，224
default_verp_delimiters 参数，224
defer daemon，31
deferred message（延迟邮件）
延迟投递，34，125
延迟转发，124
无法投递的原因，75
在队列中的时间，指定， 119
deferred queue（延迟队列），31，71，72
重递尝试，调度，72
defer service name 参数，224
defer_transports 参数，125
definition（定义）
别名的……，50
参数，在 main.cf 中，41
delay_notice_recipient 参数，224
delay（延迟）
每次客户端错误之后，63
安排延迟邮件的重新投递时间， 72
deleting queued message（删除队列中的邮件），76
Delivered-To:标头字段，133
deliver_lock_attempts 参数，224
delivery agents for E-mail（参阅 MDA）
delivery of mail（邮件投递操作），28，31-35
禁止／允许……到命令与文件， 51
queue manager 的处理，73
毁损邮件，74
错误通知函，74
本地邮件，33
To:标头字段与……， 22
多收件人，14
其他 MDA，34
LMTP，34
pipe daemon，35
Postfix，13
转发邮件，33
垃圾邮件，标示，149
虚拟别名邮件，33
虚拟邮箱邮件，33
delivery transport（投递传输），90
denial-of-service attack（DoS 拒绝服务攻击），16
分布式，146
dictionary attack（字典攻击）
luser_relay 参数，使用， 68
无字典密码机制，181
dig 工具程序，86
DIGEST-MD5 机制，175
digital signature（数字签名）， 189
directory（目录）
默认，在 Postfix 中，38
Postfix 配置文件，40
disable_dns_lookups 参数，226
disable_mime_output_conversion 参数，226
disable_vrfy_command 参数，226
discarding message（丢弃邮件），158
在内容检查之后，166
displaying queued message（显示队列中的邮件），78

distinguished name（DN辨别名称），193
distributed denial-of-service attack（分布式拒绝服务攻击），146
DNS（Domain Name System域名系统），81-89
以……为基础的黑名单，146，162
虚拟网域……的配置，104
……的定义，81
E-mail routing，82-85
BIND配置文件，83
MX记录，83
主机名称查询问题，88
MX 记录
backup MX，118-121
概论，81
主机名称层次结构，81
网域的资源记录，82
收件与……，87
寄件与……，85
Postfix 配置选项，86
PTR 记录，87
DNSBL（DNS-based Blacklists），146，162
documentation（说明文件）
Postfix 包随附的……，70
Unix（man page），20
Domain Name System（参阅 DNS）
domain name（网域名称）
于 E-mail地址中，22
完全合格的……（FQDN）
以……为拒绝客户端的条件，160
严格语法限制条件，160
主机名称伪装与……，66
对于本地投递，33
myorigin 参数，将……附加到不完整邮件地址末端，64
转发网域，34
对于虚拟别名，33
对于虚拟邮箱，33
domain（网域）
权威域名服务器，82
fast_flush_domains 参数，120
代管多个……，103-117
投递到命令，110-117
邮箱文件所有权，106
独立网域搭配系统账户，104
独立网域搭配虚拟账户，105-110
独立邮箱，110
共享网域搭配系统账户，104
……的邮件交换器（MX记录），82
在查询表中匹配……，47
处理……的参数，52
relay，33
资源记录数据库，82
指定 SASL 用户账户的网域，178
DoS（denial-of-service拒绝服务攻击），16 146
dotlock（锁定类型），92
double_bounce_sender 参数，226
DRAC（Dynamic Relay Authorization Control），55

E

egrep 命令，以……寻找 Postfix 邮件日志，57
EHLO 命令（SMTP），27
严格语法，164
E-mail
DNS与……，81-89
Internet，11-13，20-27
agent（代理制度），21
DNS 与 routing E-mail，82-85
信封地址与邮件标头，22
地址的格式，22
收件限制，63
MDA（message delivery agent），12
格式（RFC 2822），22
MTA（mail transfer agent），11
MUA（mail user agent），11
POP/IMAP，邮箱访问与……，13
Postfix安全性，15
postmaster，21
协议，12
拒收或退信，21
RFC（Request for Comments），20
SMTP，23-27
软件包，12
empty_address_recipient参数，226

encode_sasl_plain（Perl 脚本），184
encoding （编码）
　编成 base64格式的身份数据，183
　交换身份数据，175
加密，TLS，187
end of E-mail message（邮件结尾），在 SMTP 中表示，27
enhanced SMTP（ESMTP） 27
envelope address （信封地址），22
　地址伪装与……，66
　垃圾邮件发送者冒充……， 55
　SMTP RFC 的严格格式标准，165
error（错误）
　代码 SMTP，86
　编译期，264
　E-mail 通知函，74
　邮件列表，发送通知函给列表拥有者，129，132
　关于延迟或退信的通知函， 31
　运行时，264
error_service_name参数，227
ESMTP（enhanced SMTP），27
/etc/passwd文件，18
ETRN 命令，120
expand_owner_alias 参数，130
expansion of incomplete E-mail address （补齐不完整邮件地址），关闭，65
export_environment 参数，227
external databases（外部数据库），207-218
　使用查询值，48

F

fallback_relay 参数，227
fallback_transport 参数，99
false-positive spam identification（误判），145
fast flushing（快速清空），80，120
fast_flush_domains 参数，80，120，227
fast_flush_refresh_time 参数，227
fcnt （锁定类型），92
fifo（队列），60
file locking（文件锁定），92
file permissions（文件权限），Majordomo 与……，137
filenames as alias target（以文件名为别名目标），50
filter_destination_recipient_limit 参数，201
fingerprints for CC（客户端证书的指纹），194
flock（锁定类型），92
flush daemon，120
　唤醒，62
flushing queued 邮件，80
　速清，120
fork_attempts参数，227
.forward 文件，94
forward_expansion_filter 参数，228
forwarding E-mail（转寄邮件），31
　别名与……，51
　local MDA 的处理，33
　本地投递，94
　虚拟别名邮件，33
　（参阅 alias files、alias）
forward_path 参数，94
frequently asked questions（常见问题），267-270
fully qualified domain names（FQDN），严格语法条件，160
fully qualified hostname（完整主机名称），52
　过滤条件，160
　系统，39

