Perl for System Administration

David N. Blank-Edelman

O'REILLY®

Perl для системного администрирования

Дэвид Н. Бланк-Эдельман

Санкт-Петербург
2001

Дэвид Н. Бланк-Эдельман

Perl для системного администрирования

Перевод Т. Морозовой

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Научный редактор	*В. Коновалов*
Редакторы	*Л. Ага, В. Овчинников*
Корректура	*С. Беляева, С. Журавина*
Верстка	*Н. Гриценко*

Дэвид Н. Бланк-Эдельман
Perl для системного администрирования. – Пер. с англ. – СПб: Символ-Плюс, 2001. – 496 с., ил.
ISBN 5-93286-024-3

Эта книга будет полезна администраторам с различным уровнем опыта – от обычных пользователей Linux до администраторов крупных систем. Автор рассматривает основные платформы, включая Unix, Windows NT/2000 и MacOS. При наличии некоторого опыта программирования на Perl вы узнаете, как при помощи этого языка повысить производительность во многих областях, включая: управление учетными записями пользователей; наблюдение за файловой системой и отслеживание процессов; работу с сетевыми службами имен NIS и DNS; администрирование баз данных при помощи DBI и ODBC; работу со службами каталогов LDAP и ADSI; обработку и анализ файлов журналов регистрации; поддержку защищенной сети; наблюдение за удаленными устройствами средствами SNMP.

Автор – опытный системный администратор, работающий в многоплатформенном окружении, что предоставляет вам хорошую возможность поучиться на чужом опыте. Вы узнаете о возможных ловушках и способах их обойти при помощи Perl. Включенные в книгу примеры и сценарии можно использовать для решения рутинных повседневных задач.

ISBN 5-93286-024-3
ISBN 0-56592-609-9 (англ)

Издательство «Символ-Плюс». 193148, Санкт-Петербург, ул. Пинегина, 4,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 06.09.2001. Формат 70x100 1/16 . Бумага офсетная.
Печать офсетная. Объем 31 печ. л. Тираж 3000 экз. Заказ N
Отпечатано с диапозитивов в ФГУП «Печатный Двор» им. А. М. Горького
Министерства РФ по делам печати, телерадиовещания
и средств массовых коммуникаций.
197110, Санкт-Петербург, Чкаловский пр., 15.

Оглавление

Об авторе

Дэвид Н. Бланк-Эдельман – директор по технологиям факультета вычислительной техники Северо-восточного Университета (Northeastern University College of Computer Science). Последние 14 лет он работал системным и сетевым администратором на больших многоплатформенных системах в Brandies University, в технологической группе Кэмбриджа (Cambridge Technology Group) и лаборатории MIT. Кроме того, он является главным техническим редактором журнала «The Perl Journal» и написал немало статей о музыке разных стран. В свободное время он учится играть на мбире – традиционном музыкальном инструменте Зимбабве.

Предисловие

Perl – это мощный язык программирования, уходящий корнями в задачи традиционного системного администрирования. В течение многих лет он адаптировался и расширялся для работы с новыми операционными системами и новыми задачами. До сих пор, однако, не было ни одной книги, посвященной использованию Perl исключительно для системного администрирования, тем самым признавая факт глубокой исторической связи.

Если вы уже немного знакомы с Perl и вам необходимо выполнять задачи системного администрирования, то эта книга для вас. Читатели с различным уровнем опыта как в Perl, так и в системном администрировании, найдут в этой книге что-то для себя полезное.

Структура книги

Каждая глава этой книги посвящена определенному кругу задач системного администрирования. В конце каждой главы приведен список используемых в ней модулей и ссылки на более подробную информацию по рассмотренной теме. В книгу включены следующие главы:

Глава 1 «Введение»

Во введении описан материал, детально рассмотренный далее в книге, рассказано, как он может послужить и что нужно для того, чтобы получить от него максимальную пользу. В этой книге рассмотрен серьезный материал и считается, что он будет использован «влиятельными» пользователями (например, привилегированными пользователями в Unix или администраторами Windows NT/2000). Кроме того, введение содержит важные советы, помогающие писать более безопасные программы на Perl.

Глава 2 «Файловые системы»

Эта глава о том, как содержать в порядке и правильно использовать различные многоплатформенные файловые системы. Мы начнем с рассмотрения принципиальных различий между файловыми системами каждой из операционных систем. Затем мы рассмотрим процесс исследования файловой системы средствами Perl и расскажем, когда это может оказаться полезным. И, наконец, мы узнаем, как работать с дисковыми квотами из Perl.

Глава 3 «Учетные записи пользователей»

В этой главе рассказано о том, что собой представляют учетные записи на двух различных операционных системах. Основная часть главы – простейшая система ведения учетных записей, написанная на Perl. В процессе построения этой системы мы рассмотрим механизмы, необходимые для записи учетных записей в простую базу данных, основанную на XML, а также механизмы для создания учетных записей и их удаления.

Глава 4 «Действия пользователей»

В главе 4 рассмотрены различные механизмы управления процессами для всех трех операционных систем от самых простых (например, процессов в MacOS) до более сложных (в WinNT/2000). Мы организуем работу этих механизмов с помощью вспомогательных сценариев. И, наконец, мы узнаем, как средствами Perl проследить за операциями с файлами и за сетью.

Глава 5 «Службы имен TCP/IP»

Службы имен позволяют узлам в сети TCP/IP общаться друг с другом. В этой главе отражена вся история развития, начиная с файлов узлов, продолжая сетевой информационной службой (NIS) и заканчивая связующим звеном в Интернете – доменной службой имен (DNS). Для каждого шага такого пути мы покажем, как можно упростить профессиональную работу с этими службами при помощи Perl.

Глава 6 «Службы каталогов»

По мере роста сложности рассматриваемой информации растет и важность служб каталогов, которые мы используем для доступа к этой информации. Хорошо, если системные администраторы будут не просто использовать эти службы, но и создавать собственные инструменты для работы с ними. В этой главе рассказано о некоторых из наиболее популярных служб, таких как LDAP и ADSI, а также показано, как с ними работать при помощи Perl.

Глава 7 «Администрирование баз данных SQL»

Очень часто в сферу действий системного администратора попадает и работа с реляционными базами данных. А значит, системные администраторы должны быть знакомы с администрированием баз данных SQL. В этой главе рассмотрены два механизма для работы с базами данных – DBI и ODBC, а также приведены примеры их использования.

Глава 8 «Электронная почта»

В этой главе показано, как с помощью Perl лучше использовать электронную почту в качестве инструмента системного администрирования. После обсуждения основ отправки и разбора электронной почты с помощью Perl мы рассмотрим несколько интересных

приложений, включая анализатор спама и средство обработки электронной почты в службу технической поддержки.

Глава 9 «Журналы»

Системные администраторы зачастую просто утопают в море файлов журналов. Каждая машина, операционная система и программа может вести (и часто это делает) журналы. В этой главе рассмотрены системы ведения журналов в Unix и в WinNT/2000. Мы обсудим подходы к анализу такой информации с целью заставить ее работать на вас.

Глава 10 «Безопасность и наблюдение за сетью»

В последней главе мы обсудим вопросы, связанные с безопасностью. Мы покажем, как Perl может помочь сделать сеть и отдельные узлы в ней более защищенными. Кроме того, мы расскажем о некоторых технологиях контроля, включая использование протокола SNMP (простой протокол управления сетью) и «прослушивание» сети.

Приложения

В некоторых главах предполагается, что у вас уже есть знания по определенным темам, в то время как этого может и не быть. Для тех, кто не знаком с отдельными темами этой книги, есть несколько мини-руководств, которые помогут быстро разобраться в их основах. В число этих руководств входят введение в систему контроля версий (RCS), введение в протокол LDAP (облегченный протокол доступа к каталогам), введение в SQL, XML и протокол SNMP.

Условные обозначения

Курсив

Предназначен для выделения имен файлов, пользователей, каталогов, команд, узлов и адресов (URL), а также терминов, встречающихся в первый раз.

`Моноширинный шрифт`

Используется для выделения имен модулей и функций Perl, а также примеров исходного кода и результатов выполнения команды.

`Моноширинный полужирный шрифт`

Применяется для выделения пользовательского ввода в примерах.

`Моноширинный полужирный курсив`

Служит для выделения части командной строки, которая может быть изменена пользователем, а также для примечаний к программному коду.

Как с нами связаться

Мы тестировали и проверяли всю информацию, приведенную в книге, но вы можете обнаружить, что некоторые возможности изменились (или мы допустили кое-какие ошибки). Пожалуйста, сообщайте нам об ошибках, которые вы найдете, а также присылайте предложения относительно будущих изданий по следующему адресу:

O'Reilly & Associates, Inc.
101 Morris Street
Sebastopol, CA 95472
(800) 998-9938 (в США и Канаде)
(707) 829-0515 (международный/местный)
(707) 829-0104 (факс)

Кроме того, вы можете прислать нам письмо электронной почтой. Чтобы подписаться на наш список рассылки или запросить каталог, пошлите письмо по адресу:

info@oreilly.com

Чтобы задать технические вопросы или прислать комментарий к книге, пишите по адресу:

bookquestions@oreilly.com

Кроме того, у нас есть веб-сайт, посвященный специально этой книге. Там приведены примеры, список найденных ошибок и рассказано о плане следующих изданий. Эта информация доступна по адресу:

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

Подробную информацию об этой и других книгах вы найдете на вебсайте издательства O'Reilly:

http://www.oreilly.com

Благодарности

Работа над книгой оказалась жребием, очень напоминающим строительство классической арки. Она также начиналась с двух колонн в моей жизни, которые сошлись вместе, – технической и личной.

С технической «точки зрения» я глубоко признателен Ларри Уоллу, который не только создал Perl, но и вдохнул в него (да и во все сообщество Perl-программистов) особый дух. Я благодарен великим учителям – Тому Кристиансену и Рэндаллу Шварцу, которые помогли мне и бесчисленному количеству людей разобраться со всеми тонкостями языка. Выше по этой колонне стоят программисты, которые потратили уйму времени и энергии на работу с языком, а затем решили поделиться со мной и со всем остальным сообществом Perl-программистов результатами своей работы. При любой возможности я стараюсь упомя-

нуть этих людей в книге, но моя благодарность относится ко всем названным и не названным, кто обогатил культуру Perl своими усилиями.

Поднимаясь вверх по технической колонне, за разделом о Perl мы обнаруживаем раздел системного администрирования. Здесь мы находим еще одно сообщество людей, которые помогли мне, этой книге и всей компьютерной области в целом. Члены конференций Usenix, SAGE и LISA заслужили благодарность за развитие лучшего, что может предложить область системного администрирования. В особенности мне хочется сказать спасибо Реми Эварду за то влияние, которое он оказал на мое профессиональное и личное понимание этой области, являясь моим другом, учителем и примером для подражания. Именно он – тот системный администратор, каким я хотел бы стать.

Ближе к вершине профессиональной колонны находятся те, кто ответствен за создание этой книги. Для начала я хочу поблагодарить моих редакторов и других комментаторов, потративших бездну часов на изменение этого текста (перечисляю всех в алфавитном порядке): Джо Джонстона, Джерри Картера, Тома Лимонселли, Джона Монтгомери, Хриса Нандора, Майкла Пепплера, Майкла Стока, Натана Торкингтона, Элин Фриш и Тоби Эверетта. Вплоть до самого конца этой работы они продолжали учить меня тонкостям Perl. Я благодарен Рону Портеру за его иллюстрации, Ханне Дайер и Лори Леджун за прелестное животное на обложке и всем сотрудникам издательства O'Reilly, принявшим участие в подготовке книги. И, наконец, я едва достоин права благодарить Линду Муи – моего редактора, чье удивительное мастерство, точность и забота позволили мне «родить» эту книгу и вырастить ее в хорошем доме. Линда Муи просто восхитительна.

Так же как и арка не строится из одной колонны, так и эта книга появилась не без участия другой, более личной основы. Я должен поблагодарить всех людей из моего духовного сообщества «Хавурат Шалом» из Соммервилля, Массачусетс, за их постоянную поддержку в течение всего процесса. Они научили меня смыслу сообщества. Спасибо вам, *Мекор ХаШаим*, за эту книгу и благословение в жизни.

С духовной точки зрения я в долгу перед народом Шона из Зимбабве за их удивительную музыку *мбира*, которая позволяла мне сохранять здравомыслие все это время. В особенности спасибо тем, кто познакомил меня с этой музыкой, – либо учителей, либо учеников, игравших рядом со мной. Эрика Эйзим, Стюарт Кардунер, Тьют Чигамба, Вири Чигонга, Мусекива Чингодза, Форвард Квенда, Космас Магая, Наоми Моланд, Соломон Мурунгу, Пол Новитски и Нина Рубин сыграли свою роль в этом процессе.

Я благодарен моим друзьям Эвнеру, Элен и Филу Шапиро и Алексу Сковронеку за их одобрение. Особое спасибо Джону Орванту и Джоелу Сегелу, двум друзьям, чьи мудрые советы и поддержка предоставили мне возможность и смелость для борьбы. Большое спасибо факультету и сотрудникам Северо-восточного университета компьютерных наук.

Я особенно благодарен членам группы CCS Systems, которые предоставили мне пространство, время и терпение, необходимые для работы с этой книгой. Ларри Финкельштейн – декан колледжа компьютерных наук – также заслуживает особого упоминания. Я никогда не встречал человека вне системного администрирования, который лучше бы понимал системных администраторов, их нужды и всю эту область в целом. Он продолжает учить меня, особенно на примерах, что означает быть настоящим лидером.

Давайте вернемся к метафоре арки, поскольку мы уже почти на вершине. Здесь мы найдем всю мою семью. Я благодарен всем им. Основа моей семьи – Майра, Джейсон и Стивен Бланк – это те люди, чей характер и воспитание (и любовь) в течение многих лет позволили мне сейчас быть с вами. Мои благодарности – Шиммеру и Бендиру, нередко составлявшим мне приятную компанию в написании этой книги, засиживаясь за компьютером до раннего утра. Большое спасибо Кристену Портеру и Тому Доновану из TCM.

Если вы знаете, что такое арки, то, вероятно, заметили, что я упустил самое главное – ключевой камень. Это камень, который находится на самой вершине арки и удерживает всю конструкцию вместе. Синди Бланк-Эдельман была моим ключевым камнем во время работы над этой книгой. Если и есть кто-то, кто пожертвовал ради этой книги большим, чем я сам, то это именно она. Без ее любви, поддержки, заботы, юмора, руководства и вдохновения я не остался бы самим собой, не говоря уж о писании каких-либо книг.

Эта книга посвящается Синди – любви всей моей жизни.

6

- *Что такое каталог?*
- *Finger: простая служба каталогов*
- *Служба каталогов WHOIS*
- *LDAP: сложная служба каталогов*
- *ADSI (Интерфейсы служб активных каталогов)*
- *Информация о модулях из этой главы*
- *Рекомендуемая дополнительная литература*

Службы каталогов

Чем больше информационная система, тем сложнее найти в ней что-либо конкретное или выяснить, что именно в ней доступно. Когда сети разрастаются и увеличивается их сложность, обслуживать их удобно при помощи тех или иных каталогов. Пользователи в сетях могут применять службы каталогов для поиска других пользователей, для почтовых служб и иных служб сообщений. Такие сетевые ресурсы, как принтеры или доступные по сети дисковые пространства, могут быть объявлены при помощи службы каталогов. Средствами служб каталогов также можно распространять открытые ключи и сертификаты. В этой главе мы рассмотрим, как использовать Perl для взаимодействия с некоторыми из наиболее популярных служб каталогов, включая Finger, WHOIS, LDAP и ADSI.

Что такое каталог?

В главе 7 «Администрирование баз данных SQL» мною сделано предположение, согласно которому мир системного администрирования представляет собой базу данных. Каталоги – хороший пример такой оценки. Отметим некоторые явные характеристики каталогов, чтобы в дальнейшем разговоре различать «базы данных« и «каталоги»:

Использование сети

Каталоги практически всегда связаны сетью. В отличие от некоторых баз данных, расположенных на той же машине, что и их клиенты (как хорошо известный файл */etc/passwd*), службы каталогов обычно предоставляются по сетям.

Простое взаимодействие/манипуляция данными

Базы данных зачастую имеют сложные языки запросов для получения и обработки данных. Самый распространенный из них – SQL, ему уделено внимание в главе 7 и приложении D «Пятнадцатиминутное руководство по SQL». Взаимодействие с каталогами значительно проще. Клиент каталогов обычно выполняет только элементарные операции и не использует для работы с сервером никакого специального языка.

Иерархичность

Современные службы каталогов поддерживают древоподобные информационные структуры, в то время как базы данных в целом – нет.

Читаем много, пишем мало

Современные службы каталогов оптимизированы под очень специфичную передачу данных. При обычном использовании количество операций чтения/запросов к службе каталогов во много раз превосходит количество операций записи/обновлений.

Если вам встретится нечто, похожее на базу данных, но обладающее приведенными выше характеристиками, то, скорее всего, вы имеете дело с каталогом. Во всех четырех службах каталогов, которые мы рассмотрим, эти характеристики несложно заметить.

