

Jabberd2 – простой и нетребовательный к ресурсам XMPP-сервер

Часть 2. Управление базами данных BerkeleyDB и тонкая настройка Jabberd2

Михаил Кондрин

В первой части статьи [1] была рассмотрена установка и базовая настройка XMPP-сервера Jabberd2. Однако дальнейшая настройка и отладка потребуют «хирургического» вмешательства и непосредственного редактирования его баз данных.

Итак, сервер запущен, список пользователей создан, с помощью Jabber-клиента Pidgin

можно обмениваться сообщениями как с пользователями нашего сервера, так и с внешними серверами. Что даль-

ше? Как вы помните, одним из условий техзадания для нашего Jabber-сервера был отказ от использования специ-

ализированных серверов баз данных для хранения учётных записей пользователей и другой необходимой информации. Поддержка сервера баз данных требует дополнительных усилий, так что гораздо проще хранить информацию в дисковых файлах, тем более что в большей степени эта информация представляет интерес только для самого сервера Jabberd2, и у вас вряд ли появится необходимость делиться ею с каким-нибудь другим сервисом. Также для доступа к этим дисковым файлам была выбрана библиотека BerkeleyDB.

Обычно базы данных BerkeleyDB характеризуют как не нуждающиеся в администрировании. Это справедливо в том смысле, что приложение может само решать административные вопросы в более-менее автоматическом режиме в процессе выполнения. В большинстве случаев это даже позволяет восстанавливать базы данных BerkeleyDB после аварийного завершения. Вопрос же создания резервных копий (вторая головная боль системных администраторов) при этом решается созданием специализированных утилит, которые решают задачу резервирования для всего приложения целиком.

Но это в том случае, если разработчики позаботились о создании таких инструментов. К сожалению, подход авторов Jabberd2 в том, что касается баз данных BerkeleyDB, не слишком конструктивен, поскольку ничего подобного они предложить не могут. Поэтому если вы решите хранить данные Jabberd2 средствами BerkeleyDB, то вам придётся волей-неволей вникать во внутреннее устройство и методы работы с базами данных этого типа.

Есть, правда, и хорошая новость – это не слишком сложно.

От BSD 4.4 до Oracle

История BerkeleyDB, библиотеки, которая в настоящее время насчитывает десятки миллионов инсталляций, началась в 1992 году, когда молодые сотрудники университета Беркли Марго Зельцер (Margo Seltzer) и Майк Олсон (Mike Olson) под руководством Кейта Бостича (Keith Bostic) взялись за модификацию утилиты ndbm (которая использовалась для работы с базами данных в оперативной памяти) к ре-

лизу операционной системы BSD 4.4. Эта модификация оказалась удачной, и уже в 1996 году Кейт Бостич и Марго Зельцер основали семейный бизнес (к тому времени они поженились) – компанию Sleepycat, которая занималась продвижением BerkeleyDB.

Благодаря сотрудничеству с университетом Мичигана (где был реализован один из первых серверов LDAP) и с компанией Netscape первоначальный проект был существенно усовершенствован и приобрёл широкую популярность.

В 2006 году компания SleepyCat (к которой к тому времени присоединился и Майк Олсон) была куплена лидером на рынке баз данных – компанией Oracle. До этого долгое время BerkeleyDB выходила под двойной лицензией, поэтому на программистских форумах слухи об этой покупке вызвали беспокойство, что Oracle закроет код BerkeleyDB. Однако эти опасения не подтвердились, и за последующие два года вышло несколько новых релизов этой библиотеки, причем Oracle продолжает выполнять свое обещание оставить код этой библиотеки открытым и свободным.

Все трое сотрудников SleepyCat по-прежнему работают над своим продуктом, совмещая программистскую деятельность с преподавательской: Марго Зельцер одновременно является профессором Гарвардского университета.

Администрирование баз данных BerkeleyDB

Как вы уже знаете, BerkeleyDB – это библиотека, которая позволяет нескольким приложениям совместно работать как над дисковыми, так и оперативными (расположенными в памяти) базами данных. Раз речь идёт о совместной работе с дисковым файлом, то возникает вопрос контроля доступа к этому файлу, чтобы информация для каждого из активных приложений была консистентной и чтобы работа нескольких приложений в параллельном режиме не приводила к порче данных. Это одна из задач, которую BerkeleyDB решает с помощью «окружений» (environment).