G

gateway（网关）
　入站邮件，125
　出站邮件，127
generic restriction rules （一般过滤条件），156，163
gethostname() 函数，52
group （组）
　投递到虚拟邮箱文件， 107
　调用 Mailman 的进程的 GID，138
　postdrop 组，39

H

hard link（硬链接），chroot 与……，70
hash type（查询表数据库），45
hash_queue_depth 参数，228
header checks （标头检查）
　模式表，167
　以……过滤内容，198

header_address_token_limit参数，228
header_checks 参数，48，165
在文件中描述正则表达式（模式表），166
header（邮件标头）
地址伪装，66
垃圾邮件检查规则，151
Delivered-To:，133
……中的字段，22
trivial-rewrite daemon 安插的字段，29
邮件列表信息，……的范例，129
……中的 To:地址，22
header_size_limit 参数，228
HELO 命令（SMTP），25
严格语法参数，164
限制条件组合，追查，163
smtpd_helo 限制条件，151
hiding names of internal host（隐藏内部主机名称），66
hierarchical naming of host（主机名称的层次结构），81
hold queue（保留队列），71
以！符号标示的邮件，75
搬移邮件进入……，77
从……搬出邮件，78
检查客户端访问表之后将邮件存入……，158
在内容检查之后将邮件存入……，166
home_mailbox 参数，95，228
host（主机），121
目的地，于 inet传输，121
inet socket，60
查址问题，88
工具，86
（参阅 DNS、hostname）
hosting multiple domains （代管多个网域），103-117
投递到命令，110-117
自动回信程序，111-113
设定虚拟网域的自动回信程序，114-116
设定虚拟网域的 MLM，116
邮箱文件所有权，106
在……方面的 Postfix 配置，103
独立网域搭配系统账户，104
独立网域搭配虚拟账户，105-110
无限别名地址，109
邮箱文件所有权，106
虚拟别名，107
独立邮箱，110
共享网域搭配系统账户，104
hostname 命令（Unix），39
hostname （主机名称）
严格语法限制条件，160
client-based过滤规则，限制条件，157
常态联机的 SMTP client，HELO 送出的……，151
邮件交换器，在 MX 记录中，85
辨识来自……的垃圾邮件，146
对应到 IP地址，82
伪装，66
在 SMTP 对话过程中阻挡垃圾邮件，150
系统
完整名称，39
处理……的参数，52
hour（h），62
https，189

I

ignore_mx_lookup_error 参数，230
IMAP，12
Cyrus IMAP，98
与 POP 的比较，96
（参阅 POP/IMAP）
inbound mail gateway（入站邮件网关），125
include files（引入文件）
作为别名目标，50
安全风险，51
incoming E-mail （入站邮件），限制，63
incoming queue（入站队列），31，71
incomplete E-mail address （不完整邮件地址），关闭自动补齐，65
inet socket
用于 LMTP server，99
in_flow_delay 参数，230
info 文件，136
initial_destination_concurrency 参数，73，230
initialization scripts for Unix systems（开机脚本），58

input/output（输入 / 输出）
标准输入与标准输出， Unix，19
installing daemon-based content filter（安装基于守护进程的内容过滤器），203
installing Postfix（安装 Postfix），38，256-259
upgrading（升级），259
Internet
E-mail与……，11-13
主要邮件协议，12
MDA（message delivery agent），12
MTA（mail transfer agent），11
MUA（mail user agent），11
Internet Engineering Task Force（IETF）
网站，21
Internet Mail Application Protocol（参阅 IMAP）
IP 地址
client-based 过滤规则，限制条件， 157
动态，客户端 SMTP 验证，55
邮件交换器，在 MX 记录中，85
辨识来自……的垃圾邮件，146，150
映射主机名称到……（参阅 DNS）
搭配……的 PTR 记录，87
远程用户的……， 55
从……反查主机名称， 82
ipc_ lle参数，230

K

Kerberos验 证 ，149
key agreement（密钥协议），190
key/value pairs（键 –值对）
在规范查询表中，44
在一般查询表中，格式，44

L

LDAP，48，215-218
编译，263
配置，215
目录，216
Postfix/LDAP 配置范例，216
Postfix 对……的支持，检查， 207
left hand side（or LHS）of E-mail addess（邮件地址的人名部分），22
line continuation （续行）
在查询表文本文件中， 44
在 main.cf 配置文件中， 42
line_length_limit 参数，167，230
link files（链接文件），chroot与…… 70
listing mail in queue（列 出队列中的邮件），75
list（列表），可接受……的参数，44
查询表与……，47
lmtp delivery agent，37
LMTP（Local Mail Transfer Protocol） 34，97-101
Postfix与 Cyrus IMAP，98
Postfix与 Cyrus IMAP 范例，100
lmtp_connect_timeout参数，230
lmtp_data_init_timeout 参数，231
lmtp_lhlo_timeout 参数，231
lmtp_quit_timeout 参数，231
lmtp_tcp_port参数，231
local地址，31
LHS别名，50
local delivery （本地投递）
alias（别 名），95
在 mydestination 参数列 出网域名称，93
.forward 文件，94
Local Mail Transfer Protocol（参阅 LMTP）
邮箱，95
邮箱格式，91-93
maildir，92
mbox，91
mbox 与 maildir 的比较，93
收件地址，……的列表， 94
local delivery agent，33，35，37
local delivery transport（本地投递传输），90
local domains （本地 网域），87
虚拟邮件列表，……的平行版本，116
local E-mail submission（回到 Postfix），29
local part（邮件地址的人名部分），22
my_origin 参数，64
在查询表中搜索……，46
local_destination_concurrency_limit 参数，74 231
local_recipient_maps 参数，94，232
以 LDAP 为信息来源，217