Finger: простая служба каталогов

Finger и WHOIS – отличные примеры простых служб каталогов. Finger, в особенности, предоставляет доступную только для чтения информацию о пользователях на машине (впрочем, скоро мы увидим более творческий подход к его применению). Более поздние версии Finger, такие как сервер GNU Finger и его производные, имеют расширенную разновидность этой функциональности, они позволяют обращаться к одной машине и получать информацию от всех машин из вашей сети.

Finger был одной из первых широко используемых служб каталогов. Когда-то очень давно при необходимости выяснить адрес электронной почты пользователя на другом узле или даже на вашем собственном лучшим решением было применение команды *finger*. Команда *finger harry@hogwarts.edu* сообщала адрес электронной почты Гарри, будь он `harry`, `hpotter` или даже что-то менее очевидное (правда, эта команда выводила список всех остальных учащихся школы с именем Гарри). И хотя *finger* применяется до сих пор, популярность команды со временем уменьшилась, т. к. вошли в обиход домашние страницы и свободное получение информации о пользователе стало делом проблематичным.

Использование протокола Finger из Perl – еще один хороший пример правила TMTOWTDI. Когда я первый раз искал на CPAN хоть что-то, выполняющее операции с Finger, мне не удалось найти ни одного такого модуля. Сейчас-то там можно натолкнуться на модуль `Net::Finger` Дениса Тейлора (Dennis Taylor), который появился спустя примерно шесть месяцев после моего первого посещения. Скоро вы с ним познакомитесь, а пока предположим, что его не существует, и не упустим случая выяснить, как применять более общий модуль, позволяющий «связываться» по специфическому протоколу.

Протокол Finger – это очень простой текстовый протокол на базе TCP/IP. В соответствии с определением в RFC1288, он ожидает стандартное TCP-соединение на порту 79. После установки соединения клиент передает простую строку, завершающуюся последовательностью CRLF.[1] В этой строке запрашивается информация либо о конкретном пользователе, либо обо всех пользователях на машине, если строка пустая. Сервер отвечает на запрос и закрывает соединение после передачи потока данных. Можно увидеть, как это происходит, если подсоединиться к порту Finger на удаленной машине напрямую при помощи *telnet*:

```
$ telnet kantine.diku.dk 79
Trying 192.38.109.142 ...
Connected to kantine.diku.dk.
Escape character is '^]'.
cola<CR><LF>
Login: cola                             Name: RHS Linux User
Directory: /home/cola                   Shell: /bin/noshell
Never logged in.
No mail.
Plan:

Current state of the coke machine at DIKU
This file is updated every 5 seconds
At the moment, it's necessary to use correct change.
This has been the case the last 19 hours and 17 minutes

Column 1 is currently *empty*.
   It's been 14 hours and 59 minutes since it became empty.
   31 items were sold from this column before it became empty.
Column 2 contains some cokes.
   It's been 2 days, 17 hours, and 43 minutes since it was filled.
   Meanwhile, 30 items have been sold from this column.
Column 3 contains some cokes.
   It's been 2 days, 17 hours, and 41 minutes since it was filled.
   Meanwhile, 11 items have been sold from this column.
Column 4 contains some cokes.
   It's been 5 days, 15 hours, and 28 minutes since it was filled.
```

[1] Возврат каретки + перевод строки, т. е. символы с ASCII-кодами 13 и10.

```
    Meanwhile, 26 items have been sold from this column.
Column 5 contains some cokes.
    It's been 5 days, 15 hours, and 29 minutes since it was filled.
    Meanwhile, 18 items have been sold from this column.
Column 6 contains some coke-lights.
    It's been 5 days, 15 hours, and 30 minutes since it was filled.
    Meanwhile, 16 items have been sold from this column.

Connection closed by foreign host.
$
```

В данном примере мы напрямую соединились с портом Finger, набрали имя пользователя «cola», и сервер вернул требуемую информацию.

Я выбрал этот узел и этого пользователя только затем, чтобы показать, какие чудеса творились на заре появления Интернета. Серверы Finger превратились в службы для задач различного вида. В этом случае кто угодно может посмотреть, заполнен ли автомат газированных напитков, расположенный на факультете компьютерных наук в университете Копенгагена. Примеры странных устройств, подключенных к серверам Finger, можно найти на страницах Беннета Йи (Bennet Yee) в «Internet Accessible Coke Machines» и «Internet Accessible Machines» на *http://www.cs.ucsd.edu/~bsy/fun.html*.

Перенесем сетевое соединение, установленное с помощью *telnet*, в мир Perl. Средствами Perl можно открыть сокет и общаться через него. Вместо того чтобы применять низкоуровневые команды сокета, воспользуемся модулем `Net::Telnet` Джея Роджера (Jay Roger) и познакомимся с семейством модулей, работающих с сетевыми соединениями. Другие модули этого семейства (некоторые из них будут применяться в иных главах) включают *Comm.pl* Эрика Арнольда (Eric Arnold), *Expect.pm* Остина Шатца (Austin Schutz) и хорошо известный, но устаревший и непереносимый модуль *chat2.pl* Рэндала Л. Шварца (Randal L. Schwartz).

`Net::Telnet` устанавливает соединение и обеспечивает четкий интерфейс для отправки и получения данных через это соединение. Кроме того, `Net::Telnet` предоставляет удобные механизмы сканирования шаблонов, позволяющие программам наблюдать за определенными ответами от другого сервера, но в примере подобные свойства использоваться не будут.

Вот `Net::Telnet`-версия простого Finger-клиента. Эта программа принимает аргумент в виде *user@finger_server*. Если имя пользователя пропущено, будет возвращен список всех активных пользователей с сервера. Если пропущено имя узла, будет запрашиваться локальный узел:

```
use Net::Telnet;

($username,$host) = split(/\@/,$ARGV[0]);
```

```
$host = $host ? $host : 'localhost';

# создаем новое соединение
$cn = new Net::Telnet(Host => $host,
                      Port => 'finger');

# посылаем имя пользователя
unless ($cn->print("$username")){ # может быть "/W $username"
    $cn->close;
    die "Невозможно отправить строку: ".$cn->errmg."\n";
}

# собираем все полученные данные, останавливаясь при завершении соединения
while (defined ($ret = $cn->get)[1]) {
    $data .= $ret;
}

# закрываем соединение
$cn->close;

# отображаем полученные данные
print $data;
```

В RFC1288 определено, что перед именем пользователя, отправляемым на сервер, можно добавить ключ /W для «вывода подробной информации о пользователе», поэтому в программу добавлен комментарий об этом ключе.

Если нужно соединиться, используя помимо Finger другой текстовый протокол на основе TCP, можно применить очень похожую программу. Например, для соединения с сервером Daytime (который выводит локальное время) используется очень похожая программа:

```
use Net::Telnet;

$host = $ARGV[0] ? $ARGV[0] : 'localhost';

$cn = new Net::Telnet(Host => $host,
                      Port => 'daytime');

while (defined ($ret = $cn->get)[2]) {
    $data .= $ret;
}
$cn->close;
```

[1] Скобки поставлены в соответствии с требованиями Perl 5.6.1. В тексте оригинала скобки отсутствуют. – *Примеч. науч. ред.*

[2] Скобки поставлены в соответствии с требованиями Perl 5.6.1. В тексте оригинала скобки отсутствуют. – *Примеч. науч. ред.*

```
print $data;
```

Теперь читатель имеет представление о том, насколько легко создавать типовые сетевые клиенты на основе TCP. Если кто-то уже потратил время и написал модуль, специально созданный для работы с протоколом, все окажется еще проще. В случае с Finger можно воспользоваться модулем `Net::Finger` и заменить все вызовом одной функции:

```
use Net::Finger;

# finger() принимает строку user@host и возвращает полученные данные
print finger($ARGV[0]);
```

Желая показать все варианты, упомянем о возможности вызвать внешнюю программу (если она существует):

```
($username,$host) = split('@',$ARGV[0]);
$host = $host ? $host : 'localhost';

# местоположение команды finger executable, пользователи MacOS
# этим методом воспользоваться не могут
$fingerex = ($^O eq "MSWin32") ?
                $ENV{'SYSTEMROOT'}."\\System32\\finger" :
                "/usr/ucb/finger";  # (также может быть и /usr/bin/finger)

print `$fingerex ${username}\@${host}`
```

Вы познакомились с тремя различными способами выполнения Finger-запросов. Третий метод, вероятно, самый неудачный, т. к. в нем порождается другой процесс. `Net::Finger` обрабатывает простые Finger-запросы; все остальное может очень хорошо выполнить `Net::Telnet` или родственные ему модули.

Служба каталогов WHOIS

WHOIS – это еще одна полезная служба каталогов, предоставляющая доступную только для чтения информацию. WHOIS обеспечивает услуги, подобные телефонному справочнику для машин, сетей и людей. Некоторые крупные организации, такие как IBM, UC Berkeley и MIT, предоставляют услуги WHOIS, но самые известные WHOIS-серверы принадлежат InterNIC и другим компаниям, занимающимся вопросами регистрации в Интернете, в том числе RIPE (имеет дело с европейскими IP-адресами) и APNIC (Asia/Pacific, Азиатские/Тихоокеанские адреса).

При необходимости связаться с системным администратором какого-либо узла, чтобы сообщить ему о подозрительных действиях в сетях, следует использовать службу WHOIS для получения контактной информации. В большинстве операционных систем для выполнения

WHOIS-запросов существуют как GUI-инструменты, так и инструменты, запускаемые из командной строки. Типичный запрос в Unix выглядит так:

```
% whois -h whois.networksolutions.com brandeis.edu
<large legal paragraph omitted>
Registrant:
Brandeis University (BRANDEIS-DOM)
   Information Technology Services
   Waltham, MA 02454-9110
   US

   Domain Name: BRANDEIS.EDU

   Administrative Contact:
      Koskovich, Bob  (BK138)  user@BRANDEIS.EDU
      +1-781-555-1212 (FAX) +1-781-555-1212
   Technical Contact, Zone Contact:
      Hostmaster, Brandeis C  (RCG51)  hostmaster@BRANDEIS.EDU
      +1-781-555-1212 (FAX) +1-781-555-1212
   Billing Contact:
      Koskovich, Bob  (BK138)  user@BRANDEIS.EDU
      +1-781-555-1212 (FAX) +1-781-555-1212

   Record last updated on 13-Oct-1999.
   Record created on 27-May-1987.
   Database last updated on 19-Dec-1999 17:42:19 EST.

   Domain servers in listed order:

   LILITH.UNET.BRANDEIS.EDU     129.64.99.12
   FRASIER.UNET.BRANDEIS.EDU    129.64.99.11
   DIAMOND.CS.BRANDEIS.EDU      129.64.2.3
   DNSAUTH1.SYS.GTEI.NET        4.2.49.2
   DNSAUTH2.SYS.GTEI.NET        4.2.49.3
```

Если же нужно выяснить владельца определенного диапазона IP-адресов, то и тут поможет WHOIS:

```
% whois -h whois.arin.net 129.64.2
Brandeis University (NET-BRANDEIS)
   415 South Street
   Waltham, MA 02254

   Netname: BRANDEIS
   Netnumber: 129.64.0.0

   Coordinator:
      Koskovich, Bob  (BK138-ARIN)  user@BRANDEIS.EDU
      617-555-1212
```

```
   Domain System inverse mapping provided by:

   BINAH.CC.BRANDEIS.EDU        129.64.1.3
   NIC.NEAR.NET                 192.52.71.4
   NOC.CERF.NET                 192.153.156.22

   Record last updated on 10-Jul-97.
   Database last updated on 9-Oct-98 16:10:44 EDT.

The ARIN Registration Services Host contains ONLY Internet
Network Information: Networks, ASN's, and related POC's.
Please use the whois server at rs.internic.net for DOMAIN related
Information and nic.mil for NIPRNET Information.
```

В предыдущем примере применялся WHOIS-клиент из Unix, работающий в командной строке. В Windows NT и MacOS подобные клиенты не входят, тем не менее, это не должно остановить пользователей данных систем от получения доступа к нужной информации. Существует много условно бесплатных клиентов, но не так трудно с помощью модуля `Net::Whois` создать на Perl очень простой клиент (модуль `Net::Whois` первоначально был написан Чипом Салзенбергом (Chip Salzenberg), а теперь поддерживается Даной Хьюдес (Dana Hudes)). Следующий код – это лишь несколько измененная версия примера из документации, поставляемой вместе с модулем:

```
use Net::Whois;

# запрашиваем сервер, возвращая объект с результатами
my $w = new Net::Whois::Domain $ARGV[0] or
   die "Невозможно соединиться с сервером Whois\n";
die "Никакой информации о домене $ARGV[0] не найдено\n " unless ($w->ok);

# выводим части этого объекта
print "Домен: ", $w->domain, "\n";
print "Имя: ", $w->name, "\n";
print "Тег: ", $w->tag, "\n";
print "Адрес:\n", map { "    $_\n" } $w->address;
print "Страна: ", $w->country, "\n";
print "Запись создана: ".$w->record_created."\n";
print "Запись обновлена: ".$w->record_updated."\n";

# выводим серверы имен ($w->servers returns a list of lists)
print "Серверы имен:\n", map { "    $$_[0] ($$_[1])\n" } @{$w->servers};

# выводим список контактов ($w->contacts returns a hash of lists)
my($c,$t);
if ($c = $w->contacts) {
    print "Contacts:\n";
    for $t (sort keys %$c) {
        print "    $t:\n";
```

```
            print map { "\t$_\n" } @{$$c{$t}};
        }
    }
```

Запрос WHOIS сервера InterNIC/Network Solutions – это простой процесс. Для возвращения результата применяется мо«дуль `Net::Whois::Domain`. Методы этого объекта, названные в соответствии с полями, которые получает WHOIS-запрос, обеспечивают доступ к данным.

WHOIS предстоит сыграть значительную роль в главе 8 «Электронная почта», а сейчас перейдем к более сложным службам каталогов. Мы уже начали этот переход, переключаясь со службы Finger на WHOIS. Между рассмотренными способами использования Finger и WHOIS существует важное различие – структура.

Вывод Finger отличается от реализации к реализации. И хотя существуют некоторые соглашения, форму он имеет свободную. WHOIS-сервер InterNIC/Network Solutions возвращает данные более постоянной структуры. Можно рассчитывать на то, что у каждой записи будут, по крайней мере, поля `Name`, `Address` и `Domain`. Модуль `Net::Whois` полагается на эту структуру и анализирует результаты, разбивая их на поля. Существует еще один модуль Випула Вед Пракаша (Vipul Ved Prakash) – `Net::Xwhois`, который делает шаг вперед, обеспечивая интерфейс для анализа информации, по-разному отформатированной различными WHOIS-серверами.

И хотя в протоколе WHOIS нет никакого упоминания о полях, вызываемые нами модули начинают полагаться на структуру информации. Службы каталогов, о которых пойдет речь, более серьезно относятся к этой структуре.

LDAP: сложная служба каталогов

Службы LDAP (Lightweight Directory Access Protocol, облегченный протокол доступа к каталогам) и ADSI гораздо богаче и более сложны в обращении. В настоящее время существуют две популярные версии протокола LDAP (версия 2 и версия 3; при необходимости номер версии будет указываться). Этот протокол быстро стал промышленным стандартом для доступа к каталогам. Системные администраторы воспользовались протоколом LDAP, т. к. он предлагал способ централизовать и сделать доступной всю информацию об инфраструктуре. Помимо стандартного «каталога компании» существуют такие примеры приложений:

- Шлюзы NIS-к-LDAP
- Шлюзы Finger-к-LDAP
- Всевозможные базы данных аутентификации (в частности, для использования в Сети)

- Объявления о ресурсах (какие машины и периферийные устройства доступны)

Кроме того, LDAP является базой для других сложных служб каталогов, подобных активным каталогам Microsoft (Microsoft Active Directory), о которых пойдет речь в разделе «ADSI (Интерфейсы служб активных каталогов)».

Даже в случае, когда LDAP применяется только для ведения «домашней» телефонной книги, существуют веские причины научиться использовать этот протокол. LDAP-серверы можно администрировать при помощи этого же протокола; что очень напоминает серверы баз данных SQL, которые тоже можно администрировать средствами SQL. И в этом случае Perl предлагает отличные механизмы склейки для автоматизации задач администрирования LDAP. Но сначала нужно убедиться, что протокол LDAP разобран и понятен.

В приложении B «Десятиминутное руководство по LDAP» приводится краткое введение в LDAP для тех, кто не знаком с протоколом. Самая большая преграда, встающая перед системным администратором при изучении LDAP, – это неуклюжая терминология, унаследованная от родительских протоколов службы каталогов X.500. LDAP является упрощенной версией X.500, но, к сожалению, терминология от этого легче не стала. Стоит потратить время на приложение B и изучение терминов – тогда вам будет проще понять, как использовать LDAP из Perl.