Если вы заглянете в директорию /var/jabberd/db/, то обнаружите там два файла authreg.db и sm.db, один или не-

сколько файлов типа log.0000xx, а также несколько файлов вида __db.00*. И что же с ними делать?

Директория /var/jabberd/db/ – это домашняя директория environment. Файлы с расширением *.db – это база данных (authreg – регистрационная информация, а sm – пользовательские данные). Файлы log.* – это, так сказать, оперативные поправки, внесённые в базу данных, которые по мере заполнения сбрасываются в файлы *.db. А файлы __db.* – это отображение областей памяти в дисковые файлы, которые и составляют environment, поскольку реализуют совместную работу нескольких приложений над базой данных. Тут следует заметить, что Jabberd2 состоит из нескольких независимых процессов, так что для него разделение доступа к базе данных весьма актуально.

Вся информация хранится в файлах *.db и в одном из log-файлов, в том, что активен на текущий момент. Длина log-файла фиксирована, по умолчанию она задаётся во время компиляции и составляет примерно 10 Мб, и может варьироваться с помощью файла настроек DB_CONFIG (о нём ещё пойдёт речь дальше).

По мере заполнения log-файла создаётся новый, с порядковым номером на единицу больше. Поскольку информация из старых log-файлов уже сброшена в файлы *.db, то они, вообще говоря, не нужны, так что в этом случае необходимо будет позаботиться о ротации логов, что также может осуществляться автоматически приложением, если только его автор об этом позаботился. Как вы понимаете, Jabberd2 – это не тот случай.

Все файлы – бинарные, так что их просмотр в текстовом редакторе или командой less вам мало что даст. Для этого случая предусмотрены специальные утилиты, например, db_printlog, которая позволит посмотреть последние правки, внесённые в базу данных (или по крайней мере время, когда они сделаны, поскольку остальная информация не слишком вразумительна).

Какие именно из log-файлов активны в данный момент, можно узнать с помощью команды:

```
# db_archive -l
```

```
log.0000000001
```

и удалить лишние с помощью:

```
# db_archive -d
```

Также эта утилита позволяет выяснить, какие файлы данных содержатся в каталоге (если вы забыли об этом):

```
# db_archive -s
```

```
authreg.db
sm.db
```

Вывод этих двух команд указывает, какие файлы нужны в архивировании при создании резервных копий – всего три штуки: log.0000000001, authreg.db, sm.db. Файлы вида __db.* системно зависимы и поэтому непереносимы, однако они могут быть восстановлены при наличии правильно созданной резервной копии.

Проще всего при архивировании не копировать файлы вручную, а использовать готовую команду:

```
# db_hotbackup -v -c -h ./ -b /root/backup
```

```
db_hotbackup: hot backup started at Sun May 4 00:29:55 2008
db_hotbackup: ./: force checkpoint
db_hotbackup: ./: remove unnecessary log files
db_hotbackup: copying ./authreg.db to /root/backup/authreg.db
db_hotbackup: copying ./sm.db to /root/backup/sm.db
db_hotbackup: copying ./log.0000000001 to /root/backup/log.0000000001
db_hotbackup: lowest numbered log file copied: 1
db_hotbackup: /root/backup: run catastrophic recovery
db_hotbackup: /root/backup: remove unnecessary log files
db_hotbackup: hot backup completed at Sun May 4 00:29:57 2008
```

Ключ -h указывает домашнюю директорию environment, а ключ -b – директорию, куда осуществляется копирование. С помощью ключа -c перед копированием происходит сбрасывание последних изменений из log-файла в базы данных и удаление устаревших log-файлов. Кстати, принудительное сбрасывание данных можно осуществить с помощью команды:

```
# db_checkpoint -v -1
```

```
db_checkpoint: checkpoint begin: Sun May 4 00:32:31 2008
db_checkpoint: checkpoint complete: Sun May 4 00:32:31 2008
```

Только учтите, что на самом деле db_checkpoint – это не одноразовая операция, а демон, который, будучи запущенным без ключа -1, периодически проверяет состояние log-ов в текущей директории и по мере их заполнения или в определённые промежутки времени производит синхронизацию баз данных.