以 MySQL 为信息来源，211
避免拒收不明用户的邮件， 68
local_transport 参数
LMTP 与 Unix domain socket，使用，99
从 Postfix 传递邮件到 Cyrus IMAP server，99
locking files（锁定文件），92
logging（日志记录），56
邮件日志文件，57
Postfix 启动问题，40
SMTP，151
syslog daemon（syslogd），57
输出详细信息，开启， 62
LOGIN 验证机制，174
login names（登录名称），18
查询表，44-48，219
访问表，156
别名表（参阅 alias files）
指定…… 给参数，46
指定给 canonical_maps，105
规范地址，映射，64
canonical_maps，44
内容检查，165
正则表达式，165
数据库格式，45
……的默认目录，38
动态更新用户的 IP 地址，55
……的格式，44
其他格式，48
可接受列表的参数，47
正则表达式，47
地址或网域的改变，68
在……中的比对顺序，46
寄件人地址与 SASL 登录身份的对应关系，180
传输表，126
虚拟邮箱地址，33
虚拟别名地址，33，115
luser_relay 参数，68，232

M

mail（邮件）
收件，限制， 63
转发， 118-127
备用交换器， 118-121
客户端验证，173
入站邮件网关， 125
出站，126
传输表，121-125
（参阅 E-mail、Internet）
mail delivery agent（参阅 MDA）
mail delivery loop（邮件循环）
Majordomo 审阅核准与……， 133
以 myhostname 值避免……，203
邮件交换器
A 记录，85
别名与……，85
备用交换器，118-121
快速清空，120
转发收件人，119
DNS MX 记录，83
（参阅 MX record）
优先程度，84
以主机名称取代 IP 地址（于 MX 记录中），85
使用 Postfix server 架设 MX 交换器， 88
MX 记录里的优先值，85
MAIL FROM 命令（SMTP）25，151
检查客户端提供的地址，157
查不出……的有效 DNS 记录，156
reject_unknown_sender_domain 过滤规则与……，162
mail logfile （邮件日志文件），57
mail server DNS 与……，81
mail transfer agent（参阅 MTA）
mail user agent（参阅 MUA）
mailbox access（邮箱访问，参阅 message stores、POP/IMAP）
mailbox delivery（邮箱投递），95
mailbox file ownership（邮箱文件所有权），106
mailbox names （邮箱名称），需求（RFC 2142），51
mailbox_command 参数，232
mailbox_delivery_lock 参数，232
mailbox_transport 参数，232
Postfix 传递邮件到 Cyrus IMAP，99
maildir 格式，92
设定 Postfix 使用……，95

与 mbox 格式的比较，93
虚拟邮箱文件，106
maildrop/ 目录，29，35
mailing list for virtual domain （在虚拟网域架设 MLM），116
Mailing List Manager（参阅 MLM）
mailing list（邮件列表），128-142
额外别名文件，131
创建简单的……，131
MLM，133
Mailman，138-142
Majordomo，133-138
……的拥有者，129
独立列表文件，130
简易式，使用 alias 制作，129
测试，132
Mailman，138-142
configure，指定 mailman 使用的 GID，138
创建列表，139
alias 存于独立文件，140
使用 newlist 初始化列表，140
父进程的 GID，138
Python 基本要求，138
mailman 账户与组名，138
mail owner 参数，61，232
mailq 命令，75
mail_spool_directory 参数，95，232
main.cf 配置文件，40，41
编修，重新加载 Postfix，44
个别组件的配置，以 master.cf 改写……里的同名参数，63
配置变量，42
收件的限制，设定，63
续行，42
多值参数，42
指向含有……的文件，42
过滤规则 Postfix UBE），默认值，152
严格语法参数，164
Majordomo，133-138
别名表，116
创建列表，135
别名数据库，136
aliases 文件，136
给新成员的 info 信息文件，136
Majordomo 的取得与安装，135
Perl 用于检查，135
文件权限的潜在问题，137
订阅列表，137
测试，136
majordomo.cf 文件，135
malicious program sending garbage command （发出垃圾命令的恶意程序），64
man pages（在线说明页），20
Postfix 包随附的……，70
man-in-the-middle attack（中间人攻击），181
manpage_directory 参数，233
MANPATH 变量，70
map type（查询表类型），45
别名文件，49
模式表，48
masquerade_classes 参数，66
masquerade_domains 参数，66，233
避免网域名称被裁掉，66
隐藏／显示子网域的设定，66
masquerade_exceptions 参数，66
masquerade hostname（伪装主机名称），66
master daemon，28，40
master.cf 配置文件，40，59-63
chroot，61
chrooting 个别组件，69
命令（执行服务），62
过滤 smtpd 收进来的邮件，201
maxproc，62，73
改写 main.cf 里的配置信息，63
private，61
样本文件，60
服务名称，60
传输，121
传输类型，60，73
unpriv，61
wakeup，62
matching pattern（匹配模式，参阅 regular expression）
max_idle 参数，233
maximal_backoff_time 参数，73，233
maximal_queue_lifetime 参数，73，119
maxproc（master.cf），62，73
mbox 格式，91，95
虚拟邮箱文件，106