Программирование LDAP на Perl

Как и с многими другими задачами системного администрирования в Perl, первым делом при программировании LDAP следует выбрать нужный модуль. Хотя LDAP еще не самый сложный протокол, но это уже и не обычный текстовый протокол. В результате, создать что-нибудь, «говорящее» на LDAP, задача нетривиальная. К счастью, другие авторы уже сделали эту работу за нас: Грэм Бар (Graham Barr) написал модуль `Net::LDAP`, а Лейф Хедстром (Leif Hedstrom) и Клейтон Донли (Clayton Donley) создали модуль `Mozilla::LDAP` (также известный как `PerLDAP`). Отметим некоторые различия между этими двумя модулями (табл. 6.1).

Таблица 6.1. Сравнение двух LDAP-модулей

Возможность	Net::LDAP	Mozilla::LDAP (PerLDAP)
Переносимость	Только Perl	Требует Mozilla/Netscape LDAP C-SDK (исходный код свободно доступен). SDK компилируется на многих вариантах Unix, NT и MacOS
Зашифрованные SSL-сеансы	Да	Да
Асинхронные операции	Да	Только с не объектно-ориентированными API низкого уровня

Оба модуля обладают функциональностью, необходимой для выполнения обсуждаемых ниже простых задач, связанных с системным администрированием, но предоставляют ее немного различными способами. Это создает, с точки зрения обучения, редкую возможность увидеть, как два разных автора реализовали важные модули, применимые в одной и той же области. Скрупулезное сравнение обоих модулей поможет разобраться с процессом их содания, что и будет показано в главе 10 «Безопасность и наблюдение за сетью». Для облегчения сравнения в большей части примеров из этого раздела приведен синтаксис обоих LDAP-модулей. Строка `use modulename` в тексте каждого примера подскажет, какой модуль используется на этот раз.

В демонстрационных целях мы почти равнозначно будем использовать коммерческий сервер Netscape 4.0 Directory Server и свободно распространяемый сервер OpenLDAP (они находятся на *http://www.netscape.com* и *http:// www.openldap.org*). В состав обоих серверов входят практически идентичные утилиты командной строки, которые можно использовать для прототипирования и проверки программ на Perl.

Первоначальное LDAP-соединение

Соединение с аутентификацией – это, обычно, первый шаг в любой клиент-серверной LDAP-транзакции. На «языке» LDAP это называется «связыванием с сервером» (binding to the server). В LDAPv2 требовалось связаться с сервером перед отправкой команд, в LDAPv3 условия не такие жесткие.

Связывание с LDAP-сервером выполняется в контексте определенного отличительного имени (Distinguished name, DN), описанного как *привязанное отличительное имя* (*bind DN*) для данного сеанса. Такой контекст похож на регистрацию пользователя в многопользовательской системе. В многопользовательской системе текущее регистрационное имя (по большей части) определяет уровень доступа пользователя к данным в этой системе. В LDAP именно привязанное отличительное имя определяет, какие данные на LDAP-сервере доступны для просмотра и изменения. Существует также специальное *корневое отличительное имя* (*root Distinguished Name*), дабы не путать его с относительным (Relative Distinguished Name) именем, для которого не существует акронима. Корневое отличительное имя имеет полный контроль над всем деревом, что очень похоже на регистрацию с правами пользователя *root* в Unix или с правами пользователя *Administrator* в NT/2000. На некоторых серверах это имя называется *manager DN*.

Если клиент не предоставляет аутентификационной информации (например, DN-имя и пароль) во время связывания или вообще не связывается с сервером до оправки команд, это называется *анонимной аутентификацией* (*anonymous authentication*). Анонимно зарегистрированные клиенты обычно получают очень ограниченный доступ к данным на сервере.

В спецификации LDAPv3 определяется два типа связывания: простой и SASL. В простом связывании для аутентификации используются обычные текстовые пароли. SASL (Simple Authentication and Security Layer, слой простой аутентификации и безопасности) – это расширенный интерфейс аутентификации, определенный в RFC2222, позволяющий авторам клиентов/серверов встраивать различные схемы аутентификации, подобные Kerberos и одноразовым паролям. Когда клиент соединяется с сервером, он запрашивает определенный механизм аутентификации. Если сервер его поддерживает, он начнет диалог, соответствующий такому механизму, для аутентификации клиента. Во время этого диалога клиент и сервер могут договориться об уровне безопасности (например, «весь трафик между нами будет зашифрован при помощи TLS»), применяемом после завершения аутентификации.

Некоторые серверы и клиенты LDAP добавляют еще один метод аутентификации к SASL и стандартному простому способу. Этот метод является побочным продуктом использования LDAP по зашифрованным SSL (Secure Socket Layer, уровень защищенных сокетов). Для установки этого канала серверы и клиенты LDAP обмениваются криптографическими сертификатами на основе открытого ключа так же, как веб-сервер и броузеры при работе по протоколу HTTPS. LDAP-серверу можно дать указание использовать в качестве аутентификационной информации только надежный клиентский сертификат (trusted client's certificate). Из доступных Perl-модулей только PerLDAP предлагает LDAPS (зашифрованные SSL-соединения). В примерах, для того чтобы не слишком усложнять их, будем пользоваться только простой аутентификацией и незашифрованными соединениями.

Вот как выполняется простое соединение и его завершение средствами Perl:

```
use Mozilla::LDAP::Conn;
# используем пустые $binddn и $passwd для анонимной связи
$c = new Mozilla::LDAP::Conn($server, $port, $binddn, $passwd);
die "Невозможно соединиться с $server" unless $c;
...
$c->close();
```

или:

```
use Net::LDAP;
$c = Net::LDAP->new($server, port => $port) or
    die " Невозможно соединиться с $server: $@\n";
# не передаем параметры bind() для анонимной связи
$c->bind($binddn, password => $passwd) or
    die "Невозможно соединиться: $@\n";
...
$c->unbind();
```

В `Mozilla::LDAP::Conn` создание нового объекта соединения также связано с сервером. В `Net::LDAP` этот процесс состоит из двух шагов. Для инициализации соединения без выполнения привязывания в `Mozilla::LDAP` необходимо использовать функцию (`ldap_init()`) из не объектно-ориентированного модуля `Mozilla::LDAP::API`.

Приготовьтесь тщательно заключить в кавычки значения атрибутов

Небольшой совет перед тем, как перейти к дальнейшему программированию на Perl: если в относительном отличительном имени есть атрибут, значение которого содержит один из следующих символов: «+», «(пробел)», «,», «'», «>», «<» или «;», необходимо либо заключить значение в кавычки, либо экранировать эти символы обратным слэшем (\). Если значение содержит кавычки, их также нужно экранировать при помощи обратного слэша. Обратные слэши в значениях тоже экранируются обратными слэшами.

Если вы не будете аккуратны, то недостаток кавычек может сыграть с вами злую шутку.

Выполнение поиска в LDAP

Буква «D» в LDAP означает Directory (т. е. каталог), и наиболее распространенной операцией с каталогами является поиск. Для начала знакомства с LDAP неплохо выяснить, как искать информацию. Поиск в LDAP определяется такими понятиями:

Откуда начинать поиск

Это называется *базовым относительным именем* (base DN) или *базой поиска* (*search base*). Базовое DN-имя представляет собой всего лишь DN-имя элемента в дереве каталогов, от которого начинается поиск.

Где искать

Это называется *пространством* (*scope*) поиска. Пространство может быть трех видов: *base* (поиск только по базовому DN-имени), *one* (поиск по уровню, лежащему непосредственно под базовым DN-именем, не включая само базовое DN-имя) или *sub* (поиск по базовому DN-имени и всему дереву, лежащему ниже).

Что искать

Это называется *фильтрами поиска* (*search filter*). О фильтрах и их определении мы поговорим очень скоро.

Что возвращать

С целью ускорения операций поиска можно выбрать, какие атрибуты должны возвращаться для каждого элемента, найденного при помощи фильтров поиска. Кроме того, можно запросить, чтобы воз-

вращались только имена атрибутов, а не их значения. Это полезно, когда нужно узнать, у каких элементов есть данные атрибуты, но совсем не важно, что эти атрибуты содержат.

В Perl поиск выглядит примерно так (процесс соединения заменен многоточиями):

```
use Mozilla::LDAP::Conn;
...
$entry = $c->search($basedn, $scope, $filter);
die "Неуспешный поиск: ". $c->getErrorString()."\n" if $c->getErrorCode();
```

или:

```
use Net::LDAP;
...
$searchobj = $c->search(base => $basedn, scope => $scope,
                        filter => $filter);
die "Неуспешный поиск, номер ошибки #".$searchobj->code() if $searchobj-
>code();
```

Перед тем как перейти к полному примеру, поговорим о загадочном параметре `$filter`. Простые фильтры поиска имеют следующий вид:

```
<attribute name> <comparison operator> <attribute value>
```

где *<comparison operator>* определяется в RFC2254 как один из операторов, перечисленных в табл. 6.2.

Таблица 6.2. Операторы сравнения LDAP

Оператор	Значение
`=`	Точное совпадение значений. Может означать и частичное совпадение, если в определении *<attribute value>* используется * (например `cn=Tim O*`).
`=*`	Соответствует всем элементам, у которых есть значения для атрибута *<attribute name>*, независимо от того, каковы эти значения. Если вместо *<attribute value>* указать *, будет проверяться наличие именно этого атрибута в элементе (например, `cn=*` выберет элементы, у которых есть атрибуты `cn`).
`~=`	Приблизительное совпадение значений.
`>=`	Больше либо равно значению.
`<=`	Меньше либо равно значению.

Это очень похоже на Perl, но не заблуждайтесь. Две конструкции, которые могут смутить знатоков Perl, это `~=` и `=*`. Первая из них не имеет ничего общего с регулярными выражениями; она ищет приблизительное соответствие с указанным значением. В этом случае определение «приблизительное» зависит от сервера. Большинство серверов применяют алгоритм, первоначально используемый в `soundex` для определе-

ния совпадающих значений при поиске слов, которые «произносятся, как» заданное значение (в английском языке), но записываются иначе.[1]

Другая конструкция, которая может конфликтовать с вашими знаниями Perl, – это оператор =. Помимо проверки точного совпадения значений (как строковых, так и численных), оператор = можно использовать вместе с символом * в виде префикса или суффикса в качестве символов подстановки, подобно тому как это происходит в командных интерпретаторах. Например, `cn=a*` получит все элементы, имена которых (common name) начинаются с буквы «a». Строка `cn=*a*` выполнит именно то, чего вы ждете, и найдет все элементы, в атрибуте `cn` которых есть буква «a».

Можно объединить в одну строку два или более простых фильтра *<attribute name>*, *<comparison operator>*, *<attribute value>* при помощи логических операторов, создав таким образом более сложный фильтр. Он имеет следующий вид:

```
(<boolean operator> (<simple1>) (<simple2>) (<simple3>) ... )
```

Те, кто знаком с LISP, без труда разберутся с подобным синтаксисом; всем остальным придется просто запомнить, что оператор, объединяющий простые формы поиска, записывается первым. Чтобы найти элементы, удовлетворяющие обоим критериям поиска A *и* B, нужно использовать запись `(&(A)(B))`. Для элементов, удовлетворяющих критериям A *или* B *или* C, следует применить `(|(A)(B)(C))`. Восклицательный знак отрицает указанный критерий: так, A *и не* B записывается следующим образом: `(&(A)(!B))`. Составные фильтры можно объединять друг с другом, чтобы создавать фильтры поиска произвольной сложности. Вот пример составного фильтра для поиска всех Финкельштейнов, работающих в Бостоне:

```
(&(sn=Finkelstein)(l=Boston))
```

Следующий фильтр ищет человека, чья фамилия либо Финкельштейн, либо Хайндс:

```
(|(sn=Finkelstein)(sn=Hinds))
```

Для поиска всех Финкельштейнов, работающих не в Бостоне:

```
(&(sn=Finkelstein)(!(l=Boston)))
```

Для поиска всех Финкельштейнов или Хайндсов, работающих не в Бостоне:

```
(&(|(sn=Finkelstein)(sn=Hinds))(!l=Boston))
```

[1] Тот, кто захочет поэкспериментировать с алгоритмом soundex, может воспользоваться модулем Марка Милке (Mark Mielke) `Text::Soundex`.

Вот два примера программ, принимающих имя LDAP-сервера и фильтр и возвращающих результаты запроса:

```
use Mozilla::LDAP::Conn;

$server = $ARGV[0];
$port   = getservbyname("ldap","tcp") || "389";
$basedn = "c=US";
$scope  = "sub";

$c = new Mozilla::LDAP::Conn($server, $port, "","");  # анонимное соединение
die "Невозможно связаться с $server\n" unless $c;

$entry = $c->search($basedn, $scope, $ARGV[1]);
die "Ошибка поиска: ". $c->getErrorString()."\n" if $c->getErrorCode();

# обрабатываем полученные от search() значения
while ($entry) {
    $entry->printLDIF();
    $entry = $c->nextEntry();
}
$c->close();
```

```
use Net::LDAP;
use Net::LDAP::LDIF;

$server = $ARGV[0];
$port   = getservbyname("ldap","tcp") || "389";
$basedn = "c=US";
$scope  = "sub";

$c = new Net::LDAP($server, port=>$port) or
    die "Невозможно соединиться с $server: $@\n";
$c->bind() or die "Unable to bind: $@\n"; # анонимное соединение

$searchobj = $c->search(base => $basedn, scope => $scope,
                        filter => $ARGV[1]);
die " Неуспешный поиск, номер ошибки #".$searchobj->code() if $searchobj-
>code();

# обрабатываем полученные от search() значения
if ($searchobj){
    $ldif = new Net::LDAP::LDIF("-");
    $ldif->write($searchobj->entries());
    $ldif->done();
}
```

А вот отрывок из получаемых данных:

```
$ ldapsrch ldap.bigfoot.com '(sn=Pooh)'
...
```

```
dn: cn="bear pooh",mail=poohbear219@hotmail.com,c=US,o=hotmail.com
mail: poohbear219@hotmail.com
cn: bear pooh
o: hotmail.com
givenname: bear
surname: pooh
...
```

Перед тем как улучшить этот пример, посмотрим на код, обрабатывающий результаты, полученные от `search()`. Это одно из тех мест, где модули отличаются моделью программирования. Оба примера возвращают одну и ту же информацию в формате LDIF (LDAP Data Interchange Format, формат обмена данными LDAP), о котором речь пойдет позже, но данные они получают совершенно разными способами.

Модель `Mozilla::LDAP` остается справедливой для подпрограмм анализа поиска, описанных в спецификации C API в RFC1823. Если поиск был успешным, возвращается первый найденный элемент. Для просмотра результатов необходимо последовательно запросить следующие элементы. Вывод содержимого каждого получаемого элемента выполняет метод `printLDIF()`.

Модель программирования `Net::LDAP` имеет больше сходства с определением протокола из RFC2251. Результаты поиска LDAP возвращаются в объекты сообщений. Для получения списка элементов из этих пакетов в предыдущем примере использовался метод `entries()`. Вывод всех элементов вместе выполняет метод из смежного модуля `Net::LDAP::LDIF`. Для последовательного вывода всех элементов, как было с `printLDIF()` в первом примере, можно использовать похожий метод `write()`, но показанный выше вызов более эффективен.

Немного поработаем с предыдущим примером. Как уже отмечалось в данной главе, поиск можно выполнять быстрее, ограничив количество возвращаемых в результате атрибутов. С модулем `Mozilla::LDAP` это настолько же просто, насколько просто добавить дополнительные параметры в вызов метода `search()`:

```
use Mozilla::LDAP::Conn;
...
$entry = $c->search($basedn,$scope,$ARGV[1],0,@attr);
```

Первый дополнительный параметр – это логический флаг, определяющий, будут ли значения атрибутов опущены в результатах поиска. Значение по умолчанию – ложь (0), т. к. в большинстве случаев нас интересуют не только имена атрибутов.

Следующий дополнительный параметр – это список имен возвращаемых атрибутов. Знатоки Perl заметят, что список внутри списка интерполируется, так что последняя строка эквивалентна строке (ее можно так и прочитать):

```
$entry =
  $c->search($basedn,$scope,$ARGV[1],0,$attr[0],$attr[1],$attr[2],...);
```

Если мы изменим строку первоначального примера:

```
$entry = $c->search($basedn,$scope,$ARGV[1]);
```

на:

```
@attr = qw(mail);
$entry = $c->search($basedn,$scope,$ARGV[1],0,@attr);
```

то получим следующий результат, в котором для элемента будут показаны только атрибуты DN и mail:

```
...
dn: cn="bear pooh",mail=poohbear219@hotmail.com,c=US,o=hotmail.com
mail: poohbear219@hotmail.com
...
```

Изменения, которые необходимо внести, чтобы получить определенные атрибуты средствами `Net::LDAP`, тоже не сложны:

```
use Net::LDAP;
...
# можно было бы добавить "typesonly => 1" для получения только
# типов атрибутов, как и в предыдущем случае для первого
# необязательного параметра
$searchobj = $c->search(base => $basedn, filter => $ARGV[1],
                        attrs => \@attr);
```

Обратите внимание, что `Net::LDAP` принимает *ссылку* на массив, а не сами значения массива, как в случае с `Mozilla::LDAP`.