Содержимое директории с резервной копией сейчас выглядит таким образом:

```
# ls /root/backup
```

```
authreg.db log.0000000001 sm.db
```

В таком виде база данных не работоспособна. То есть после копирования необходимо будет её восстановить, предварительно переместив в ту же директорию конфигурационный файл DB_CONFIG (если вас не устраивают параметры по умолчанию) и запустив там команду:

```
# db_recover -c -v -h ./ -e
```

```
Finding last valid log LSN: file: 1 offset 522761
Recovery starting from [1][28]
Recovery complete at Sun May 4 00:49:52 2008
Maximum transaction ID 800003eb Recovery checkpoint [1][522761]
```

Здесь ключи запуска, помимо очевидного -v (verbose), означают, что команда делает восстановление после катастрофического завершения (-c) и восстанавливает окружение -e в текущей директории (-h ./). Последняя строка результата указывает, какой записи в log-файле соответствует точка синхронизации. С помощью db_printlog можно убедиться, что это предпоследняя запись в log-файле.

Теперь резервный каталог выглядит таким образом:

```
# ls ./
```

```
DB_CONFIG __db.002 __db.004 __db.006 log.0000000001
__db.001 __db.003 __db.005 authreg.db sm.db
```

Проверим результат:

```
# db_verify -h ./ authreg.db sm.db
```

Отсутствие сообщений об ошибках указывает, что копирование и восстановление базы данных прошло успешно.

По мере обновления программного обеспечения может возникнуть ситуация, что формат базы данных не соответствует новой версии библиотеки BerkeleyDB, которая была установлена на компьютере в процессе обновления. В этом случае нужно также обновить базы данных с помощью утилиты db_upgrade, соответствующей новой версии библиотеки:

```
# db_upgrade -v -h ./ authreg.db sm.db
```

```
db_upgrade: authreg.db upgraded successfully
db_upgrade: sm.db upgraded successfully
```

Последний вопрос, который обычно возникает у администраторов баз данных, – это оптимизация работы базы данных. Для BerkeleyDB решение этого вопроса сводится к написанию файла DB_CONFIG с параметрами environment, более отвечающими мощности и возможностям вашей системы.

С помощью перечисленных параметров можно управлять величиной дискового кэша и кэша в памяти, методом ведения логов и их размерами, механизмами блокировок и т. д.:

- set_cachesize
- set_data_dir
- set_flags DB_AUTO_COMMITIDB_CDB_ALLDBIDB_DIRECT_DBIDB_DIRECT_LOGIDB_DSUNYC_DBIDB_DSUNYC_LOGIDB_LOG_AUTOREMOVEIDB_LOG_INMEMORYIDB_NOLOCKINGIDB_MULTIVERSIONIDB_NOMMAPIIDB_NOPANICIDB_OVERWRITEIDB_PANIC_ENVIRONMENTIDB_REGION_INITIDB_TIME_NOTGRANTEDIDB_TXN_NOSUNYCIDB_TXN_SNAPSHOTIDB_TXN_WRITE_NOSUNYCIDB_YIELDCPU
- set_lg_bsize
- set_lg_dir

- set_lg_max
- set_lg_mode
- set_lg_regionmax
- set_lk_detect DB_LOCK_DEFAULTIDB_LOCK_EXPIREIDB_LOCK_MAXWRITEIDB_LOCK_MINLOCKSIDB_LOCK_MINWRITEIDB_LOCK_OLDESTIDB_LOCK_RANDOMIDB_LOCK_YOUNGEST
- set_lk_max_lockers
- set_lk_max_locks
- set_lk_max_objects
- set_mp_mmapsize
- set_shm_key
- set_thread_count
- set_timeout
- set_tmp_dir
- set_tx_max
- set_verbose
- mutex_set_align
- mutex_set_max
- set_tas_spins
- rep_set_config DB_REP_CONF_BULKIDB_REP_CONF_DELAYCLIENTIDB_REP_CONF_NOAUTOINITIDB_REP_CONF_NOWAIT

Более подробную информацию по каждому из параметров можно получить в документации по соответствующей C функции из BerkeleyDB API в каталоге docs/api_c, где файлы руководства для этого класса функций имеют приставку env_. Так что, скажем, справка по set_cachesize находится в файле env_set_cachesize.html. Большинство параметров принимают в качестве значений одно или несколько строчных или целочисленных значений, что можно узнать в документации. Для случаев, когда выбор ограничен набором элементов, варианты допустимых значений приведены ранее. Например, можно использовать такой DB_CONFIG-файл:

```
# one 0GB 2MB cache
set_cachesize 0 2097152 1

# Transaction Log settings
set_lg_max 2097152
set_lg_bsize 262144
set_flags DB_LOG_AUTOREMOVE
set_flags DB_DIRECT_LOG
set_flags DB_DIRECT_DB
```

который увеличивает размер кэша до 2 Мб (стандартных 256 Кб для современных программ обычно всегда оказывается мало, что приводит к час-

тым обращениям к файлу базы данных на диске), ограничивает размер дискового log-файла 2 Мб, а log-файла в оперативной памяти – 262 Кб.