MDA（mail delivery agent），12
内容过滤，198
定义，21
垃圾邮件过滤，149
特殊用途，设定个别用户的 UBE rule，168
message ID（信息代码），75
message stores （邮箱），12
使用 LMTP 投递到非标准的……，97
格式，91-93
maildir，92，95
mbox，91，95
mbox 与 maildir 的比较，93
Postfix 与 POP/IMAP 对于……的认同，97
使用 POP 与 IMAP 取信，12
独立，使用……代管虚拟网域，110
shell access to（on Unix），13
message（邮件），E-mail（电子邮件）
以内容为辨别垃圾邮件的依据，148
由 Postfix 执行的投递操作，31-35
本地，33
其他 MDA，34
转发邮件，33
虚拟别名邮件，33
虚拟邮箱邮件，33
邮件与标题地址的格式（RFC 2822），22
……的格式，22
……如何进入 Postfix 系统，29-31
转寄，31
通知函，31
本地提交，29
网络邮件，30
分析垃圾邮件常用的词汇，146
排入 Postfix 队列的……，31
垃圾邮件，Postfix 能采取的动作，150-153
通过 Postfix 的流程，35-37
message_size_limit 参数，63，233
MIME 编码，23
转换字符串到……格式，183
mime_header_checks 参数，165，233
minimal_backoff_time 参数，73，235
minute（m），62
MLM（Mailing List Manager），128，133
Mailman，138-142
configure，指定 mailman 使用的 GID，138
创建列表，139
Python 需求，138
Majordomo，133-138
创建列表，135
mmencode 命令，183
modular design（模块化设计），Postfix，15
MTA（mail transfer agent），11
内容过滤，198
定义，21
邮件交换器，specified in MX 记录，82
message header 与……，22
rejecting or bouncing 邮件，21
routing E-mail with DNS MX 记录，82
SMTP 协议，使用，13
MUA（mail user agent），11
内容过滤，198
定义，21
邮件标头与……，22
POP/IMAP 与 SMTP，12
POP/IMAP server，从……取出邮件，13
SMTP 协议，用途，13
过滤垃圾邮件，149
mutual_auth（password 机制），182
MX 记录，82
备用交换器，118-121
DNS 检查规则，以……为准的客户端限制条件，162
没有……的网域，88
优先值，84
Postfix 对于……的查询，85
……的准则，85
mydestination 参数，33，54
由 local MDA 处理的网域，93
虚拟网域，增加，104
mydomain 参数，52，235
myhostname 参数，39，52
守护进程型过滤器与……，202
mynetwork 参数，54，235
设定守护进程型过滤器，204
出站邮件转发，127
mynetworks_style 参数，54

myorigin 参数，52，64，235
MySQL，48，208
编译，262
……的配置，208
MySQL 参数，209
MySQL/Postfix 配置的范例，210
Postfix 对于……的支持，检查，207
mysql-local.cf 文件，211

N

named pipe（命名管道 fifo 传输类型），61
name（传输服务名称）
fifo，61
inet，60
unix，61
nameservers for domain（域名服务器），82
namespace（命名空间），独立，虚拟网域名称，104
nested_header_checks 参数，165
network E-mail（网络邮件）
进入 Postfix系统，30
Postfix 的投递程序，13
network/netmask notation（网址表示法），54
network（网络）
在同……上的系统之间传递邮件，34
IP地址，列表，47
socket（inet传输类型），60
new／目录（maildir），93
newaliases 命令，39，49，131，136
newaliases_path 参数，235
newlist 命令（Mailman），140
NIS，49
noactive（password机制），181
noanonymous（password 机制），182
nobody account，95
nodictionary（password 机制），181
non_fqdn_reject_code 参数，160
noplaintext（password 机制），181
notifications of E-mail errors（错误通知函），31，74
notify_classes 参数，74，235
nslookup 工具，86
nsswitch.conf 文件，89

O

Official Release package（正式版），17
One-Time Password（参阅 OTP）
open relay（开放转发），54，145
anonymous 验证机制与……，175
DNS-based 黑名单，146
以 Postfix UBE rules避免……，152
openssl 命令，188
产生用户的公私钥组合，193
产生服务器的公私钥组合，190
签署证书，193
OpenSSL 函数库，188
openssl x509命令，194
operating system（操作系统），为特定……预编的 Postfix 包，16
OTP（单次密码验证机制），175
outbound mail relay（转发出站邮件），126
出站邮件，控制……使用的资源，73
owner_request_special 参数，130
邮件列表的拥有者，退信通知，132

P

PAM 作为 SASL验证架构，177
parameter（参数），219-247
内容检查，165
样本文件的说明，70
网域类型，87
LDAP，215
main.cf 与配置文件样本，41
MySQL，209
SASL 密码验证，180
严格语法，164
系统的主机名称与网域名称，52
参与 TLS联机的 SMTP client，195
参与 TLS联机的 SMTP server，191
parent_domain_matches_subdomains 参数，47，236
password 参数，对于 MySQL的设定，209
password（密码）
验证架构，选择，176
验证机制，选定，181
以 TLS 保护，173
SASL验证在存储与核验方面的应用，174