Представление элементов в Perl

Эти примеры программ могут вызвать ряд вопросов о представлении элементов и о работе с ними, – в частности, как сами элементы хранятся и обрабатываются в программе на Perl. Дополняя рассказ о поиске в LDAP, ответим на некоторые из них, хотя позже в разделе о добавлении и изменении элементов подобные вопросы будут рассматриваться подробно.

Если `Mozilla::LDAP` выполняет поиск и возвращает экземпляр объекта элемента, то можно обратиться к отдельным атрибутам этого элемента, применяя синтаксис, используемый в Perl при работе с хэшами списков. `$entry->{attributename}` – это *список*[1] значений атрибута с таким именем. Я выделил слово «список», т. к. атрибуты даже с одним

[1] Если точнее, то ссылка на список (как, впрочем, далее явствует из текста). – *Примеч. науч. ред.*

значением хранятся в анонимном списке, на который ссылается этот ключ хэша. Для получения единственного значения атрибута необходимо использовать запись `$entry->{attributename}->[0]`. Некоторые методы модуля `Mozilla::LDAP::Entry` возвращают атрибуты элемента (табл. 6.3).

Таблица 6.3. Методы Mozilla::LDAP::Entry

Вызов метода	Возвращает
`$entry->exists($attrname)`	*true*, если элемент имеет атрибут с таким именем
`$entry->hasValue($attrname,$attrvalue)`	*true*, если элемент имеет названный атрибут с указанным значением
`$entry->matchValue($attrname,$attrvalue)`	Так же, как и предыдущий, только ищется соответствие регулярному выражению, определенному в качестве значения атрибута
`$entry->size($attrname)`	Количество значений этого атрибута (обычно 1, если только атрибут не обладает несколькими значениями)

Отдельные методы имеют дополнительные параметры, узнать о которых можно в документации по `Mozilla::LDAP::Entry`.

Из программы видно, что методы для доступа к атрибутам элементов в `Net::LDAP` несколько отличаются. После проведения поиска все результаты инкапсулируются в один объект. Получить отдельные атрибуты каждого элемента из этого объекта можно, применив один из двух способов.

Во-первых, модуль может преобразовать все полученные элементы в одну большую структуру данных, доступную пользователям. `$searchobj->as_struct()` возвращает структуру данных, представляющую собой хэш хэшей списков. Метод возвращает ссылку на хэш, ключами которого являются DN-имена полученных элементов. Значения ключей – это ссылки на анонимные хэши, ключами которых являются имена атрибутов. Ключам соответствуют ссылки на анонимные массивы, содержащие значения данных атрибутов (рис. 6.1).

Вывести первое значение атрибута `cn` для всех элементов из структуры данных позволяет такой код:

```
$searchstruct = $searchobj->as_struct;
for (keys %$searchstruct){
    print $searchstruct->{$_}{cn}[0],"\n";
}
```

Можно также сначала использовать один из этих методов и выделить объекты для отдельных элементов из объекта, возвращаемого в результате поиска:

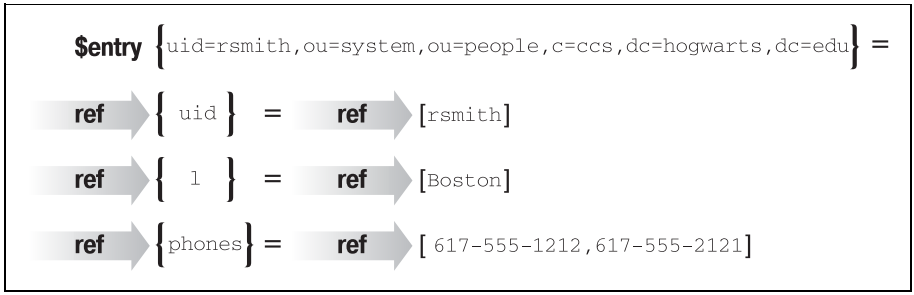

***Рис. 6.1.** Структура данных, возвращаемая методом as_struct( )*

```
# возвращает указанный элемент
$entry   = $searchobj->entry($entrynum);

# действует подобно shift() в Perl для списка элементов
$entry   = $searchobj->shift_entry;

# действет подобно pop() в Perl для списка элементов
$entry   = $searchobj->pop_entry;

# возвращает все элементы в виде списка
@entries = $searchobj->entries;
```

После того как получен объект элемента, можно применить один из указанных методов (табл. 6.4).

Таблица 6.4. Методы элементов Net::LDAP

Вызов метода	Возвращает
`$entry->get($attrname)`	Значение атрибута в указанном элементе
`$entry->attributes()`	Список имен атрибутов для этого элемента

Можно объединить эти методы в довольно четкую цепочку. Например, следующая строка получает значение атрибута `cn` первого возвращаемого элемента:

```
$value = $searchobj->entry(1)->get(cn)
```

Теперь, когда вы умеете получать доступ к отдельным атрибутам и значениям, возвращаемым в результате поиска, посмотрим, как поместить подобные данные в каталог сервера.

Добавление элементов при помощи LDIF

Перед тем как рассматривать общие методы добавления элементов в каталог LDAP, давайте вспомним о названии этой книги и рассмотрим технологию, полезную, в основном, системным администраторам и администраторам каталогов. Она использует формат данных, помогаю-

щий загрузить данные на сервер каталогов. Мы рассмотрим способы записи и чтения LDIF.

LDIF, определенный в нескольких стандартах RFC[1], предлагает простое текстовое представление для элементов каталогов. Вот простой пример LDIF из последнего чернового стандарта Гордона Гуда (Gordon Good):

```
version: 1
dn: cn=Barbara Jensen, ou=Product Development, dc=airius, dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Barbara Jensen
cn: Barbara J Jensen
cn: Babs Jensen
sn: Jensen
uid: bjensen
telephonenumber: +1 408 555 1212
description: A big sailing fan.

dn: cn=Bjorn Jensen, ou=Accounting, dc=airius, dc=com
objectclass: top
objectclass: person
objectclass: organizationalPerson
cn: Bjorn Jensen
sn: Jensen
telephonenumber: +1 408 555 1212
```

Формат должен быть вам понятен. После номера версии LDIF перечислены DN-имена каждого элемента, определения `objectClass` и атрибуты. Разделителем элементов является пустая строка.

Наша первоочередная задача – научиться создавать файлы LDIF из существующих элементов каталогов. Кроме того что мы обеспечим себе данные для следующего раздела (в котором рассматривается чтение файлов LDIF), такая возможность позволит использовать LDIF-файлы любым способом при помощи обычных операций Perl, работающих с текстом.

При обсуждении поиска в LDAP было показано, как вывести элементы в формате LDIF. Изменим код предыдущего примера так, чтобы он записывал данные в файл:

```
use Mozilla::LDAP::Conn;
use Mozilla::LDAP::LDIF;

<выполняем связывание и поиск>
```

[1] В период написания книги они были доступны в черновом варианте. – *Примеч. науч. ред.*

```
open(LDIF,">$LDIFfile!") or die ""Невозможно записать в $LDIFfile:$!\n";
# создаем новый объект LDIF и передаем дескриптор
$ldif = new Mozilla::LDAP::LDIF(\*LDIF);

while ($entry) {
    $ldif->writeOneEntry($entry);
    $entry = $c->nextEntry();
}

$c->close();
close(LDIF);
```

Модуль `Mozilla::LDAP` располагает методом `writeEntries()`, позволяющим принять массив элементов и записать их подобным образом.

Используя `Net::LDAP`, изменить первоначальную программу еще проще. Вместо:

```
$ldif = new Net::LDAP::LDIF("-");
```

применим:

```
$ldif = new Net::LDAP::LDIF($filename,"w");
```

для записи выводимых данных в указанный файл, а не на стандартный вывод.

Теперь совершим обратное действие и прочитаем файлы LDIF (вместо того, чтобы в них записывать). Методы объекта из модуля, о котором пойдет речь, позволяют легко добавить элементы в каталог.[2]

При чтении LDIF-данных из Perl осуществляется процесс, обратный тому, который применялся в предыдущих примерах для записи. Каждый список элементов считывается и преобразуется в экземпляр объекта элемента, который затем передается соответствующему методу изменения каталога. Оба модуля считывают и анализируют данные, так что процесс довольно прост. Например, с использованием `Mozilla::LDAP` можно написать такую программу:

```
use Mozilla::LDAP::Conn;
use Mozilla::LDAP::LDIF;

$server   = $ARGV[0];
$LDIFfile = $ARGV[1];
$port     = getservbyname("ldap","tcp") || "389";
```

[1] Предполагается наличие строки «$LDIFfile=$ARGV[1];». – *Примеч. науч. ред.*

[2] Файлы LDIF могут содержать специальную директиву `changetype:`, которая говорит о том, что информацию об элементе необходимо удалить или изменить, а не просто добавить. Из двух применяемых модулей только `Net::LDAP` напрямую поддерживает `changetype:` через метод `Net::LDAP::LDIF::read_cmd()`.

```
$rootdn   = "cn=Manager, ou=Systems, dc=ccs, dc=hogwarts, dc=edu";
$pw       = "secret";

# считываем файл LDIF, указанный вторым аргументом в
# командной строке
open(LDIF,"$LDIFfile") or die "Невозможно открыть $LDIFfile:$!\n";
$ldif = new Mozilla::LDAP::LDIF(\*LDIF);

# анализируем все элементы, сохраняем их в @entries
@entries = $ldif->readEntries();
close(LDIF);

# неанонимное соединение
$c = new Mozilla::LDAP::Conn($server,$port,$rootdn,$pw);
die "Невозможно соединиться с $server\n" unless $c;

# обходим в цикле список элементов, добавляя их на каждой итерации
for (@entries){
    $c->add($_); # добавляем этот элемент в каталог
    warn "Ошибка при добавлении ". $_->getDN().": ".$c->getErrorString()."\n"
      if $c->getErrorCode();
}
$c->close();
```

В этом примере отражено применение методов `getErrorCode()` и `getErrorString()` для получения любых ошибок (и сообщения о них), происходящих в процессе загрузки данных. Ошибки могут появиться по целому ряду причин, включая дублирование DN/RDN-имен, нарушение схемы, проблемы с иерархией и т. д., так что очень важно проверить их при изменении элемента.

И еще одно замечание, перед тем как перейти к рассмотрению `Net::LDAP`: в этом и последующих примерах в демонстрационных целях используется корневое DN-имя (manager DN). Обычно же, если можно избежать применения такого контекста в повседневной работе, это следует делать. Образец правильной настройки LDAP-сервера включает создание могущественной учетной записи или группы учетных записей (которые не являются корневым DN-именем) для управления каталогами. При создании собственных приложений не забывайте этот совет.

Для `Net::LDAP` программа, добавляющая LDIF-элемент, выглядит таким образом:

```
use Net::LDAP;
use Net::LDAP::LDIF;

$server   = $ARGV[0];
$LDIFfile = $ARGV[1];
$port     = getservbyname("ldap","tcp") || "389";
$rootdn   = "cn=Manager, ou=Systems, dc=ccs, dc=hogwarts, dc=edu";
$pw       = "secret";
```

```
# считываем файл LDIF, указанный вторым аргументом в
# командной строке
# последний параметр "r" для чтения, "w" для записи
$ldif = new Net::LDAP::LDIF($LDIFfile,"r");
@entries = $ldif->read();

$c = new Net::LDAP($server, port => $port) or
    die "Невозможно соединиться с $server: $@\n";
$c->bind(dn => $rootdn, password => $pw) or die "Ошибка при связывании:
$@\n";

for (@entries){
    $res = $c->add($_);
    warn "Ошибка при добавлении ". $_->dn().": код ошибки ".$res->code."\n"
      if $res->code();
}

$c->unbind();
```

Несколько замечаний к этому примеру:

- При желании можно объединить два оператора чтения LDIF в одну строку:

```
@entries = new Net::LDAP::LDIF($LDIFfile,"r")->read;
```

- Если попытка `add()` не удалась, следует запросить десятичный код ошибки. Например, возможно такое сообщение:

```
Ошибка при добавлении cn=Ursula Hampster, ou=Alumni Association,
ou=People,
o=University of Michigan, c=US: код ошибки 68
```

  Если сервер возвращает текстовое сообщение, метод `error()` получает его так же, как это было в примере с `Mozilla::LDAP`:

```
print "Сообщение об ошибке: ".$res->error."\n";
```

  Безопаснее было бы проверить код возврата, как в предыдущем примере, поскольку серверы LDAP не всегда передают текстовые сообщения об ошибках в своих ответах. Если нужно преобразовать десятичный код ошибки в сообщение об ошибке или в название сообщения, модуль `Net::LDAP::Util` предлагает для этого две подпрограммы: `ldap_error_text()` и `ldap_error_name()`.

- 68 в десятичной системе счисления – это 44 в шестнадцатеричной. Следовательно, приведенная ниже строка из *Constant.pm* имеет прямое отношение к нашему случаю:

```
sub LDAP_ALREADY_EXISTS          () { 0x44 }
```

Теперь нам известно, что мы пытались добавить элемент из файла LDIF, который уже существовал в каталоге.

Добавление элементов при помощи стандартных операций LDAP

На этот раз мы заглянем вглубь процесса добавления элементов, чтобы научиться создавать и заполнять элементы вручную, не считывая их из файла, как в прошлый раз. Два модуля обрабатывают этот процесс по-разному, поэтому работать с ними следует отдельно. Модуль `Mozilla::LDAP` ближе к классическому стилю объектно-ориентированного программирования. Создадим новый экземпляр объекта:

```
use Mozilla::LDAP::Entry;
$e = new Mozilla::LDAP::Entry()
```

и начнем его заполнять. Следующий шаг – дать элементу отличительное имя DN. Это можно сделать при помощи метода `setDN()`:

```
$e->setDN("uid=jay, ou=systems, ou=people, dc=ccs, dc=hogwarts, dc=edu");
```

Для заполнения других атрибутов, таких как `objectClass`, следует пойти по одному из двух путей. Можно сделать ряд предположений относительно структуры данных, используемой для представления элемента (по существу, это хэш списков), и заполнить ее напрямую:

```
$e->{cn} = ['Jay Sekora'];
```

В данном случае используется имя атрибута в качестве ключа хэша и ссылка на анонимный массив, хранящий данные. Модуль `Mozilla::LDAP` ожидает, что значениями хэша будут ссылки на массив, а не сами данные, так что следующее, хоть и выглядит заманчиво, но будет неверным:

```
# воплощенное зло (или, по крайней мере, просто неверно)
$e->{cn} = 'Jay Sekora';
```

В качестве альтернативы можно действовать наверняка и применять метод объекта для добавления данных:

```
$e->addValue('cn', 'Jay Sekora');
```

Для добавления нескольких значений атрибуту нужно повторно вызывать метод `addValue()`:

```
$e->addValue('title', 'Unix SysAdmin');
$e->addValue('title', 'Part-time Lecturer');
```

Мне больше по душе второй подход, т. к. при его использовании менее вероятно, что программа перестанет работать, если в следующих версиях модуля изменится способ представления данных.

После того как элемент заполнен, можно вызвать метод `add()` для внесения его в каталог. Вот маленький сценарий, который добавляет эле-

мент в каталог. В качестве аргументов командной строки он принимает имя сервера, идентификатор пользователя (будет использоваться как часть отличительного имени) и общее имя:

```
use Mozilla::LDAP::Conn;

$server   = $ARGV[0];
$port     = getservbyname("ldap","tcp") || "389";
$suffix   = "ou=People, ou=Systems, dc=ccs, dc=hogwarts, dc=edu";
$rootdn   = "cn=Manager, ou=Systems, dc=ccs, dc=hogwarts, dc=edu";
$pw       = "secret";

# неанонимное соединение
$c = new Mozilla::LDAP::Conn($server,$port,$rootdn,$pw);
die "Невозможно соединиться с $server\n" unless $c;

$e = new Mozilla::LDAP::Entry;
# DN-имя - это идентификатор пользователя плюс суффикс,
# определяющий, куда поместить его в дереве каталогов
$e->setDN("uid=$ARGV[1],$suffix");
$e->addValue('uid', $ARGV[1]);
$e->addValue('cn', $ARGV[2]);
$c->add($e);
die "Ошибка при добавлении: ". $c->getErrorString()."\n" if $c-
>getErrorCode();
```

Обратите внимание, что в программе не выполняется проверка ошибок при вводе. Если вы пишете сценарий, который действительно может использоваться в интерактивном режиме, необходимо проверять вводимые данные, чтобы убедиться, что в них нет неэкранированных специальных символов, подобных запятым. Обратитесь к ранее приведенному «совету с совой» за разъяснениями о том, как заключать в кавычки значения атрибутов.