Кроме того, с помощью флагов конфигурируется автоматическое удаление устаревших логов, а также отключается системная буферизация доступа к файлам баз данных и логов (поскольку они и так буферизуются средствами BerkeleyDB).

Проверить эффект от изменения параметров по умолчанию можно с помощью команды «db_stats -e», которая позволяет получить суммарную информацию о размерах кэша, размерах log-файлов, количества выполненных и незаконченных транзакций, установленных локах и репликациях (по отдельности эту информацию можно получить с помощью ключей -m, -l, -t, -c и -r соответственно). Также эта команда с помощью ключей -s и -d выдаёт минимальные сведения о состоянии баз данных, например, количестве записей в них.

Для полноты стоит упомянуть об уже встречавшейся в предыдущей части команде db_load. Хотя в предыдущей статье с помощью этой команды нам удалось создать и загрузить список пользователей authreg.db, однако для более сложных случаев создание текстового файла в формате, пригодном для загрузки в BerkeleyDB, оказывается ничем не легче, чем прямое редактирование баз данных с помощью программного интерфейса. В следующем разделе мы как раз и перейдём к программированию с помощью библиотеки BerkeleyDB.

В недрах sm.db

Иногда бывает нужно оперативно изменить данные в базах данных BerkeleyDB или посмотреть уже имеющуюся там информацию. Если для решения первой части задачи можно прибегнуть к команде db_load, то со второй частью не всё так просто. К сожалению универсального инструмента для этой цели нет и быть не может, что связано именно с присущими BerkeleyDB ограничениями. Но эта задача может быть решена с помощью написания скрипта для каждого конкретного случая, в данном случае нашей целью будет создание «официального утверждённого списка контактов»

(published roster), хранящегося на стороне сервера. Для этого нам придётся вручную отредактировать базу данных sm.db, которая содержит динамически изменяемые настройки Jabberd2 и используется, в основном, компонентом sm. В отличие от authreg.db её не нужно создавать до запуска сервера; создаётся она при регистрации первого пользователя, а затем пополняется по большей части в автоматическом режиме.

Попутно нам придётся немного глубже разобраться с устройством BerkeleyDB. В качестве рабочего инструмента будем использовать скриптовый язык TCL, который завоевал популярность благодаря простоте своего синтаксиса («все есть строка, а каждая инструкция начинается с имени процедуры, за которым следует список фактических параметров»).

Более того, BerkeleyDB имеет встроенный интерфейс под этот язык, наравне с C++ и Java, сборка которого активируется выбором параметра --enable-tcl при компиляции библиотеки. Более полное представление об архитектуре и программировании BerkeleyDB можно получить из книги [2], однако там в качестве рабочего языка при разборе примеров взят C++.

В качестве напоминания в одном абзаце будут перечислены основные элементы языка Tcl. Этого хватит, чтобы разобраться с приведёнными ниже скриптами.

Tcl – это фактически shell, у переменных типов нет, всё является строкой. Каждая команда Tcl представляет собой вызов функции, за которым следует список параметров. Например, создание собственной функции:

```
proc myfunction { z } {
    puts [set z]
}
```

представляет собой вызов функции proc, которой передано три параметра – имя определяемой функции myfunction, список формальных параметров, состоящий из одного элемента z и тела процедуры, в котором выдаётся на печать значение параметра. Фигурные скобки являются группирующими элементами, квадратные означают выполнение функции и подстановку результата. То есть в дан-

ном случае [set z] приводит к подстановке вместо переменной её значение. В качестве синтаксической глазури в tcl для той же самой цели используется более привычный \$. Команда set используется также для присваивания set z 1, например.

BerkeleyDB устроена более примитивно, чем более популярные иерархические или реляционные базы данных, поскольку все данные в BerkeleyDB хранятся в виде списков пар – «ключ-значение». Если проводить аналогию с реляционными базами данных, то мы имеем как бы таблицу с одной колонкой – первичным индексом, и второй – значением. В то же время именно простота устройства BerkeleyDB позволяет использовать её в качестве строительного блока для создания как иерархических (OpenLDAP), так и реляционных (MySQL) баз данных.