SASL，作为验证架构， 178
使用 Unix 系统密码作为 SASL 架构，177
pathname（路径名称），219
path（路径），宏，94
pattern matching （模式匹配，参阅 regular expression）
pcre（Perl-compatible regular expression） 47
PEM 格式的证书文件，188
performance（效率），Postfix 与……， 10
Perl
删除队列中特定邮件地址的邮件，……脚本，76
encode_sasl_plain.pl 脚本，184
Majordomo 的 approve 脚本，135
Majordomo，需求，133
Perl-compatible regular expression（pcre） 47
permit 限制条件，153，163
permit_auth_destination 参数，159
permit_mynetworks 参数，154，159
permit_sasl_authenticated 限制条件，180，181
permit_tls_clientcerts，195
persistent message storage（永久性邮箱），12
PGP，187
pickup daemon，29，35
唤醒，62
pickup_service_name 参数，236
pipe daemon，115
透过……传递邮件， 35
以……执行你自己的命令， 62
pipe MDA，37
代表收件列表的宏变量， 200
pipelining，reject_unauth_pipelining 过滤规则，160
pipes，named（fifo 传输服务），61
PKCS12 格式，客户端证书， 193
PLAIN 机制（验证），174，183
plaintext password （明文密码）
noplaintext 机制，181
搭配 saslauthd daemon，181
plug-in，auxprop，178
POP，12
pop-before-smtp，55
POP/IMAP，96
以 LMTP 协议处理邮件，34
本地投递与……，90
LMTP，97-101
maildir 邮箱格式，92
mbox 邮箱格式，92
邮箱格式，选择，93
非标准的邮箱格式， 97
邮箱访问与……，13
传递虚拟网域的邮件到……， 110
POP vs IMAP，96
Postfix，搭配……，96
Postfix 投递到虚拟账户与……， 103
独立的账户数据库，共享， 176
虚拟邮箱，设定，106
Portable Operating System Interface （POSIX） 47
port （通信端口）
目的地，对于 inet 传输，123
inet socket，60
公认…… SMTP（port 25） 24
POSIX 扩展正则表达式（参阅 regexp）
Post Office Protocol（参阅 POP、POP/IMAP）
postalias 命令，49，248
postcat 命令，248
-q 选项（显示队列内容），78
封装…… 的 shell 脚本，79
postconf 命令，39，248
-h myhostname，178
-l（locking），选项，92
-m（map），选项，45
-m 选项，检查 MySQL 与 LDAP 的支持能力，207
postdrop 命令，29，248
postdrop 组，39
Postfix
与 Sendmail 相兼容，10
角色定位，13
提供信息与源程序的网站， 16
postfix 命令，56，248
启动、停止、重新加载 Postfix，57
postfix 虚账户，39
postkick 命令，248
postlock 命令，248
postlog 命令，248

postmap 命令，45，249
以……检查 LDAP 配置，217
以……检查 MySQL 配置文件，212
对 clientcerts 文件执行……，194
对虚拟别名查询表执行……，105，115
-q 选项，测试查询表，48
以……测试规范映射表，65
以……测试模式表，168
postmaster，21
错误通知函的收件人，74
postqueue 命令，71，75，249
-f（flush）选项，80，124
-p 选项，以……列出所有邮件，75
-s（site）选项，80，121
postsuper 命令，71，75，249
-d ALL（删除队列里的所有邮件），76
-d（delete 选项，76
-h（hold）选项，77
-H 选项（移动邮件到正常队列），78
-r（requeue）选项，78
preference values for MX（邮件交换器的优先值），85
Postfix SMTP client，87
printf 命令，183
private 栏（master.cf），61
privilege levels for processes（进程的特权等级），15
unpriv 字段（master.cf），61
process（进程）
数量限制（maxproc，于 master.cf 中），62
限制可用的传输服务，73
Postfix，安全性与……，15
用于……的虚账户，19
shells 与……，15
Unix，标准 I/O，19
process_id_directory 参数，236
protocols for E-mail（邮件协议），12
IMAP，12
POP，12
POP/IMAP，邮箱访问与……，13
SMTP，12
邮件提交与……，13
proxy server，遭垃圾邮件发送者利用，146
proxy_interfaces 参数，236
pseudo-accounts（虚账户）
用于守护进程型过滤器，202
用于一般过滤程序，200
在 Unix 系统上的……，19
PTR 记录，82
DNS 检查规则，以……为客户端限制条件，162
反解成主机名称，87
public-key cryptography（公钥密码学），189
客户端证书，193
产生服务器端证书，190
Python，用于 Mailman，139

Q

qmgr daemon（参阅 queue management）
qmgr_clog_warn_time 参数，236
qmgr_message_active_limit 参数，236
qmgr_message_recipient_minimum 参数，237
qmqpd_error_delay 参数，237
queue ID，75
以……显示队列内容，79
queue management（队列管理），71-80
qmgr daemon 运作原理，71-75
损毁邮件，74
等待邮件，72
错误通知，74
信息传递，73
队列调度，72
用于……的工具，75-80
删除邮件，76
显示邮件，78
清空邮件，80
保留邮件，77
列出邮件，75
将邮件放回队列，78
queue manager（队列管理器），29
网络邮件，处理，30
（参阅 queue management）
queue manager，Postfix，31
queue scan（队列扫描）
……间隔时间的安排，73
决定……间隔时间，58
queue_directory 参数，71，237
chroot 位置，指定，61
新的根目录，69

queueing message, 28
queue_minfree 参数, 71
queue_run_delay 参数, 58. 73. 237
queue (队列)
  默认目录, 38
  incoming, active, deferred, hold 与 corrupt, 71
  (参阅 queue management)

R

rbl_reply_maps 参数, 237
RCPT TO命令 (SMTP), 25, 151
  检查……提供的地址, 157
  在……之后拒绝客户端, 153
realtime blacklists (实时黑名单), 146
  以……为准的限制条件, 156, 162
Received:标头字段, 23
receiving limit (收件限制), 63
  来自客户端的错误, 63
  单封邮件的收件地址限制, 63
  对于任何传输类型, 74, 114
receiving mail (接收邮件),DNS 与……, 87
receiving message (接收邮件) (Postfix 系统), 28, 29-31
  转寄, 31
  通知函, 31
  本地提交的邮件, 29
  网络邮件, 30
recipient address (收件地址), 151
  对于队列里的邮件, 75
recipient_canonical_maps参数, 65. 237
recipient (收件人)
  错误通知, 74
  本地投递, 列表, 94
  多个……, 于单一邮件中, 14
  经过转发的邮件, 119
  由 pipe daemon 展开的 … 列表, 200
recursive lookup (递归式查询), 46
redelivery attempts for deferred mail (重新投递延迟邮件), 72
regexp, 47
regular expression (正则表达式)
  内容检查参数, 模式表, 165
  用于查询表, 47
  用于访问表, 158
  用于内容检查的模式表, 165
  Perl-compatible (pcre). 47
  POSIX 扩展 regexp)
  以 postmap命令测试, 168
reject 限制条件, 153, 163
reject_code 参数, 239
rejected message (遭拒收的邮件), 21
  默认的通知期限, 153
  回复码, 151
  (参阅 response code)
rejecting mail (拒收邮件)
  在内容检查之后, 166
  在检查访问表之后, 157
  垃圾邮件
    选择 4xx 或 5xx 回复码, 164
    在 SMTP 对话过程中立刻……, 148
  寄给不明用户的邮件, 68
reject_invalid_hostname 参数, 154, 163, 164
reject_non_fqdn_hostname. 160
reject_non_fqdn_recipient, 160
reject_non_fqdn_sender, 160
reject_rbl_client, 163
reject_rhsbl_client, 163
reject_rhsbl_sender, 163
reject_sender_login_mismatch, 180
reject_unauth_destination 参数, 154, 159
reject_unauth_pipelining, 160
reject_unknown_client, 162
reject_unknown_hostname, 162
reject_unknown_recipient_domain, 162
reject_unknown_sender_domain, 156, 162
relay address (转发地址), 31
  寄到……邮件的投递程序, 33
relay control (转发控制), 54
  客户端证书验证, 56
  动态 IP 的解决方案, 55
  限制转发访问, 54
  SMTP验证, 55
relay delivery transport (relay MDA) . 90
relay domain (转发网域), 87
relay_domains 参数 ,34, 119
  架设内部网络的邮件网关, 125
  清空目的地为 ……的邮件, 80
relay_domains_reject_code参数, 239