Теперь перейдем к `Net::LDAP`. При желании процесс добавления элементов для `Net::LDAP` может быть менее объектно-ориентированным. В него входят модуль `Entry` (`Net::LDAP::Entry`) и конструктор для экземпляра объекта элемента. Однако он содержит еще одну функцию `add()`, которая способна принимать структуру данных для добавления элемента за один шаг:

```
$res = $c->add(
    dn => 'uid=jay, ou=systems, ou=people, dc=ccs, dc=hogwarts, dc=edu',
    attr => [ 'cn'   => 'Jay Sekora',
              'sn'   => 'Sekora',
              'mail' => 'jayguy@ccs.hogwarts.edu',
              'title'=> ['Sysadmin','Part-time Lecturer'],
              'uid'  => 'jayguy',
            ]
            );
die "невозможно добавить, код ошибки #".$res->code() if $res->code();
```

На этот раз add() передается два аргумента.[1] Первый – это DN-имя для элемента; второй – ссылка на анонимный массив пар атрибут-значение. Обратите внимание, что атрибуты с несколькими значениями, например title, определяются при помощи вложенного анонимного массива. Тем, кто привык работать со структурами данных в Perl и кому не нравится объектно-ориентированный стиль программирования, такой подход придется больше по душе.

Удаление элементов

Удаление элементов из каталога – это простое дело (и необратимое, так что будьте осторожны). Вот отрывок программы, из которой, для краткости, снова удален код, реализующий соединение с сервером:

```
use Mozilla::LDAP::Conn;
...
# если у вас есть элемент, вы можете использовать
# $c->delete($entry->getDN())
$c->delete($dn) or
    die "Невозможно удалить элемент: ". $c->getErrorString()."\n";

use Net::LDAP;
...
$res = $c->delete($dn);
die "Невозможно удалить, код ошибки #".$res->code() if $res->code();
```

Важно обратить внимание на то, что в обоих модулях delete()удаляет по одному элементу за один раз. Если необходимо убрать поддерево целиком, сначала следует найти все дочерние элементы этого поддерева, используя пространство sub или one, а затем обойти в цикле возвращаемые значения, удаляя элементы на каждой итерации. После того как уничтожены дочерние элементы, можно удалить вершину этого поддерева.

Изменение имен элементов

Последние операции с LDAP, которые мы рассмотрим, касаются двух типов изменений элементов LDAP. Первый тип – это изменение DN- и RDN-имен. Преобразовать RDN-имя элемента просто, и эта операция поддерживается обоими модулями. Вот версия для Mozilla::LDAP:

```
use Mozilla::LDAP::Conn;
...
$c->modifyRDN($newRDN,$oldDN,$delold) or
    die "Невозможно переименовать элемент:". $c->getErrorString()."\n";
```

[1] Точнее говоря, с точки зрения синтаксиса Perl, передается четыре аргумента. Однако их можно рассматривать как два именованных аргумента. – *Примеч. науч. ред.*

В приведенном отрывке все должно быть понятно, за исключением параметра `$delold` метода `modifyRDN()`. Если он равен `true`, то LDAP-библиотеки удалят из элементов значения, совпадающие с измененными RDN-именами. Например, если первично в RDN-имени элемента содержался атрибут `l` (от «location», местоположение), но само RDN-имя было изменено, то старый атрибут `l` элемента будет удален и останется только новое значение.

Вот эквивалентный вариант для переименования элемента в `Net::LDAP`:

```
use Net::LDAP;
...
$res = $c->moddn($oldDN,
                 newrdn       => $newRDN,
                 deleteoldrdn => 1);
die "Невозможно переименовать, код ошибки #".$res->code() if $res->code();
```

В действительности метод `moddn()` модуля `Net::LDAP` может гораздо больше, чем показано в предыдущем примере. До сих пор изменялось только RDN-имя элемента, в то время как местоположение элемента в иерархии дерева каталогов оставалось прежним. В LDAP версии 3 появилась более мощная операция для переименования, позволяющая произвольным образом менять местоположение элемента в дереве каталогов. Метод `moddn()`, вызванный с дополнительным параметром `newsuperior`, предоставляет доступ к такой возможности. Если добавить параметр таким образом:

```
$result = $c->moddn($oldDN,
                    newrdn       => $newRDN,
                    deleteoldrdn => 1,
                    newsuperior  => $parentDN);
die "Невозможно переименовать, код ошибки #".$res->code() if $res->code();
```

то элемент из `$oldDN` будет перенесен и станет дочерним элементом DN-имени, определенного в `$parentDN`. Гораздо эффективнее использовать этот метод, а не последовательность `add()` или `delete()`, как требовалось раньше, для перемещения элементов в дереве каталогов, но подобная возможность поддерживается не всеми LDAP-серверами. В любом случае, если вы скрупулезно проектируете структуру дерева каталогов, вам реже придется переносить элементы с места на место.

Изменение атрибутов элемента

Теперь перейдем к более распространенным операциям – изменению атрибутов и значений атрибутов элемента. В этом случае тоже существуют значительные различия между модулями `Mozilla::LDAP` и `Net::LDAP`. Применяя `Mozilla::LDAP` для изменения атрибута элемента,

необходимо использовать один из методов, представленных в табл. 6.5.

Таблица 6.5. Методы изменения элементов в Mozilla::LDAP

Метод	Действие
`$entry->addValue($attrname, $attrvalue)`	Добавляет указанное значение заданному атрибуту в указанном элементе.
`$entry-> removeValue($attrname, $attrvalue)`	Удаляет указанное значение для заданного атрибута указанного элемента. Если это значение единственное для атрибута, то удаляется и весь атрибут.
`$entry-> setValue($attrname, $attrvalue1,…)`	Изменяет значения указанного атрибута в заданное значение или значения.
`$entry-> remove($attrname)`	Удаляет указанный атрибут (вместе со значениями) из элемента.

После того как внесены все изменения элементов (при помощи перечисленных методов), нужно вызвать метод `update()` для данного LDAP-соединения, чтобы распространить эти изменения на сервер каталогов. `update()` вызывается со ссылкой на элемент в качестве аргумента (т. е. `$c->update($entry)`).

Применим эти методы для глобального поиска и замены. Рассмотрим такой сценарий: один из отделов вашей компании переводят из Бостона в Индиану. Эта программа изменит все элементы, местоположением которых является Бостон:

```
use Mozilla::LDAP::Conn;

$server = $ARGV[0];
$port   = getservbyname("ldap","tcp") || "389";
$basedn = "dc=ccs,dc=hogwarts,dc=edu";
$scope  = "sub";
$rootdn = "cn=Manager, ou=Systems, dc=ccs, dc=hogwarts, dc=edu";
$pw     = "secret";

# неанонимное соединение
$c = new Mozilla::LDAP::Conn($server,$port,$rootdn,$pw);
die "Невозможно соединиться с сервером $server\n" unless $c;

# обратите внимание, что мы запрашиваем как можно меньше
# информации для ускорения поиска
$entry = $c->search($basedn,$scope,"(l=Boston)",1,'');
die "Ошибка поиска:". $c->getErrorString()."\n" if $c->getErrorCode();

if ($entry){
    while($entry){
        $entry->removeValue("l","Boston");
```

```
            $entry->addValue("l","Indiana");
            $c->update($entry);
            die "Ошибка при обновлении:" . $c->getErrorString() . "\n"
              if $c->getErrorCode();
            $entry = $c->nextEntry();
        };
    }
    $c->close();
```

Для изменения элементов в `Net::LDAP` применяется другой подход. В нем все только что рассмотренные методы модуля `Mozilla::LDAP` объединены в одном «суперметоде» `modify()`. Параметры, передаваемые этому методу, и определяют его функциональность (табл. 6.6).

Таблица 6.6. Методы для изменения элементов в Net::LDAP

Параметр	Действие
`add => {$attrname => $attrvalue}`	Добавляет указанный элемент с заданным значением.
`add => {$attrname => [$attrvalue1, $attrvalue2...]}`	Добавляет указанный атрибут с заданным набором значений.
`delete => {$attrname => $attrvalue}`	Удаляет указанный атрибут с заданным значением.
`delete => {$attrname => []}` `delete => [$attrname1,$attrname2...]`	Удаляет атрибут или набор атрибутов независимо от их значений.
`replace => {$attrname => $attrvalue}`	Действует, как `add`, только заменяет текущее значение указанного атрибута. Если `$attrvalue` является ссылкой на пустой анонимный список (`[]`), метод становится синонимом для приведенной выше операции удаления.

Обязательно обратите внимание на знаки пунктуации в предыдущей таблице. Некоторые параметры принимают ссылки на анонимные хэши, другие – на анонимные массивы. Если их перепутать, трудностей не миновать.

Можно объединять несколько таких параметров в одном и том же вызове `modify()`, но это представляет собой потенциальную проблему. Когда `modify()` вызывается с набором параметров, например, так:

```
$c->modify($dn,replace => {'l' => "Medford"},
               add     => {'l' => "Boston"},
               add     => {'l' => "Cambridge"});
```

нет никаких гарантий, что указанные операции добавления будут выполняться после замены. Если необходимо, чтобы операции выполнялись в определенном порядке, можно применять синтаксис, подобный только что рассмотренному. Вместо использования набора дискрет-

ных параметров можно передать единственный массив, содержащий очередь команд. Вот как это работает: `modify()` принимает параметр `changes`, значение которого – список. Данный список считается набором пар. Первая половина пары – это операция, которую необходимо выполнить, вторая половина – ссылка на анонимный массив, содержащий данные для этой операции. Например, если мы хотим гарантировать, что операции из предыдущего фрагмента кода выполнятся в нужном порядке, то можем написать:

```
$c->modify($dn, changes =>
                  [ replace => ['l' => "Medford"],
                    add     => ['l' => "Boston"],
                    add     => ['l' => "Cambridge"]
                  ]);
```

Внимательно посмотрите на пунктуацию: она отличается от других параметров, которые приводились раньше.

Учитывая информацию, передаваемую функции `modify()`, можно переписать для `Net::LDAP` предыдущую программу, меняющую Бостон на Индиану:

```
use Net::LDAP;

$server   = $ARGV[0];
$port     = getservbyname("ldap","tcp") || "389";
$basedn   = "dc=ccs,dc=hogwarts,dc=edu";
$scope    = "sub";
$rootdn   = "cn=Manager, ou=Systems, dc=ccs, dc=hogwarts, dc=edu";
$pw       = "secret";

$c = new Net::LDAP($server, port => $port) or
  die "Невозможно соединиться с сервером $server: $@\n";
$c->bind(dn => $rootdn, password => $pw) or die "Ошибка при соединении:
$@\n";

$searchobj = $c->search(base  => $basedn, filter => "(l=Boston)",
                        scope => $scope,  attrs  => [''],
                        typesonly => 1);
die "Ошибка поиска: ".$searchobj->error()."\n" if ($searchobj->code());

if ($searchobj){
    @entries = $searchobj->entries;
    for (@entries){
        $res=$c->modify($_->dn(), # dn() получает DN-имя этого элемента
                   delete => {"l" => "Boston"},
                   add    => {"l" => "Indiana"});
        die "Невозможно изменить, код ошибки #".$res->code() if $res->code();
    }
}

$c->unbind();
```

Собираем все вместе

Теперь, когда нам известны все основные LDAP-функции, напишем несколько небольших сценариев для системного администрирования. Мы импортируем базу данных машин из главы 5 «Службы имен TCP/IP» на сервер LDAP и затем сгенерируем некую полезную информацию, основываясь на LDAP-запросах. Вот пара выдержек из этого простого файла (для того только, чтобы напомнить вам формат):

```
name: shimmer
address: 192.168.1.11
aliases: shim shimmy shimmydoodles
owner: David Davis
department: software
building: main
room: 909
manufacturer: Sun
model: Ultra60
-=-
name: bendir
address: 192.168.1.3
aliases: ben bendoodles
owner: Cindy Coltrane
department: IT
building: west
room: 143
manufacturer: Apple
model: 7500/100
-=-
```

Первое, что нужно сделать, – приготовить сервер каталогов для приема этих данных. Мы будем использовать нестандартные атрибуты, так что придется обновить схему сервера. Различные серверы выполняют это по-разному. Например, сервер каталогов Netscape имеет симпатичную графическую консоль Directory Server Console для подобных изменений. Другие серверы требуют внесения изменений в текстовые конфигурационные файлы. Работая с OpenLDAP, можно использовать нечто подобное в файле, включенном основным конфигурационным файлом для определения собственных пользовательских классов объектов для машины:

```
objectclass machine
        requires
                objectClass,
                cn
        allows
                address,
                aliases,
                owner,
                department,
```

```
building,
room,
manufacturer,
model
```

После того как сервер настроен нужным образом, можно подумать об импортировании данных. Один из вариантов – провести загрузку большой единой операцией с помощью LDIF. Если приведенный выше отрывок из базы данных напомнил вам о формате LDIF, значит, вы на правильном пути. Эта схожесть упрощает преобразование. Тем не менее, нужно остерегаться ловушек:

Продолжающиеся строки

В нашей базе данных нет элементов, значения которых занимали бы несколько строк, иначе следовало бы убедиться, что вывод удовлетворяет стандарту LDIF. Стандарт LDIF требует, чтобы все длинные строки начинались строго с одного пробела.

Разделители элементов

Между элементами в базе данных в качестве разделителя используется симпатичная последовательность `-=-`. Два разделителя строк (т. е. пустая строка) должны находиться между элементами LDIF, так что нужно будет удалить эту последовательность из вводимых данных.

Разделители атрибутов

В настоящее время в наших данных есть только один атрибут с несколькими значениями: aliases (псевдонимы). LDIF обрабатывает многозначные атрибуты, перечисляя каждое значение на отдельной строке. Если встретится несколько атрибутов, то понадобится специальный код, печатающий для каждого значения отдельную строку. Если бы не эта особенность, программа, преобразующая наш формат в LDIF, представляла бы собой одну строку кода на Perl.

Но даже и с этими ловушками программа преобразования на удивление проста:

```
$datafile = "database";
$recordsep = "-=-\n";
$suffix   = "ou=data, ou=systems, dc=ccs, dc=hogwarts, dc=edu";
$objectclass = <<EOC;
objectclass: top
objectclass: machine
EOC

open(DATA,$datafile) or die "Невозможно открыть $datafile:$!\n";

# Модули Perl не работают с этим, даже если в спецификации сказано обратное
# print "version: 1\n"; #
```

```
while (<DATA>) {
    # выводим заголовок для каждого элемента
    if (/name:\s*(.*)/){
        print "dn: cn=$1, $suffix\n";
        print $objectclass;
        print "cn: $1\n";
        next;
    }
    # обрабатываем многозначный атрибут aliases
    if (s/^aliases:\s*//){
        @aliases = split;
        foreach $name (@aliases){
            print "aliases: $name\n";
        }
        next;
    }
    # обрабатываем конец разделителя записей
    if ($_ eq $recordsep){
        print "\n";
        next;
    }
    # в противном случае просто печатаем найденный атрибут
    print;
}

close(DATA);
```

Если выполнить эту программу, то она выведет файл LDIF, выглядящий примерно так:

```
dn: cn=shimmer, ou=data, ou=systems, dc=ccs, dc=hogwarts, dc=edu
objectclass: top
objectclass: machine
cn: shimmer
address: 192.168.1.11
aliases: shim
aliases: shimmy
aliases: shimmydoodles
owner: David Davis
department: software
building: main
room: 909
manufacturer: Sun
model: Ultra60

dn: cn=bendir, ou=data, ou=systems, dc=ccs, dc=hogwarts, dc=edu
objectclass: top
objectclass: machine
cn: bendir
address: 192.168.1.3
aliases: ben
```

```
aliases: bendoodles
owner: Cindy Coltrane
department: IT
building: west
room: 143
manufacturer: Apple
model: 7500/100
...
```

Имея этот LDIF-файл, можно применять одну из программ, распространяемых с сервером, для загрузки этих данных на сервер. Например, *ldif2ldbm*, входящий в состав обоих серверов OpenLDAP и Netscape Directory Server, считывает LDIF-файл и напрямую импортирует его в формат сервера каталогов, избавляя от необходимости проходить через LDAP. Хотя эта программа используется только при неработающем сервере, она может обеспечить самый быстрый способ загрузки большого количества данных. Если нельзя остановить сервер, можно применить только что написанную на Perl программу для чтения LDIF-файлов и передать подобный файл на LDAP-сервер.