Вторым отличительным моментом BerkeleyDB является отсутствие типизации данных, так что в качестве ключей и их значений могут быть использованы любые типы данных, в том числе массивы, списки, если только приложения, подключённые к базе данных, умеют их правильно интерпретировать. С одной стороны, это даёт большой простор для разработчиков приложений, так как они не ограничены в выборе структур данных. А с другой – усложняет обмен данными между приложениями, поскольку для работы с базой данных приложение должно изначально точно знать используемые типы структур. Собственно поэтому и не существует универсальных утилит для доступа к базам данных BerkeleyDB. Вообще-то, и использо-

вание db_load применимо только для баз данных, содержащих только строковые значения, так что нам в некотором смысле повезло, что authreg.db не содержит записей другого типа.

Тип доступа и механизм индексации в базах данных можно выбрать при создании базы данных. Всего BerkeleyDB поддерживает 4 типа – Queue, Recno, Hash, BTree. Условно говоря, эти типы означают, каким именно образом BerkeleyDB индексирует и хранит базы данных. Все базы, используемые в Jabberd2, представляют собой хэши.

Любой файл базы данных BerkeleyDB (*.db) на самом деле может содержать несколько «таблиц»/списков. Чтобы посмотреть, что представляет собой база данных sm.db изнутри, откроем её с помощью скрипта list-sm.tcl:

```
load /usr/lib/libdb tcl-4.4.so
[berkdb env -home ./ -recover] close
set e [berkdb env -data_dir ./ -home ./]
set h [berkdb open -rdonly -env $e sm.db]
set c [$h cursor]
catch {
  for {set entry [$c get -first]} { $entry != {} } {
    { set entry [$c get -next]} {
      eval entry $entry
      puts [lindex $entry 0]
    }
  }
}
$c close
$h close
$e close
```

Идея проста. Вначале подгружается интерфейс доступа к библиотеке BerkeleyDB, а потом открывается environment,

«АКАДЕМИЯ КОРПОРАТИВНЫХ СИСТЕМ» Центр обучения и сертификации ИТ-специалистов

Лучшие условия обучения для системных администраторов

Дневная и вечерняя форма обучения
Тестирование по будням и в выходные дни

Подготовка к международной сертификации:
Microsoft, Cisco, Linux и др. Тестирование



Более 400 курсов по направлениям:

- ✓ **Системное администрирование**
Microsoft, Linux, FreeBSD, Sun Solaris
- ✓ **Администрирование и разработка баз данных**
на Microsoft SQL Server и Oracle
- ✓ **Работа с сетевым оборудованием Cisco**
- ✓ **Разработка и поддержка приложений** в среде Microsoft Visual Studio .Net
- ✓ **Технологии защиты информации в сетях** под управлением Microsoft, Linux, FreeBSD
- ✓ **Управление проектами** по методологиям PMI, IPMA с использованием Microsoft Project



В честь юбилея учебного центра для системных администраторов действует акция

«5 лет успеха – подарки всем!»

со специальными условиями и ценами

www.a-sys.ru

(495) 748-37-41

Мы находимся в центре Москвы: 5 минут пешком от м. «Пролетарская»

Рисунок 1. Электронная визитка

что (как вы помните) позволяет безопасным образом контролировать доступ к базам данных.

На самом деле окружение открывается дважды, первый раз в режиме восстановления, что является рекомендуемым способом обработки ошибок, вызванных возможным некорректным завершением работы при предыдущем открытии базы данных. Затем в уже созданном окружении открывается интересующая нас база данных, причём делается это в режиме read-only, поскольку для баз, содержащих несколько таблиц, иного выбора нет (позже вы увидите, как открыть доступ к отдельным таблицам на запись).

Далее в базе данных создаётся курсор-итератор, и в цикле последовательно перебираются все ключи, которые представляют собой имена списков. Вся логика работы с базой данных собрана в операторе перехвата ошибок catch, что гарантирует корректный переход к закрытию всех открытых дескрипторов точек доступа к файлам и базам данных при возникновении ошибок.