relayhost参数，127
relaying mail（转发邮件），118-127
备用交换器，118-121
快速清空，120
转发收件人列表，119
……客户端验证，173
入站邮件网关，125
开放转发，遭垃圾邮件发送者利用，145
出站，126
转发收件人，119
垃圾邮件发送者的手段，143
遭垃圾邮件发送者入侵并用于……，146
传输表，121-125
格式，121
延后投递，124
relay_recipient_maps参数，119
列出合法收件地址，126
relay_transport参数，239
reloading Postfix
使main.cf的改变生效，44
postfix reload命令，使用，57
relocated user（迁出的用户），改写……的地址，68
relocated_maps参数，46，68，239
remote user（远程用户）
……的IP地址，55
SASL验证，173
reply code（回复码，参阅response code）
repository for suspected spam（暂存可疑的垃圾邮件），149
Request for Comment（参阅RFC）
requeuing mail（重新排列邮件），78
resolv.conf文件，89
resolve_dequoted_address参数，239
resource error（资源错误），74
资源记录，82
用于查询的命令行工具，86
response code（响应码）
硬响应与软响应，154
Postfix用来拒收邮件的……，151
拒绝不明客户端的请求，162
以4xx或5xx拒收垃圾邮件，164
因不明客户端的主机名称而拒绝请求，162
因客户端的主机名称不完整而拒绝请求，160
因收件地址不完整而拒绝请求，160
因收件网域不明而拒绝请求，162
因寄件方网域不明而拒绝请求，162
SMTP，25，86
restriction classes（限制条件分类），150，168
范例，170
正常限制条件的例外，168
restriction（限制条件）
设定给客户端辨识的规则，150
访问表，156-159
……的定义，156
DNS限制条件，160
限制条件范例，154
一般限制条件，163
原理，153
列表，152
其他，159
实时黑名单，162
SMTP检查规则与限制条件，150，152
严格语法，160
测试新的……，154
permit_sasl_authenticated，180
Postfix反垃圾邮件范例，171
reject_sender_login_mismatch，180
retrieving mail from message store（访问邮箱中的邮件，参阅POP/IMAP）
reverse lookup of IP address to hostname（从IP地址反查出主机名称），82，87
RFC 2142（邮箱名称规定），51
RFC 2554, "SMTP Service Extension for Authentication"，173
RFC 2821（SMTP协议），23
RFC 2822（邮件标题、正文、地址的格式），22
RFC 3207, STARTTLS，187
RFC 822（邮件信息格式），22
RFC 882（定义DNS服务），81
RFC（Request for Comment），20
root账户，19
#提示符，19
别名与所有者身份，51

需要 root权限的服务，61
指向……的系统别名，51
routing E-mail(邮件路由)，82-85
DNS BIND配置文件，83
DNS MX记录，83
规则，85
外来邮件，31
MX 记录，82
run time error（运行时错误），264

S

sample configuration file （配置文件样本），41
sample_directory 参数，70，239
SASL 验证，55，173-186
选择验证架构，176
选择验证机制，174
供 SMTP server用于验证客户端身份，173
对 Postfix 的设定步骤，176-182
设定步骤汇整，182
启动 SASL，180
用于SASL验证的参数，180
核准授权用户，181
避免寄件人假冒，180
指定验证架构，177-178
指定密码机制，181
概论，174
Postfix，搭配使用，176
SASLv2，176
测试验证配置，182-184
SMTP 客户端验证，185
saslauthd daemon，177
-a 选项，177
sasldb auxiliary property plug-in（辅助性专属外挂模块），178
saslpasswd2命令，178
search order （比对顺序），查询表，46
second （秒，s），62
security(安全性)，10，14-16
服务器进程的 chroot环境，69
抵御攻击的设计因素，15
引入文件与……，51
Postfix 的模块化结构，15
shell与 process，15
Simple Authentication and Security Layer（参阅 SASL 验证）
Transport Layer Security （参阅 TLS）
select_field 参数，209，211，212
sender（寄件人）
在队列中的邮件的……，75
sender address（寄件人地址）
以 MAIL FROM命令表示（参阅 MAIL FROM command）
伪装，避免，180
sender_canonical_maps 参数，65
sending mail（寄件），DNS 与……，85
Postfix 配置选项，86
PTR 记录，87
Sendmail，9
别名文件，兼容格式，49
Postfix对于……的兼容性，10
-q 选项，指定队列扫描的间隔时间，58
成为特权进程的安全问题，15
sendmail 命令，29，35，58，199
sendmail_path 参数，240
独立 网域，103
搭配系统账户，104
搭配虚拟账户，105，110
service name（服务名称，于 master.cf 配置文件），60
session key（TLS），189
setgid_group 参数，240
共享网域，103
showq_service_name 参数，240
signed certificate（数字签名证书），189
自己扮演 CA角色，190
转换成 MUA 能接受的格式，194
服务器，192
Simple Authentication and Security Layer （参阅 SASL验证）
Simple Mail Transport Protocol（参阅 SMTP）
site（站点）
适合快速清空的……，120
清空寄送到特定……的邮件，80
size(大小)
限制收件的……，63
在队列中的邮件，75
S/Key（参阅 OTP 验证机制）