Добавим еще один способ: вот программа, в которой пропускается промежуточный шаг создания LDIF-файла, и наши данные напрямую импортируются на LDAP-сервер:

```
use Net::LDAP;
use Net::LDAP::Entry;

$datafile  = "database";
$recordsep = "-=-";
$server    = $ARGV[0];
$port      = getservbyname("ldap","tcp") || "389";
$suffix    = "ou=data, ou=systems, dc=ccs, dc=hogwarts, dc=edu";
$rootdn    = "cn=Manager, o=University of Michigan, c=US";
$pw        = "secret";

$c = new Net::LDAP($server,port => $port) or
  die "Невозможно соединиться с сервером $server: $@\n";
$c->bind(dn => $rootdn,password => $pw) or die "Ошибка при соединении: $@\n";

open(DATA,$datafile) or die "Невозможно открыть $datafile:$!\n";

while (<DATA>) {
    chomp;
    # в начале новой записи создаем новый экземпляр объекта
    if (/^name:\s*(.*)/){
        $dn="cn=$1, $suffix";
        $entry = new Net::LDAP::Entry;
        $entry->add("cn",$1);
        next;
    }
    # особый случай -- многозначные атрибуты
```

```
    if (s/^aliases:\s*//){
        $entry->add('aliases',[split()]);
        next;
    }

    # если дошли до конца записи, добавляем ее не сервер
    if ($_ eq $recordsep){
        $entry->add("objectclass",["top","machine"]);
        $entry->dn($dn);
        $res = $c->add($entry);
        warn "Ошибка добавления для " . $entry->dn() . ": код ошибки " .
              $res->code."\n"
          if $res->code();
        undef $entry;
        next;
    }

    # добавляем все остальные атрибуты
    $entry->add(split(':\s*')); # считаем, что у атрибута только одно
значение
}

close(DATA);
$c->unbind();
```

После того как данные были импортированы на сервер, можно приступить к довольно любопытным вещам. В следующих примерах мы будем поочередно обращаться к двум LDAP-модулям. Для краткости в каждом примере не будет повторяться заголовок, в котором устанавливаются конфигурационные переменные, и код для соединения с сервером.

Так что же можно сделать с данными, расположенными на сервере LDAP? Можно на лету создать файл узлов:

```
use Mozilla::LDAP;
...
$entry = $c->search($basedn,'one','(objectclass=machine)',0,
                    'cn','address','aliases');
die "Ошибка поиска:". $c->getErrorString()."\n" if $c->getErrorCode();

if ($entry){
    print "#\n\# host file - GENERATED BY $0\n
           # DO NOT EDIT BY HAND!\n#\n";
    while($entry){
        print $entry->{address}[0],"\t",
              $entry->{cn}[0]," ",
              join(' ',@{$entry->{aliases}}),"\n";
        $entry = $c->nextEntry();
    };
}
$c->close();
```

Вот что получается:

```
#
# host file - GENERATED BY ldap2hosts
# DO NOT EDIT BY HAND!
#
192.168.1.11    shimmer shim shimmy shimmydoodles
192.168.1.3     bendir ben bendoodles
192.168.1.12    sulawesi sula su-lee
192.168.1.55    sander sandy mickey mickeydoo
```

Можно найти имена всех машин, произведенных Apple:

```
use Net::LDAP;
...
$searchobj = $c->search(base  => $basedn,
                        filter => "(manufacturer=Apple)",
                        scope => 'one', attrs => ['cn']);
die "Ошибка поиска: ".$searchobj->error()."\n" if ($searchobj->code());

if ($searchobj){
    for ($searchobj->entries){
        print $_->get('cn'),"\n";
    }
}

$c->unbind();
```

Вот и результат:

```
bendir
sulawesi
```

Можно сгенерировать список владельцев машин:

```
use Mozilla::LDAP;
...
$entry = $c->search($basedn,'one','(objectclass=machine)',0,
                    'cn','owner');
die "Ошибка поиска:". $c->getErrorString()."\n" if $c->getErrorCode();

if ($entry){
    while($entry){
        push(@{$owners{$entry->{owner}[0]}},$entry->{cn}[0]);
        $entry = $c->nextEntry();
    };
}
$c->close();
for (sort keys %owners){
    print $_.":\t".join(' ',@{$owners{$_}})."\n";
}
```

Получилось так:

```
Alex Rollins:   sander
Cindy Coltrane: bendir
David Davis:    shimmer
Ellen Monk:     sulawesi
```

Заодно можно проверить, является ли владельцем машины пользователь с текущим идентификатором (псевдо-аутентификация):

```
use Mozilla::LDAP::Conn;
use Sys::Hostname;

$user = (getpwuid($<))[6];

$hostname = hostname;
$hostname =~ s/^([^.]+)\..*/$1/; # удаляем имя домена из имени узла
...
$entry = $c->search("cn=$hostname,$suffix",'base',"(owner=$user)",1,'');

if ($entry){
    print "Владелец ($user) зарегистрирован на машине $hostname.\n";
}
else {
    print "$user не является владельцем этой машины ($hostname)\n.";
}
$c->close();
```

Эти отрывки должны показать, как можно использовать доступ к LDAP из Perl для системного администрирования, и вдохновить вас на создание собственных программ. В следующем разделе эти идеи будут перенесены на новый уровень, что позволит нам увидеть целый интерфейс администрирования, построенный на основе LDAP.

ADSI (Интерфейсы служб активных каталогов)

В последнем разделе этой главы мы обсудим платформо-зависимый интерфейс службы каталогов, разработанный с учетом только что рассмотренного материала.

В Microsoft создали сложную службу каталогов, основанную на LDAP, под названием Active Directory для использования ее в сердцевине интерфейса администрирования Windows 2000. Служба Active Directory является репозиторием для всей важной конфигурационной информации (пользователи, группы, системные политики, поддержка установки программного обеспечения и т. д.), применяемой в сети машин с Windows 2000.

При разработке активных каталогов в Microsoft осознали, что для этой службы необходимо создать программный интерфейс более высокого уровня. Для этого был разработан интерфейс ADSI (Active Directory Service Interfaces, интерфейсы служб активных каталогов). Надо отдать должное разработчикам Microsoft, им удалось понять, что новый интерфейс нужно будет расширить и охватить такие области системного администрирования, как принтеры и службы NT. Подобный размах делает ADSI чрезвычайно полезным для тех, кто пишет сценарии, автоматизирующие выполнение задач системного администрирования. Перед тем как ознакомиться с его работой, приведем несколько основных концепций и терминов, которые нам понадобятся.

Основы ADSI

ADSI можно считать оболочкой вокруг произвольной службы каталогов, действующей в рамках ADSI. В среде интерфейса есть *провайдеры* (*providers*), которые представляют собой реализации ADSI для LDAP, WinNT 4.0 и Novell Directory Service. Говоря на «языке» ADSI, каждая из этих служб каталогов (WinNT не является службой каталогов) и домены данных (data domains) называются *пространством имен* (*namespaces*). ADSI предоставляет единый способ запроса и изменения данных, найденных в этих пространствах имен.

Чтобы понять ADSI, необходимо иметь представление об объектной модели компонентов COM (Component Object Model), на которой построен интерфейс ADSI. О модели COM существует много книг, но следует остановится на таких ключевых понятиях:

- Все, с чем работают в COM, – это *объекты* (*objects*).[1]
- Объекты имеют *интерфейсы* (*interfaces*), обеспечивающие набор *методов* (*methods*), применяемых для взаимодействия с этими объектами. Из Perl можно использовать методы, предоставляемые или наследуемые от интерфейса под названием *IDispatch*. К счастью, большинство методов ADSI, предоставляемых интерфейсами ADSI и их производными (например, IADsUser, IADsComputer, IADsPrintQueue), унаследованы от *IDispatch*.
- Значения, инкапсулируемые объектом, запрашиваемые и изменяемые посредством этих методов, называются *свойствами* (*properties*). В этой главе будут рассматриваться два типа значений: свойства, *определяемые интерфейсом* (*interface-defined properties*), и свойства, *определяемые схемой* (*schema-defined properties*). Иными словами,

[1] На самом деле COM – это протокол, используемый для связи с этими объектами как часть более крупного стандарта OLE (связывание и встраивание объектов). В данном разделе я постараюсь оградить читателя от трясины акронимов Microsoft, но те, кому нужны подробности, могут обратиться к ресурсам, доступным на *http://www.microsoft.com/com*.

первые будут определяться как часть интерфейса, а вторые – в объекте схемы. Подробнее об этом говорится чуть ниже. Если в данной главе не будут явно упоминаться «свойства схемы», значит, подразумевать следует свойства интерфейса.

Все это относится к стандартным понятиям объектно-ориентированного программирования. Сложности начинаются, когда сталкиваются терминологии ADSI/COM и других объектно-ориентированных миров, подобных LDAP.

Например, в ADSI рассматривается два различных типа объектов: *лист* (*leaf*) и *контейнер* (*container*). Объект-лист содержит данные; объект-контейнер содержит другие объекты, т. е. является для них *родительским* (*parent*). В LDAP самыми точными определениями для этих терминов были бы «элемент» и «точка ветвления». С одной стороны, мы говорим об объектах со свойствами, а с другой – об элементах с атрибутами. Как разобраться с подобными разногласиями, если учесть, что оба названия определяют одни и те же данные?

Вот как можно это понимать: в действительности, сервер LDAP обеспечивает доступ к дереву элементов и связанных с ними атрибутов. Когда интерфейс ADSI используется вместо LDAP для получения элемента дерева, ADSI вытягивает элемент из сервера LDAP, заворачивает его в несколько слоев блестящей оберточной бумаги и передает вам в качестве COM-объекта. Для получения содержимого этой посылки следует применять нужные методы, которые теперь называются «свойствами». Если внести изменения в свойства данного объекта, можно вернуть объект ADSI, чтобы последний распаковал информацию и передал ее обратно в дерево LDAP.

Вполне разумным выглядит вопрос: «А почему бы не обратиться напрямую к серверу LDAP?» На этот вопрос есть два хороших ответа: если мы знаем, как использовать ADSI для работы с одним типом служб каталогов, то мы знаем, как работать со всеми ними (или, по крайней мере, с теми, которые имеют ADSI-провайдеры). Второй ответ будет дан чуть позже, когда станет ясно, как можно упростить программирование служб каталогов при помощи инкапсуляции ADSI.

Необходимо ввести понятие *AdsPaths*, чтобы перейти к ADSI-программированию в Perl. ADsPaths предоставляет нам уникальный способ ссылаться на объекты из любого пространства имен. Они выглядят так:

```
<progID>:<path to object>
```

`<progID>` – это идентификатор провайдера (например WinNT или LDAP), а `<path to object>` – это специфичный для провайдеров способ поиска объекта в пространстве имен. Часть `<progID>` *чувствительна к регистру*. Если использовать `winnt`, `ldap` или `WINNT` вместо `WinNT` и `LDAP`, программы не будут работать.

Вот как выглядят примеры ADsPath из документации ADSI SDK:

```
WinNT://MyDomain/MyServer/User
WinNT://MyDomain/JohnSmith,user
LDAP://ldapsvr/CN=TopHat,DC=DEV,DC=MSFT,DC=COM,O=Internet
LDAP://MyDomain.microsoft.com/CN=TopH,DC=DEV,DC=MSFT,DC=COM,O=Internet
```

Это не совпадение, что они похожи на URL, т. к. и URL и ADsPath служат одним и тем же целям. Они оба пытаются обеспечить недвусмысленный способ сослаться на данные, предоставляемые различными службами данных. В случае с AdsPath из LDAP используется синтаксис LDAP URL из RFC, упомянутых в приложении B (RFC2255).

Более подробно AdsPath будет рассматриваться при обсуждении двух уже упомянутых пространств имен – *WinNT* и *LDAP*. Но сначала разберемся, как, в общих чертах, ADSI используется из Perl.

Использование ADSI из Perl

Семейство модулей `Win32::OLE`, поддерживаемое Жаном Дюбуа (Jan Dubois) и Гурусами Сарати (Gurusamy Sarathy), предоставляет мост от Perl к ADSI (который построен на COM как часть OLE). После загрузки основного модуля он используется для запроса ADSI-объектов:

```
use Win32::OLE;

$adsobj = Win32::OLE->GetObject($ADsPath) or
    die "Невозможно получить объект для $ADsPath\n";
```

`Win32::OLE->GetObject()` принимает *моникер* (*moniker*) OLE (уникальный идентификатор объекта, в данном случае это ADsPath) и возвращает объект ADSI. Этот вызов также обрабатывает процесс *связывания* (*binding*) с объектом, уже рассмотренный при обсуждении LDAP. По умолчанию связывание с объектом производится от имени пользователя, запускающего сценарий.

Этот совет может уберечь вас от испуга. Если выполнить эти две строчки кода в отладчике и изучить содержимое возвращаемой ссылки на объект, то можно увидеть нечто подобное:

```
DB<3> x $adsobj
0  Win32::OLE=HASH(0x10fe0d4)
     empty hash
```

Не волнуйтесь. `Win32::OLE` использует все могущество связанных переменных (tied variables). Кажущаяся пустой структура данных, как по волшебству, передаст информацию из объекта, если верно к ней обратиться.

Для доступа к значениям свойств интерфейса объекта ADSI используется ссылка на хэш:

Инструменты ADSI

Для использования материала из этой главы необходимо установить ADSI хотя бы на одной машине в сети. Эта машина может служить (через DCOM) ADSI шлюзом для остальных машин. Посетите сайт Тоби Эверета (Toby Everett), ссылка на который приведена ниже, чтобы узнать подробнее, как настроить ADSI для работы с DCOM.

Любая машина с Windows 2000 имеет встроенный в операционную систему интерфейс ADSI. Для всех остальных Win32-машин придется загрузить и установить бесплатный дистрибутив ADSI 2.5, находящийся в *http://www.microsoft.com/adsi*. По этой же ссылке вы найдете документацию по ADSI, включая *adsi25.chm*, – сжатую помощь в формате HTML, содержащую лучшую доступную документацию по ADSI.

Даже если вы работаете с Windows 2000, я советую загрузить ADSI SDK с сайта Microsoft по указанной ссылке, поскольку в него входит эта документация и удобный броузер объектов ADSI под названием *AdsVW*. SDK поставляется с примерами программирования ADSI на нескольких языках, включая Perl. К сожалению, примеры из текущего дистрибутива ADSI полагаются на устаревший модуль *OLE.pm*, так что, в лучшем случае, вы сможете получить несколько советов, но не надо использовать эти примеры в качестве стартовой точки.

Перед тем как начать писать программы, стоит загрузить броузер объектов ADSI Тоби Эверета (написанный на Perl) с *http://opensource.activestate.com/authors/tobyeverett*. Он научит вас перемещаться по пространствам имен ADSI. Обязательно посетите этот сайт, начиная карьеру программиста ADSI, поскольку он является одним из лучших доступных сайтов по применению ADSI из Perl.

```
$value = $adsobj->{key}
```

Например, если этот объект имеет свойство `Name`, определенное как часть его интерфейса (а так и есть), вы можете применить:

```
print $adsobj->{Name}."\n";
```

При помощи такой же записи можно присваивать значения свойствам интерфейсов:

```
$adsobj->{FullName}= "Oog";  # устанавливаем свойство в кэше
```

Свойства объекта ADSI хранятся в кэше (называемом *кэшем свойств* (*property cache*)). Первый запрос к свойствам объекта заполняет дан-

ный кэш. Последующие запросы к тем же свойствам позволяют получить информацию из этого кэша, а *не из службы каталогов*. Если вы хотите вручную заполнить кэш, можно вызвать методы `GetInfo()` или `GetInfoEx()` (расширенная версия `GetInfo()`) для данного экземпляра объекта, применяя синтаксис, который скоро будет рассмотрен.

Из-за того что первое считывание информации происходит автоматически, методы `GetInfo()` и `GetInfoEx()` часто остаются незамеченными. Существуют ситуации, когда эти методы следует употреблять, хотя в книге такие случаи рассматриваться не будут. Вот две подобные ситуации:

1. Некоторые свойства объектов можно получить, только явно вызвав `GetInfoEx()`. LDAP-провайдер Microsoft Exchange 5.5 представляет собой самый характерный пример, поскольку многие из его свойств не доступны, если не вызвать сначала `GetInfoEx()`. Детальную информацию об этой несовместимости можно найти на *http://opensource.activestate.com/authors/tobyeverett*.
2. Если несколько человек имеют право изменять в каталоге данные, то вызванный вами объект может быть кем-то преобразован, пока вы с ним работаете. Если это произойдет, данные в кэше свойств этого объекта устареют. `GetInfo()` и `GetInfoEx()` обновят этот кэш.

Для обновления службы каталогов и источников данных, предоставляемых через ADSI, после изменения объекта *нужно* вызвать специальный метод `SetInfo()`. `SetInfo()` сбрасывает изменения из кэша свойств в службу каталогов и источники данных. (Это должно напомнить вам о необходимости вызывать метод `update()` в `Mozilla::LDAP`. В данном случае идея та же.)

Вызывать методы экземпляра объекта ADSI не сложно:

```
$adsobj->Method($arguments...)
```

Поэтому, если бы мы изменили свойства объекта, как это предлагалось сделать в предыдущем предупреждении, то могли бы использовать такую строку сразу же после кода, вносящего изменения:

```
$adsobj->SetInfo();
```

В результате данные из кэша свойств помещаются обратно в службу каталогов или источник данных.

Вы будете часто использовать метод `Win32::OLE->LastError()` из модуля `Win32::OLE`. Он возвращает ошибку, полученную в результате последней операции OLE. Применение ключа *–w* с Perl (т. е. *perl –w script*) также приводит к подробным сообщениям о неудачных попытках OLE-операций. Зачастую эти сообщения об ошибках – единственная помощь, которая вам доступна, так что попытайтесь с толком ее использовать.