Запускается скрипт в директории, где находится файл sm.db, таким образом:

```
$ tclsh ../scripts/list-sm.tcl
```

```
active
disco-items
logout
motd-message
privacy-default
privacy-items
queue
roster-groups
roster-items
status
vacation-settings
vcard
```

Так выглядит список «таблиц» в только что созданной базе данных. Здесь и далее предполагается, что все скрипты собраны в отдельной директории /var/jabberd/scripts. Запомните также, что в скрипте нужно указывать ту версию библиотеки BerkeleyDB, с которой фактически собран Jabberd2. Как правило, в системе (для совместимости) присутствуют несколько версий этой библиотеки, так что тут нужно быть внимательным.

Попробуем теперь продвинуться дальше и прочитать из таблицы vcard список виртуальных визитных карточек с информацией о пользователях, которую они сочли нужным опубликовать в Jabberd. Впоследствии, например, можно будет сделать скрипт, который переносил бы эту информацию или её часть на LDAP-сервер.

Чтение таблицы осуществляется скриптом list-vcard.tcl:

```
source ../scripts/serialize.tcl
load /usr/lib/libdb tcl-4.4.so
[berkdb env -home ./ -recover] close
set e [berkdb env -home ./]
set h [berkdb open -env $e sm.db vcard]
set c [$h cursor]
catch {
    for {set entry [$c get -first]} { $entry != {} } {
        { set entry [$c get -next] } {
            eval set vcards $entry
            puts "->key(jid) = [lindex $vcards 0]"
            set card [deserialize [lindex $vcards 1]]
            foreach {field value} $card {
                puts "$field : $value"
            }
        }
    }
}
$c close
$h close
$e close
```

Как видите, по сравнению с предыдущим скриптом изменений немного. Во-первых, на этот раз открывается только одна из «таблиц» (vcard) базы данных sm.db, и, во-вторых, добавлена специальная обработка значений с помощью дополнительного файла serialize.tcl. Хотя как ключ, так и данные в BerkeleyDB могут быть любого типа, но разработчики Jabberd2, как правило, используют в качестве ключей строки, а для значений используют список-хэш. Формат списка выглядит так – вначале идёт имя элемента списка в виде строки, заканчивающейся нулём, затем идёт целое число, которое обозначает тип элемента, затем его значение, и так для каждого из элементов.

Функция deserialize как раз и служит для разбора записей, выполненных в таком формате, и возвращает их в виде парного Tcl-списка, который потом группируется попарно через разделитель-двоеточие и выводится на экран. Вот, например, моя электронная визитка (та же самая, что и на рис. 1):

```
$ tclsh ../scripts/list-vcard.tcl
->key(jid) = mike@hppi.troitsk.ru
nickname : mike
email : mkondrin@hppi.troitsk.ru
fn : Kondrin Mikhail
adr-region : MO
adr-country : RU
n-given : Mikhail
n-family : Kondrin
```

До настоящего ldif-формата, пригодного для загрузки на сервер LDAP, пока ещё далеко, но направление движение, по крайней мере, понятно.

Не стоит думать, что BerkeleyDB поддерживает только простые таблицы. Поскольку BerkeleyDB позволяет осуществлять объединение (join) по первичному и вторичному индексам, в нем можно хранить данные достаточно сложной структуры. Также замечательной чертой BerkeleyDB является наличие транзакций, что позволяет вносить изменения в несколько таблиц атомарным способом, в стиле «всё или ничего».

Как раз для решения поставленной вначале задачи — заполнение published-roster — нам и придётся столкнуться с транзакциями. В данном случае это будет скорее вынужденной мерой, поскольку в Jabberd2 доступ ко всем таблицам из sm.db возможен только в режиме транзакций. Чтобы Jabberd2 мог использовать созданную нами таблицу, в таком же режиме нам и следует открыть published-roster. Все элементы в таблице имеют ключ, состоящий из пустой строки, а в качестве значений используется хэш-список с полями ask from to (целочисленными) jid name group (строкового типа). Эта информация построчно заносится в текстовый файл published.txt, откуда она и будет впоследствии считываться скриптом. Например, в нашем случае можно предложить такой вариант этого файла:

```
root@myrealm.ru 0 0 0 Admin root
postmaster@myrealm.ru 0 0 0 Admin postmaster
mike@myrealm.ru 0 0 0 Colleagues mike
```