S/MIME，187
SMTP，9，12
客户端验证，185
以 SASL 协议进行客户端验证，173
在……对话过程判断客户端是否为垃圾邮件源，152
对话 client-detection过滤规则，150
延迟所有投递操作，125
邮件软件包，12
提交，13
加强，27
信封地址，指定…… 22
ETRN 命令，120
对话过程的日志记录，151
投递到本机之外的其他系统，35
概论，23-27
Postfix/TLS 配置，191
在……对话过程中立即拒绝垃圾邮件，148
Postfix对于……响应码的应对， 86
响应码，定义的列表， 25
以 telnet 测试 SASL验证过程，182-184
STARTTLS extension，187
严格语法
检查限制条件，client-based 过滤规则，156
参数，164
TLS client 设定， 195
SMTP验证，55
对于使用动态 IP地址的客户端，55
smtp客户端服务，创建特殊的……， 63
smtp delivery agent（smtp MDA） 33，34
smtp 传输服务，91
smtp_bind_address 参数， 240
smtp_connect_timeout 参数，63
smtpd daemon，30，35
对于收件的限制，强加于……， 63
smtp_data_done_timeout参数，240
smtp_data_xfer_timeout 参数，240
smtpd_banner参数， 242
smtpd_client_restrictions
守护进程型过滤器与……， 204
smtpd_client_restrictions参数，150
smtpd.conf 文件，177
saslauthd，供 Postfix 用于验证，177
smtpd_data_restrictions 参数，150，242
smtpd_delay_reject参数，153
smtpd_error_sleep_time参数，63，242
smtp_destination_concurrency_limit参数，74
smtpd_expansion_filter 参数，242
smtpd_hard_error_limit 参数，63
smtpd_helo_required参数，164，243
smtpd_helo_restrictions，150，151 163
守护进程型过滤器与……，204
smtpd_history_flush_threshold参数，243
smtpd_noop_commands 参数，243
smtpd_recipient_limit参数，243
smtpd_recipient_restrictions参数，150，154，181
增添 permit_tls_clientcerts到过滤规则组合，195
设定守护进程型过滤器， 204
smtpd_restriction_classes参数，243
smtpd_sasl_auth_enable 参数，180，184
smtpd_sasl_security_options参数，181
smtpd_sender_login_maps 参数，180
smtpd_sender_restrictions，150，164
守护进程型过滤器与……，204
smtpd_soft_error_limit 参数，63 243
smtpd_tls_ask_ccert 参数，195
smtpd_tls_CAfile 参数，192
smtpd_tls_CApath参数，192
smtpd_tls_cert_file 参数，192
smtpd_tls_key_file 参数，192
smtpd_use_tls 参数，192
smtp_helo_timeout 参数，241
smtp_mail_timeout 参数，241
smtp_pix_workaround_delay_time 参数，241
smtp_quit_timeout 参数，241
smtp_randomize_addresses 参数，87
smtp_rcpt_timeout参数，242
smtp_sasl_auth_enable 参数，185
smtp_sasl_password_maps 参数，185
smtp_sasl_security_options参数，185
smtp_skip_5xx_greeting 参数，242
smtp_tls_CAfile 参数， 197
smtp_tls_cert_file 参数，195
smtp_tls_key_file 参数，195
smtp_use_tls 参数，195

socket
LMTP server, 110
SMTP client联机到 Postfix, 150
Unix-domain或 TCP，用于 LMTP投递操作，98
socket_type 参数，99
soft_bounce参数，244
测试新的限制条件，154
software errors （软件错误），74
spam （垃圾邮件），143-172
反垃圾邮件行为，148
以 relay control 阻挡……，54
设定 Postfix 去检查……，149-172
判别条件的分类，149
以客户端身份为准的判别条件，150-164
内容检查，165-168
自定义的过滤规则组合，168
严格语法参数，164
检测依据，145-148
客户端，146
内容，148
误判合法邮件为垃圾邮件，145
……的主要方法，146
垃圾邮件的手段，148
诚信问题，143
不完整地址，用途，65
luser_relay 参数与……，68
open realy（开放转发），145
Postfix的反垃圾邮件范例，171
……引起的问题，144
special character （特殊字符），于别名定义中，50
spoofing sender address （假冒寄件人地址），避免，180
SQL statement created by Postfix （由 Postfix 创建的 SQL语句），209
SSL （Secure Sockets Layer） （参阅 TLS）
standard error （stderr），19
standard input （stdin），19
standard output （stdout），19
starting Postfix （启动 Postfix），第一次，39
57
别名文件，系统，39
系统的主机名称，39
STARTTLS命令，189
status code （状态码），SMTP服务器回复的……，86
stopping Postfix （postfix stop 命令），57
storage of E-mail message （邮箱），12
strace，70
strict syntax （严格语法）
参数，164
strict syntax checking （严格语法检查）
以……判别客户端是否为垃圾邮件源的条件，156，160
参数，149
strict 7bit headers参数，244
strict_8bitmime_body 参数，244
strict_rfc821_envelopes 参数，165，244
strong authentication （强效验证），187
subdomain （子网域），以网域名称隐藏……，66
subnet （子网络），以 network/netmask格式表示，54
superuser（参阅 root account）
swap_bangpath 参数，244
symbolic names for port （通信端口的符号名称），61
symlink （符号链接）
chroot 与……，70
对于初始脚本，59
syntax （语法），严格遵守标准（参阅 strict syntax checking）
syslog daemon，56
syslog_name 参数，244
system account （系统账户），103
搭配独立的虚拟网域，104
搭配共享网域，104
system alias （系统别名），51
system logfile （系统日志文件），56
system startup （开机）
在……时启动 Postfix，58-59
写出一个初始脚本，58
在……自动启动 saslauthd，177