ADSI-код, который до сих пор рассматривался, выглядел как обычный код на Perl, поскольку внешне они похожи. Теперь перейдем к более сложным вопросам.

Работа с объектами контейнер/коллекция

Ранее в этом разделе уже упоминались два типа объектов ADSI: лист и контейнер. Объект-лист представляет собой только данные, тогда как контейнер (известный еще как коллекция – в терминах OLE/COM) содержит другие объекты. Еще одно отличие двух типов объектов в контексте ADSI состоит в том, что объект-лист не имеет дочерних объектов в иерархии, а у контейнеров такие объекты есть.

Объекты-контейнеры требуют специальной обработки, т. к. в большинстве случаев нас интересуют данные, инкапсулированные их дочерними объектами. Существует два способа обратиться к таким объектам из Perl. `Win32::OLE` имеет специальную функцию под названием `in()`, которая недоступна по умолчанию, если модуль загружается стандартным способом. Если необходимо получить к ней доступ, надо в начале программы использовать следующее:

```
use Win32::OLE 'in';
```

`in()` возвращает список ссылок на дочерние объекты, хранящиеся в этом контейнере. Это позволяет писать легко читаемые программы на Perl:

```
foreach $child (in $adsobj){
    print $child->{Name}
}
```

Другой путь заключается в том, чтобы загрузить один из полезных потомков `Win32::OLE` под названием `Win32::OLE::Enum`. `Win32::OLE::Enum->new()` создает объект-перечислитель из какого-либо объекта-контейнера:

```
use Win32::OLE::Enum;

$enobj = Win32::OLE::Enum->new($adsobj);
```

Для этого объекта можно вызвать несколько методов и получить дочерние объекты `$adsobj`. Подобный подход должен напомнить вам способ, применяемый в операциях поиска с `Mozilla::LDAP`; процесс тот же самый.

`$enobj->Next()` возвращает ссылку на следующий экземпляр дочернего объекта (или следующие X объектов, если задан необязательный параметр). `$enobj->All` возвращает список ссылок на экземпляры объектов. `Win32::OLE::Enum` предлагает несколько больше методов (подробнее о них сказано в документации), но этими вы будете пользоваться чаще всего.

Идентификация объекта-контейнера

Заранее нельзя узнать, является ли объект контейнером. Не существует способа из Perl «спросить» объект, не контейнер ли он. Максимум, что можно сделать, – попытаться создать объект-перечислитель и, если эта попытка не удастся, фиксировать данный результат. Вот короткий пример, который делает именно это:

```
use Win32::OLE;
use Win32::OLE::Enum;

eval {$enobj = Win32::OLE::Enum->new($adsobj)};
print "Объект " . ($@ ? "не " : "") . "является контейнером \n";
```

Второй способ – посмотреть на другие источники, описывающие этот объект. Все это плавно перетекает в третью сложность.

Как же узнать что-нибудь об объекте?

До сих пор мы избегали одного большого и, возможно, самого важного вопроса. Скоро нам придется работать с объектами из двух пространств имен. Уже понятно, как получить и установить свойства объектов и как вызвать методы для этих объектов, но все справедливо только в случае, если известны названия этих свойств и методов. Откуда берутся эти названия? Как их можно найти?

Нет единого места, в котором можно найти ответы на эти вопросы, но существует несколько источников, из которых можно почерпнуть нужную информацию для формирования практически всей картины. Первое место – это документация по ADSI, особенно та помощь, о которой говорилось во врезке «Инструменты для ADSI». В этом файле содержится огромное количество материала. Для ответа на наш вопрос о названиях свойств и методов нужно начать с *Active Directory Service Interfaces 2.5→ADSI Reference→ADSI System Providers*.

Иногда имена методов можно найти только в документации, но существует другой, более интересный подход для поиска названий свойств. Можно использовать метаданные, предоставляемые самим ADSI. Именно здесь на сцену выходят свойства схемы, о которых говорилось раньше.

Каждый объект ADSI имеет свойство под названием `Schema`, которое связывает ADsPath с его объектом схемы. В частности, следующий пример:

```
use Win32::OLE;

$ADsPath = "WinNT://BEESKNEES,computer";
$adsobj  = Win32::OLE->GetObject($ADsPath) or
    die "Невозможно получить объект для $ADsPath\n";
print "Это объект ".$adsobj->{Class}.", схема находится в:\n".
```

```
        $adsobj->{Schema},"\n";
```

выведет:

```
Это объект Computer, схема находится в: WinNT://DomainName/Schema/Computer
```

Значение `$adsobj->{Schema}` – это путь ADsPath к объекту, описывающему схему для объектов класса `Computer` в этом домене. Здесь мы используем термин «схема» в том же смысле, что и в разговоре про схемы LDAP. В LDAP схемы определяют, какие атрибуты могут и должны присутствовать в элементах определенных классов объектов. В ADSI схема содержит ту же информацию об объектах определенного класса и их свойства схемы.

При желании посмотреть на возможные имена атрибутов объекта следовало бы взглянуть на значения двух свойств объекта схемы: `MandatoryProperties` и `OptionalProperties`. Изменим предыдущий оператор `print`:

```
$schmobj = Win32::OLE->GetObject($adsobj->{Schema}) or
             die "Невозможно получить объект для $ADsPath\n";
print join("\n",@{$schmobj->{MandatoryProperties}},
                @{$schmobj->{OptionalProperties}}),"\n";
```

Тогда получится:

```
Owner
Division
OperatingSystem
OperatingSystemVersion
Processor
ProcessorCount
```

Теперь известны возможные имена свойств схемы в пространстве имен WinNT для объектов `Computer`. Отлично.

Свойства схемы получаются и устанавливаются несколько иначе, чем свойства интерфейсов. Свойства интерфейсов обрабатываются примерно так:

```
# получение и установка свойств ИНТЕРФЕЙСОВ
$value = $obj->{property};
$obj->{property} = $value;
```

Свойства схемы получаются и устанавливаются при помощи специальных методов:

```
# получение и установка свойств СХЕМЫ
$value = $obj->Get("property");
$obj->Put("property","value");
```

Все, что касается свойств интерфейсов, о чем говорилось до сих пор, остается справедливым и для свойств схемы (т. е. кэш свойств, `SetInfo()`,

и т. д.). Помимо необходимости применения специальных методов для получения и установки значений, единственное, что отличает данные свойства, – это их имена. Иногда один и тот же объект может иметь два различных имени для одного и того же свойства, одно для свойств интерфейса, другое для свойств схемы. Например, два этих свойства получают основные настройки для пользователя:

```
$len = $userobj->{PasswordMinimumLength};  # свойство интерфейса
$len = $userobj->Get("MinPasswordLength"); # то же самое свойство схемы
```

Наличие двух типов свойств обусловлено тем, что свойства интерфейса существуют в виде части модели COM. Разработчики, определяя интерфейс при создании программы, также определяют свойства интерфейса. Позже, если они хотят расширить набор свойств, им приходится изменять и COM-интерфейс, и любой код, использующий этот интерфейс. В ADSI разработчики могут изменить свойства схемы в провайдере без необходимости изменять лежащий в основе COM интерфейс этого провайдера. Очень важно разобраться с обоими типами свойств, т. к. иногда некоторые данные объекта доступны через свойства только одного типа.

На практике, если вы ищете только названия свойств интерфейса или схемы и не собираетесь писать программы для их поиска, я рекомендую использовать ADSI-броузер Тоби Эверета, о котором я упоминал ранее. Вот пример этого броузера в действии (рис. 6.2).

Как альтернативный вариант упомянем программу *ADSIDump* из каталога *General* примеров SDK, которая может вывести содержимое всего дерева ADSI.

Поиск

Эта последняя сложность, которую следует обсудить, перед тем как двигаться дальше. В разделе «LDAP: сложная служба каталогов» мы провели достаточно времени в разговорах о поиске в LDAP. Но в мире ADSI мы вряд ли услышим хоть слово по этому поводу. Все из-за того, что в Perl (и любом другом языке, в котором используется тот же OLE-интерфейс автоматизации) поиск с ADSI очень сложен; более того, поиск поддеревьев и поиск, в котором используются не самые простые фильтры, мучительно сложен. (Все остальное не так плохо.) Сложный поиск проблематичен, т. к. для его выполнения необходимо выйти за пределы ADSI и использовать совершенно иную методологию для получения данных (не говоря уже о том, что придется выучить новые акронимы от Microsoft).

Но тот, кто занимается системным администрированием, привык смеяться над сложностями, так что начнем с простого поиска, а потом перейдем к более сложным вопросам. Простой поиск, затрагивающий один объект (пространство `base`) или его непосредственных потомков

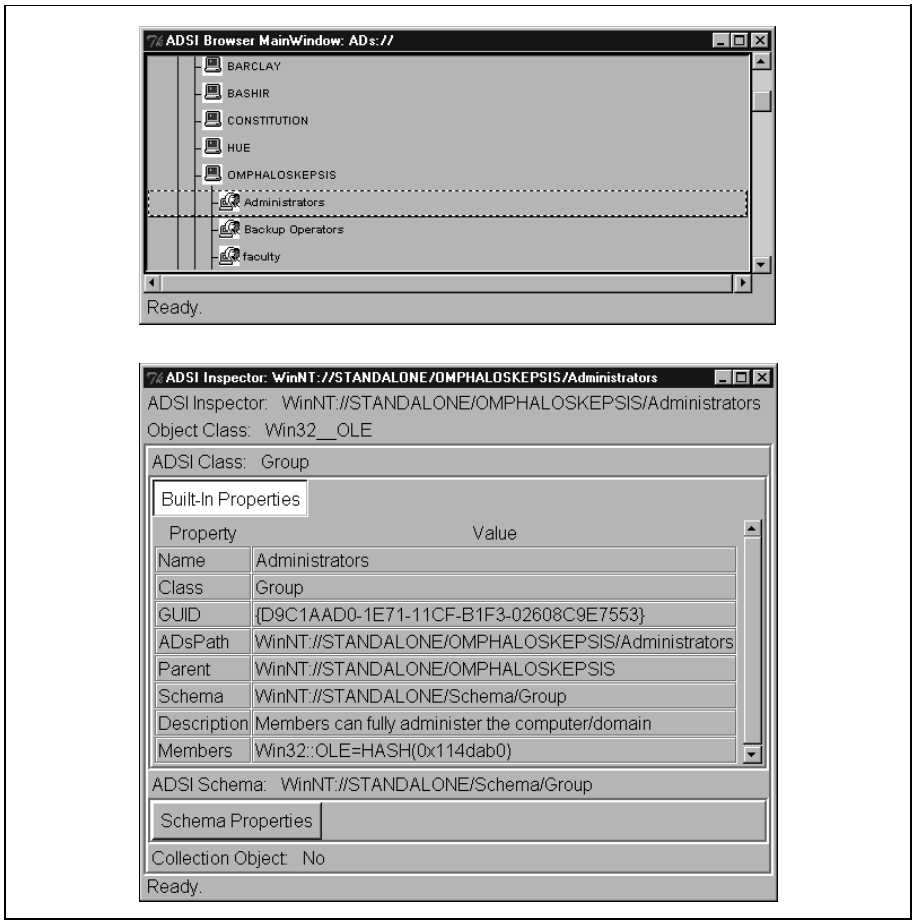

Рис. 6.2. *ADSI-броузер Эверета, отображающий объект Administrators*

(пространство `one`), можно выполнить вручную при помощи Perl. Сделать это можно так:

- Для одного объекта получите нужные свойства и используйте обычные операторы сравнения для определения соответствия:

```
if ($adsobj->{cn} eq "Mark Sausville" and $adsobj->{State} eq "CA"){...}
```

- Для поиска дочерних объектов примените технологии доступа к контейнерам, о которых говорилось раньше, а затем изучите каждый дочерний объект. Несколько примеров поиска такого типа будут рассмотрены очень скоро.

Для того чтобы выполнить более сложный поиск, затрагивающий, скажем, все дерево каталогов или поддерево, вам придется переключиться на использование другой технологии «промежуточного уров-

ня» под названием ADO (ActiveX Data Objects, объекты данных ActiveX). ADO предоставляет языкам сценариев интерфейс к базам уровня Microsoft OLE DB. OLE DB обеспечивает общий интерфейс, ориентированный на базы данных, к источникам данных, подобным реляционным базам данных и службам каталогов. В нашем случае ADO будет применяться для «разговора» с ADSI (который, в свою очередь, общается с самой службой каталогов). Поскольку ADO – это методология, ориентированная на базы данных, рассматриваемая программа предваряет материал об ODBC, о котором речь пойдет в главе 7.

ADO работает только с провайдером LDAP ADSI.
В пространстве имен WinNT она работать не будет.

ADO – это отдельная тема, которая лишь затрагивает службы каталогов, поэтому будет рассмотрен только один пример с короткими пояснениями, остальные примеры работают с ADSI. Дополнительную информацию об ADO можно найти на *http://www.microsoft.com/ado*.

Вот пример программы, выводящей имена всех групп, найденных в данном домене. Детальное обсуждение программы приведено ниже.

```
use Win32::OLE 'in';

# получаем объект ADO, устанавливаем провайдер, открываем соединение
$c = Win32::OLE->new("ADODB.Connection");
$c->{Provider} = "ADsDSOObject";
$c->Open("ADSI Provider");
die Win32::OLE->LastError() if Win32::OLE->LastError();

# подготавливаем и выполняем запрос
$ADsPath = "LDAP://ldapserver/dc=example,dc=com";
$rs = $c->Execute("<$ADsPath>;(objectClass=Group);Name;SubTree");
die Win32::OLE->LastError() if Win32::OLE->LastError();

until ($rs->EOF){
    print $rs->Fields(0)->{Value},"\n";
    $rs->MoveNext;
}

$rs->Close;
$c->Close;
```

Блок кода после загрузки модуля получает экземпляр объекта ADO Connection, устанавливает имя провайдера для этого экземпляра объекта, а затем просит его открыть соединение. Соединение открывается от имени пользователя, запускающего сценарий, хотя можно было установить другие свойства объекта, позволяющие изменить такое поведение.

Затем выполняется собственно поиск при помощи `Execute()`. Поиск можно осуществлять средствами одного из двух «диалектов»: SQL или ADSI.[1] Диалект ADSI, как видно из программы, использует командную строку, состоящую из четырех аргументов, каждый из которых разделен точкой с запятой.[2] Вот эти аргументы:

- ADsPath (в угловых скобках), определяющий сервер и базовое DN-имя для поиска.
- Фильтр поиска (применяется тот же синтаксис LDAP-фильтров, что упоминался раньше).
- Имя или имена (разделенные запятыми) возвращаемых свойств.
- Пространство поиска: либо Base, либо OneLevel, либо SubTree (в соответствии со стандартом LDAP).

`Execute()` возвращает ссылку на первый из объектов ADO `RecordSet`, получаемых в результате запроса. По очереди запрашивается каждый из объектов `RecordSet`, распаковываются объекты, которые в нем содержатся, и выводится свойство `Value`, возвращаемое методом `Fields()` для каждого из этих объектов. Свойство `Value` содержит значение, которое запрашивалось в командной строке (имя объекта `Group`). Вот как выглядит отрывок получаемых данных на машине с Windows 2000:

```
Administrators
Users
Guests
Backup Operators
Replicator
Server Operators
Account Operators
Print Operators
DHCP Users
DHCP Administrators
Domain Computers
Domain Controllers
Schema Admins
Enterprise Admins
Cert Publishers
Domain Admins
Domain Users
```

[1] Тем, кто знает SQL, первый диалект покажется проще. Диалект SQL предлагает несколько интересных возможностей. Например, MS SQL Server 7 можно настроить так, что он будет знать об ADSI-провайдерах, а не только об обычных базах данных. Это означает, что вы можете выполнять SQL-запросы, которые одновременно обращаются к объектам ActiveDirectory через ADSI.

[2] Будьте внимательны при использовании провайдера ADSI ADO: около символов точки с запятой не может быть никаких пробелов, иначе запрос выполняться не будет.

```
Domain Guests
Group Policy Admins
RAS and IAS Servers
DnsAdmins
DnsUpdateProxy
```

Выполнение распространенных задач при помощи пространства имен WinNT и LDAP

Теперь, когда мы разобрались со «списком» сложностей, можно перейти к выполнению некоторых распространенных задач системного администрирования, используя ADSI из Perl. Цель – дать понять, какие задачи можно решать при помощи представленной информации об ADSI. Затем рассмотреть и использовать код, который пригоден для написания собственных программ.

Для этих целей будет использоваться одно из двух пространств имен. Первое пространство – *WinNT*, которое предоставляет доступ к объектам Windows NT 4.0, таким как пользователи, группы, принтеры, службы и т. д.

Второе – это наш старый знакомый – *LDAP*. *LDAP* мы выбираем провайдером при переходе к Windows 2000 и ее службе Active Directory, основанной на LDAP. Большинство объектов *WinNT* также доступны через *LDAP*. Ведь даже в Windows 2000 существуют задачи, которые можно выполнить, только используя пространство имен WinNT (например, создание учетных записей на локальной машине).