где учётные записи разнесены по двум группам — реальных пользователей (группа Colleagues) и административных/технических аккаунтов (Admin). Скрипт выглядит таким образом:

```
source ../scripts/serialize.tcl
load /usr/lib/libdb tcl-4.5.so
[berkdb env -home ./ -recover] close
set e [berkdb env -home ./]
set h [berkdb open -dup -hash -create -auto_commit -env $e
    $e sm.db published-roster]
set f [open published.txt]
set t [$e txn]
set err [ catch {
    foreach line [split [read $f] "\n"] {
        set a [ pub_roster format $k]
        $h put -txn $t "" $a
    }
}]
if { $err } {
    $t abort
} else {
    $t commit
}
close $f
$h sync
$h close
$e close
```

Открытие базы данных sm.db происходит в транзакционном режиме за счёт выставленного флага auto_commit, она имеет тип Hash и не требует уникальности индекса (-dup). Все изменения, вносимые в файл, обернуты в транзакцию, причём в отличие от предыдущих случаев вызов catch анализируется на предмет обнаружения ошибок, и (в зависимости от результата) транзакция либо откатывается, либо нормальным образом завершается после прочтения файла и заполнения published-roster. Затем база данных синхронизируется, т.е. все изменения, хранящиеся в кэше, в памяти, принудительно сбрасываются на диск. Сама процедура форматирования записей pub_roster_format для по-

мещения в базу данных выглядит довольно уродливо, и поэтому вынесена также в подгружаемый файл serialize.tcl. В итоге результат выглядит примерно так, как показано на рис. 2.

На этом задачу можно считать решённой. При входе на сервер каждый пользователь теперь должен видеть отдельные общие группы со списком «опубликованных» пользователей, которые администратор может пополнять с помощью скрипта, приведённого выше. Следует подчеркнуть, что использование именно локальной базы данных для этой цели связано с «техническим заданием», которое мы перед собой поставили. Если же, скажем, у вас уже есть готовый каталог LDAP с электронными визитными карточками сотрудников, то вам, скорее, имеет смысл использовать его в качестве хранилища v-cards. Jabberd2 достаточно гибок в этом отношении и позволяет выбрать тип хранилища информации для каждого из модулей индивидуально. Например, в конфигурационном файле sm.xml.dist из дистрибутива Jabberd2 показано, каким образом можно настроить хранение визиток и published roster в каталоге LDAP, отдельно от остальных данных, для которых используется сервер MySQL.

И в завершение небольшая ложка дёгтя. Как уже говорилось, основной недостаток Jabberd2 — это практически полное отсутствие документации и несовпадение уже имеющихся руководств с реальным положением дел. В эту же ловушку попал и я сам. Этот цикл статей был написан в расчёте на серию релизов Jabberd2 2.1.x, последний из которых (2.1.24.1) вышел в апреле этого года. Однако уже в следующем релизе (2.2.0) произошли изменения в архитектуре Jabberd2, в результате чего схема, приведенная в предыдущей статье, уже не соответствует действительности. Компонент resolver был упразднен, а вся его функциональность перенесена в компонент s2s. При этом единственный информативный блок <lookup> из его конфигурационного файла также перемещен в файл s2s.xml. В общем-то, для динамично меняющегося проекта такие резкие изменения не являются чем-то необычным, но хотелось бы, чтобы эти изменения хоть каким-то образом находили свое отражение в документации, хотя бы на уровне файлов README (где по-прежнему упоминается resolver, как один из независимых компонентов) и ChangeLog. К сожалению, этот призыв можно адресовать не только авторам Jabberd2, но и многим другим разработчикам свободного программного обеспечения. 🍷

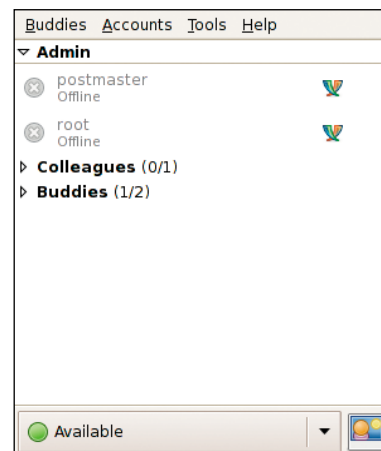


Рисунок 2. Открытая база данных

1. Кондрин М. Jabberd2 — простой и нетребовательный к ресурсам XMPP-сервер. //Системный администратор, №8, 2008 г. — С. 34-40.
2. Himanshu Yadava. The Berkeley DB Book. Apress, 2007.