T

table 参数 （MySQL），209
targets for alias （别名的目标），50
别名文件的限制，51

TCP socket，LMTP 使用的……，98
telnet，以……测试 SASL验证，182-184
text editor（文本编辑器），编辑 main.cf 配置文件，41
time limits on deferred mail redelivery attempt（对于重新投递延迟邮件的时限），73
time units（计时单位），62，219
TLS（Transport Layer Security），56，173，187-197
  证书，188
    自己开设 CA，188
    客户端，193-195
    产生服务器证书，190
    安装 CA 根证书，191
    公钥密码学，用途，189
  编译，260
  设定 TLS-SMTP client，195
  Postfix 与……，188
  Postfix/TLS 配置，191
    summary of，192
tmp/ 目录（maildir），93
To: address in E-mail message header（于邮件标头中的 To:地址），22
tracing message through Postfix（邮件通过 Postfix 的流程），35-37
tracing service failures in chroot environment（在 chroot环境下追查服务错误原因），70
Transport Layer Security（参阅 TLS）
transport maps（传输表），121-125
  记录，121
    host，121
    port，123
    右手边值……的格式，123
    传输，121
  延迟传递时间，124
    延迟投递，125
    延迟转发，124
transport type（传输类型）
  inet unix 与 fifo，60
  ……于 master.cf 配置文件中的字段，121
  有效的……服务名称，61
transport_destination_recipient_limit 参数，74，114
transport_maps参数，115
  搭配 LDAP 的设定，217
  查询表，比对顺序，46
  指向传输表，126
transport_retry_time 参数，246
transport（传输）
  邮件投递……的配置，73
  Postfix 的投递操作，90
trivial-rewrite daemon，29，36
  路由信息，queue manager 的抉择，31

U

UBE（Unsolicited Bulk E-mail）（垃圾邮件，参阅 spam）
UID（user id），Unix，18
undeliverable mail（无法投递的邮件），31
undisclosed_recipients_header 参数，246
Unix，18
  命令提示符，19
  hostname命令，39
  ……的初始脚本，58
  登录名称与 UID，18
  过长行，以\接续，20
  man page，20
  Postfix 与……，4
  虚账户，19
  shell process，Postfix 与……，15
  标准输入／输出，19
  superuser（root）账户，19
  以系统密码作为 SASL验证架构，177
unix（传输类型），60
Unix-domain socket，LMTP，98
unknown_address_reject_code 参数，162
unknown_client_reject_code参数，162，246
unknown_hostname_reject_code 参数，162
unknown_local_recipient_reject_code 参数，246
unknown_virtual_alias_reject_code 参数，246
Unsolicited Bulk E-mail（UBE）（参阅 spam）
upgrading Postfix（升级 Postfix），259
user account（用户账户）
  SASL 为 SMTP server创建……，178

独立网域搭配虚拟账户
邮箱文件所有权， 106
系统……
搭配独立虚拟网域，104
搭配共享网域，104
系统……与虚拟……， 103
虚拟，搭配独立网域， 105-110
user参数，关于 MySQL 的设定，209
user （用户）
.forward 文件，local MDA对 …的检查，33
NIS 数据库，49
密码（参阅 password）
迁出的……，改写……的地址， 68
垃圾邮件，标示，149
不明，68
uuencoding，23

V

variable expansion（变量扩展），200
以……指定路径，94
variable（变量）
配置，42
Venema，Wietse，1-2，9
verbose logging information（详细的日志信息），62
verp_delimiter_filter参数， 246
Vexira AntiVirus for mail server，204
虚拟账户，103
搭配独立网域，105-110
SMTP user 的外部密码数据库，176
virtual alias address（虚拟别名地址），31，33
virtual alias（虚拟别名），87
无限别名地址，109
virtual delivery agent（virtual MDA），33，91，106
virtual domain（虚拟网域），87，103
DNS 配置，104
MySQL 配置，212
Postfix 对于……邮件的处理，103
virtual mailbox address（虚拟邮箱地址），31
virtual mailbox catchall address（虚拟邮箱无限别名地址），109
virtual mailbox domain（虚拟邮箱网域），virtual MDA，91
virtual mailbox（虚拟邮箱），87
virtual mailing list manager（虚拟网域的MLM），架设，116
virtual_alias_domains 参数，33，104，109
virtual_alias_maps 参数，33，46，105，107，115，247
virtual_gid_maps 参数， 107
virtual_mailbox_base 参数，106，247
virtual_mailbox_domains 参数，33，105，109
列出可收信的虚拟网域名称， 110
virtual_mailbox_limit 参数，247
virtual_mailbox_maps 参数，33，106，247
指向含有有效地址的查询表， 110
virtual_transport 参数，110，247
virtual_uid_maps 参数， 107
virus（病毒）
反病毒过滤器，198，199
以 header_checks扫描……，167
Vexira AntiVirus 软件，205

W

wakeup（master.cf），62
warn_if_reject 参数，154
warning message after content-checking （在内容检查之后的警告信息），166
web site（网站）,Postfix在线说明，70
week（w），62
well-known port（公认通信端口，SMTP 的25端口），24
where_field 参数（MySQL），209，211
whitelist applications （白名单）,预先核准，145
WHOSON，55