Программы, работающие с этими различными пространствами имен, похожи друг на друга (в конце концов, частично в этом и заключается смысл применения ADSI), но необходимо обратить внимание на два важных различия. Во-первых, формат ADsPath немного отличается. В соответствии с ADSI SDK, ADsPath в WinNT может иметь следующий вид:

```
WinNT://DomainName[/ComputerName[/ObjectName[,className]]]]
WinNT://DomainName[/ObjectName[,className]]]
WinNT://ComputerName,computer]
WinNT:
```

ADsPath в LDAP выглядит так:

```
LDAP://HostName[:PortNumber][/DistinguishedName]
```

Обратите внимание, что при работе с NT 4 ADsPath в LDAP требует указывать имя сервера (в Windows 2000 это изменилось). Это означает, что пространство имен LDAP нельзя просмотреть с верхнего уровня, как пространство WinNT, т. к. необходимо указать начальный сервер. В пространстве имен WinNT любой может применить ADsPath или просто `WinNT:` для начала поиска в иерархии доменов.

Также обратите внимание, что свойства объектов в двух пространствах имен похожи, но не идентичны. Например, можно обратиться к одним и тем же объектам из обоих пространств имен WinNT и LDAP, но обратиться к некоторым свойствам Active Directory конкретного объекта пользователя можно только через пространство имен LDAP.

Особенно важно заметить различия между схемами в этих двух пространствах имен. Например, класс `User` для WinNT не имеет обязательных свойств, тогда как класс `User` в LDAP требует наличия свойств `cn` и `samAccountName` в каждом объекте пользователя.

Не забывая об этих различиях, посмотрим на сам код. В целях экономии места пропустим бо́льшую часть проверок ошибок, но рекомендуется запустить сценарий с ключом *-w* и добавить в текст программы примерно такие строки:

```
die "Ошибка OLE:".Win32::OLE->LastError() if Win32::OLE->LastError();
```

Работа с пользователями через ADSI

Для получения списка пользователей домена применяется следующее:

```
use Win32::OLE 'in';

$ADsPath = "WinNT://DomainName/PDCName,computer";
$c = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";
foreach $adsobj (in $c){
    print $adsobj->{Name},"\n" if ($adsobj->{Class} eq "User");
}
```

Для создания нового пользователя и установки его полного имени (свойство Full Name):

```
use Win32::OLE;

$ADsPath="WinNT://DomainName/ComputerName,computer";
$c = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

# создаем и возвращаем объект User
$u = $c->Create("user",$username);
$u->SetInfo();  # нужно создать пользователя, перед тем как менять значения

# в пространстве имен WinNT: пробел между "Full" и "Name" недопустим
$u->{FullName} = $fullname;
$su->SetInfo();
```

Если `ComputerName` – это первичный контроллер домена (Primary Domain Controller), то мы создали пользователя домена. Если нет, этот пользователь будет локальным для данной машины.

Эквивалентная программа создания глобального пользователя (при помощи LDAP нельзя создавать локальных пользователей) в Active Directory выглядит так:

```
use Win32::OLE;

$ADsPath = "LDAP://ldapserver,CN=Users,dc=example,dc=com";

$c = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

# создаем и возвращаем объект User
$u=$c->Create("user","cn=".$commonname);
$u->{samAccountName} = $username;
# нужно сначала создать пользователя в каталоге, а потом менять значения
$u->SetInfo();

# пробел между "Full" и "Name" требуется при работе с пространством имен
LDAP:
$u->{'Full Name'} = $fullname;
$u->SetInfo();
```

Для удаления пользователя нужно внести лишь небольшие изменения:

```
use Win32::OLE;

$ADsPath = "WinNT://DomainName/ComputerName,computer";
$c = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

# удаляем объект User, заметьте, мы в границах контейнера
$c->Delete("user",$username);
$u->SetInfo();
```

Изменить пароль пользователя можно при помощи единственного метода:

```
use Win32::OLE;

$ADsPath = "WinNT://DomainName/ComputerName/".$username;
$u = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

$u->ChangePasssword($oldpassword,$newpassword);
$u->SetInfo();
```

Работа с группами через ADSI

Для перечисления доступных групп достаточно лишь немного подправить программу, выводящую список пользователей. Меняется только такая строка:

```
print $adsobj->{Name},"\n" if ($adsobj->{Class} eq "Group");
```

Создание и удаление групп выполняется при помощи тех же методов `Create()` и `Delete()`, которые применялись для создания и удаления учетных записей. Единственное различие – первый аргумент нужно изменить на «group». Вот так:

```
$g = $c->Create("group",$groupname);
```

Для добавления пользователя в группу (определяемую при помощи `GroupName`) после ее создания используется следующее:

```
use Win32::OLE;

$ADsPath = "WinNT://DomainName/GroupName,group";

$g = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

# используется ADsPath для указанного объекта пользователя
$g->Add($userADsPath);
```

Здесь действуют те же правила относительно локальных пользователей и пользователей домена (глобальных), которые мы рассмотрели выше. Для того чтобы добавить пользователя домена в группу, `$user-ADsPath` должна указывать на пользователя на PDC для этого домена.

Для удаления пользователя из группы применяйте:

```
$c->Remove($userADsPath);
```

Работа с разделяемыми ресурсами через ADSI

Теперь займемся более интересными задачами ADSI, адресованными посвященным. Можно применять ADSI, чтобы предоставить в совместное пользование часть локального дискового пространства на машине:

```
use Win32::OLE;

$ADsPath = "WinNT://ComputerName/lanmanserver";

$c = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

$s = $c->Create("fileshare",$sharename);
$s->{path}        = 'C:\directory';
$s->{description} = "This is a Perl created share";
$s->SetInfo();
```

Разделяемые ресурсы можно удалять при помощи метода `Delete()`.

Перед тем как перейти к другим задачам, хочу воспользоваться случаем и напомнить вам о необходимости обратиться к документации SDK перед работой с каким-либо из этих ADSI-объектов. Кое-какие неожиданности могут оказаться полезными. Если вы заглянете в раздел *Acti-*

ve Directory Service Interfaces 2.5→ADSI Reference→ ADSI Interfaces→ Persistent Object Interfaces→ IADsFileShare файла помощи ADSI 2.5, то увидите, что объект `fileshare` имеет свойство `CurrentUserCount`, которое соответствует количеству пользователей, подсоединенных в настоящее время к разделяемому ресурсу. Этот нюанс может очень сильно пригодиться.

Работа с очередями и заданиями печати через ADSI

Вот как можно определить названия очередей на определенном сервере и модели принтеров, используемых для обслуживания этих очередей:

```
use Win32::OLE 'in';

$ADsPath="WinNT://DomainName/PrintServerName,computer";

$c = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

foreach $adsobj (in $c){
    print $adsobj->{Name}.":".$adsobj->{Model}."\n"
      if ($adsobj->{Class} eq "PrintQueue");
}
```

После того как стало известно название очереди печати, можно напрямую связаться с ней для запросов и управления:

```
use Win32::OLE 'in';

# таблица получена из раздела
# 'Active Directory Service Interfaces 2.5->ADSI Reference->
# ADSI Interfaces->Dynamic Object Interfaces->IADsPrintQueueOperations->
# IADsPrintQueueOperations Property Methods' (уф!) из ADSI 2.5 SDK

%status =
  (0x00000001 => 'PAUSED',             0x00000002 => 'PENDING_DELETION',
   0x00000003 => 'ERROR' ,             0x00000004 => 'PAPER_JAM',
   0x00000005 => 'PAPER_OUT',          0x00000006 => 'MANUAL_FEED',
   0x00000007 => 'PAPER_PROBLEM',      0x00000008 => 'OFFLINE',
   0x00000100 => 'IO_ACTIVE',          0x00000200 => 'BUSY',
   0x00000400 => 'PRINTING',           0x00000800 => 'OUTPUT_BIN_FULL',
   0x00001000 => 'NOT_AVAILABLE',      0x00002000 => 'WAITING',
   0x00004000 => 'PROCESSING',         0x00008000 => 'INITIALIZING',
   0x00010000 => 'WARMING_UP',         0x00020000 => 'TONER_LOW',
   0x00040000 => 'NO_TONER',           0x00080000 => 'PAGE_PUNT',
   0x00100000 => 'USER_INTERVENTION', 0x00200000 => 'OUT_OF_MEMORY',
   0x00400000 => 'DOOR_OPEN',          0x00800000 => 'SERVER_UNKNOWN',
   0x01000000 => 'POWER_SAVE');

$ADsPath = "WinNT://PrintServerName/PrintQueueName";

$p = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";
```

```
print "Состояние принтера " . $c->{Name} . " -- " .
   ((exists $p->{status}) ? $status{$c->{status}} : "NOT ACTIVE") . "\n";
```

Объект PrintQueue имеет несколько методов для контроля очереди печати: Pause(), Resume() и Purge(). Это позволяет управлять действиями самой очереди. А что если мы захотим изучить или обработать конкретные задачи из очереди?

Для того чтобы добраться до самих заданий, необходимо вызвать метод PrintJobs() объекта PrintQueue. PrintJobs() возвращает коллекцию, состоящую из объектов PrintJob, каждый из которых имеет ряд свойств и методов. Например, вот как можно показать список заданий из определенной очереди:

```
use Win32::OLE 'in';

# таблица получена из раздела
# 'Active Directory Service Interfaces 2.5->ADSI Reference->
# ADSI Interfaces->Dynamic Object Interfaces->IADsPrintJobOperations->
# IADsPrintJobOperations Property Methods' (двойное уф) в ADSI 2.5 SDK

%status = (0x00000001 => 'PAUSED',  0x00000002 => 'ERROR',
           0x00000004 => 'DELETING',0x00000010 => 'PRINTING',
           0x00000020 => 'OFFLINE', 0x00000040 => 'PAPEROUT',
           0x00000080 => 'PRINTED', 0x00000100 => 'DELETED');

$ADsPath = "WinNT://PrintServerName/PrintQueueName";

$p = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

$jobs = $p->PrintJobs();
foreach $job (in $jobs){
  print $job->{User} . "\t" . $job->{Description} . "\t" .
        $status{$job->{status}} . "\n";
}
```

Каждое задание можно приостановить (Pause()) и продолжить (Resume()).

Работа со службами NT/2000 через ADSI

В последнем наборе примеров рассмотрим, как находить, запускать и останавливать службы на машине с NT/2000. Как и другие примеры из этой главы, эти короткие программки необходимо запускать с достаточными привилегиями для осуществления выполняемых действий.

Для получения списка служб на машине и их состояний можно использовать такую программу:

```
use Win32::OLE 'in';
```

```
# эта таблица получена из раздела
# 'Active Directory Service Interfaces 2.5->ADSI Reference->
# ADSI Interfaces->Dynamic Object Interfaces->IADsServiceOperations->
# IADsServiceOperations Property Methods' ADSI 2.5 SDK

%status =
  (0x00000001 => 'STOPPED',          0x00000002 => 'START_PENDING',
   0x00000003 => 'STOP_PENDING',     0x00000004 => 'RUNNING',
   0x00000005 => 'CONTINUE_PENDING',0x00000006 => 'PAUSE_PENDING',
   0x00000007 => 'PAUSED',           0x00000008 => 'ERROR');

$ADsPath = "WinNT://DomainName/ComputerName,computer";

$c = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

foreach $adsobj (in $c){
  print $adsobj->{DisplayName} . ":" . $status{$adsobj->{status}} . "\n"
      if ($adsobj->{Class} eq "Service");
}
```

Для запуска, остановки, приостановки или продолжения работы службы вызываются очевидные методы (`Start()`, `Stop()` и т. д.). Вот как можно запустить службу Network Time на машине с Windows 2000, если ранее она была остановлена:

```
use Win32::OLE;

$ADsPath = "WinNT://DomainName/ComputerName/W32Time,service";

$s = Win32::OLE->GetObject($ADsPath) or die "Невозможно получить $ADsPath\n";

$s->Start();
# можно в этом месте проверять в цикле состояние до тех пор,
# пока служба не будет запущена
```

Во избежание потенциальных конфликтов имен пользователей и компьютеров, можно переписать предыдущий пример:

```
use Win32::OLE;

$d = Win32::OLE->GetObject("WinNT://Domain");
$c = $d->GetObject("Computer", $computername);
$s = $c->GetObject("Service", "W32Time");

$s->Start();
```

Для остановки службы нужно всего лишь изменить последнюю строчку на:

```
$s->Stop();
# можно в этом месте проверять в цикле состояние до тех пор,
# пока служба не будет остановлена
```

Эти примеры должны подсказать вам, какой контроль над системой можно получить при помощи ADSI из Perl. Службы каталогов и их интерфейсы могут быть весьма могущественной частью вашей компьютерной инфраструктуры.

Информация о модулях из этой главы

Название	Идентификатор на CPAN	Версия
`Net::Telnet`	JROGERS	3.01
`Net::Finger`	FIMM	1.05
`Net::Whois`	DHUDES	1.9
`Net::LDAP`	GBARR	0.20
`Mozilla::LDAP`	LEIFHED	1.4
`Sys::Hostname` (входит в состав Perl)		
`Win32::OLE` (входит в состав ActiveState Perl)	JDB	1.11

Рекомендуемая дополнительная литература

Finger

«RFC1288:The Finger User Information Protocol», D. Zimmerman, 1991.

WHOIS

ftp://sipb.mit.edu/pub/whois/whois-servers.list – это список наиболее крупных WHOIS-серверов.

«RFC954:NICNAME/WHOIS», K. Harrenstien, M. Stahl, and E. Feinler, 1985.

LDAP

«An Internet Approach to Directories», Netscape, 1997 – отличное введение в LDAP (*http://developer.netscape.com/docs/manuals/ldap/ldap.html*).

«An LDAP Roadmap & FAQ», Jeff Hodges, 1999 (*http://www.kingsmountain.com/ldapRoadmap.shtml*).

http://www.ogre.com/ldap/ и *http://www.linc-dev.com/* – домашние страницы соавторов PerLDAP.

http://www.openldap.org/ – свободно распространяемый LDAP-сервер, находится в стадии активной разработки.

http://www.umich.edu/~dirsvcs/ldap/index.html – домашняя страница «прародителя» служб каталогов OpenLDAP и Netscape. Некоторая документация представляет интерес до сих пор.

«Implementing LDAP», Mark Wilcox (Wrox Press, 1999).

«LDAP-HOWTO», Mark Grennan, 1999 (*http://www.grennan.com/ldap-HOWTO.html*).

«LDAP Overview Presentation», Bruce Greenblatt, 1999 (*http://www.directory-applications.com/presentation/*).

«LDAP:Programming Directory-Enabled Applications With Lightweight Directory Access Protocol», Tim Howes and Mark Smith (Macmillan Technical Publishing, 1997).

Netscape Directory Server Administrator's/Installation/Deployment Guides and SDK documentation (*http://developer.netscape.com/docs/manuals/directory.html*).

«RFC1823:The LDAP Application Program Interface», T. Howes, M. Smith, 1995.

«RFC2222:Simple Authentication and Security Layer (SASL)», J. Myers, 1997.

«RFC2251:Lightweight Directory Access Protocol (v3)», M. Wahl, T. Howes, S.Kille, 1997.

«RFC2252:Lightweight Directory Access Protocol (v3):Attribute Syntax Definitions», M.Wahl, A. Coulbeck, T. Howes, S. Kille, 1997.

«RFC2254:The String Representation of LDAP Search Filters», T. Howes, 1997.

«RFC2255:The LDAP URL Format», T. Howes, M. Smith, 1997.

«RFC2256:A Summary of the X.500(96) User Schema for use with LDAPv3», M. Wahl, 1997.

«The LDAP Data Interchange Format (LDIF)–Technical Specification» (в состоянии разработки), Gordon Good, 1999 (можно найти на *http://search.ietf.org/internet-drafts/draft-good-ldap-ldif-0X.txt*, где X – это номер текущей версии).

«Understanding and Deploying Ldap Directory Services», Tim Howes, Mark Smith, Gordon Good (Macmillan Technical Publishing, 1998).

«Understanding LDAP», Heinz Jonner, Larry Brown, Franz-Stefan Hinner, Wolfgang Reis, Johan Westman, 1998. Превосходное введение в LDAP (*http://www.redbooks.ibm.com/ abstracts/sg244986.html*).

ADSI

http://cwashington.netreach.net/ – еще один хороший сайт (посвящен не только Perl) по созданию сценариев для ADSI и других технологий от Microsoft.

http://www.microsoft.com/adsi – канонический источник информации по ADSI; обязательно загрузите отсюда ADSI SDK.

http://opensource.activestate.com/authors/tobyeverett – содержит коллекцию документации по использованию ADSI из Perl Тоби Эверета.

http://www.15seconds.com – еще один хороший сайт (посвящен не только Perl) по созданию сценариев для ADSI и других технологий от Microsoft.

«Windows 2000 Active Directory», by Alistair G. Lowe-Norris (O'Reilly, 1999).