NAME

DBI::ProxyServer - a server for the DBD::Proxy driver

SYNOPSIS

```
use DBI::ProxyServer;
DBI::ProxyServer::main(@ARGV);
```

DESCRIPTION

DBI::Proxy Server is a module for implementing a proxy for the DBI proxy driver, DBD::Proxy. It allows access to databases over the network if the DBMS does not offer networked operations. But the proxy server might be useful for you, even if you have a DBMS with integrated network functionality: It can be used as a DBI proxy in a firewalled environment.

DBI::ProxyServer runs as a daemon on the machine with the DBMS or on the firewall. The client connects to the agent using the DBI driver DBD::Proxy, thus in the exactly same way than using DBD::mysql, DBD::mSQL or any other DBI driver.

The agent is implemented as a RPC::PlServer application. Thus you have access to all the possibilities of this module, in particular encryption and a similar configuration file. DBI::ProxyServer adds the possibility of query restrictions: You can define a set of queries that a client may execute and restrict access to those. (Requires a DBI driver that supports parameter binding.) See "CONFIGURATION FILE".

The provided driver script, dbiproxy, may either be used as it is or used as the basis for a local version modified to meet your needs.

OPTIONS

When calling the *DBI::ProxyServer::main()* function, you supply an array of options. These options are parsed by the Getopt::Long module. The ProxyServer inherits all of RPC::PlServer's and hence Net::Daemon's options and option handling, in particular the ability to read options from either the command line or a config file. See RPC::PlServer. See Net::Daemon. Available options include

chroot (**--chroot=dir**)

(UNIX only) After doing a *bind()*, change root directory to the given directory by doing a *chroot()*. This is useful for security, but it restricts the environment a lot. For example, you need to load DBI drivers in the config file or you have to create hard links to Unix sockets, if your drivers are using them. For example, with MySQL, a config file might contain the following lines:

```
my $rootdir = '/var/dbiproxy';
my $unixsockdir = '/tmp';
my $unixsockfile = 'mysql.sock';
foreach $dir ($rootdir, "$rootdir$unixsockdir") {
mkdir 0755, $dir;
}
link("$unixsockdir/$unixsockfile",
"$rootdir$unixsockdir/$unixsockfile");
require DBD::mysql;

{
'chroot' => $rootdir,
...
}
```

If you don't know *chroot()*, think of an FTP server where you can see a certain directory tree only after logging in. See also the --group and --user options.

clients
An array ref with a list of clients. Clients are hash refs, the attributes *accept* (0 for denying access and 1 for permitting) and *mask*, a Perl regular expression for the clients IP number or its host name.

configfile (**--configfile=file**)
Config files are assumed to return a single hash ref that overrides the arguments of the new method. However, command line arguments in turn take precedence over the config file. See the ''CONFIGURATION FILE'' section below for details on the config file.

debug (**--debug**)
Turn debugging mode on. Mainly this asserts that logging messages of level ''debug'' are created.

facility (**--facility=mode**)
(UNIX only) Facility to use for Sys::Syslog. The default is **daemon**.

group (**--group=gid**)
After doing a *bind()*, change the real and effective GID to the given. This is useful, if you want your server to bind to a privileged port (<1024), but don't want the server to execute as root. See also the --user option.

GID's can be passed as group names or numeric values.

localaddr (**--localaddr=ip**)
By default a daemon is listening to any IP number that a machine has. This attribute allows one to restrict the server to the given IP number.

localport (**--localport=port**)
This attribute sets the port on which the daemon is listening. It must be given somehow, as there's no default.

logfile (**--logfile=file**)
Be default logging messages will be written to the syslog (Unix) or to the event log (Windows NT). On other operating systems you need to specify a log file. The special value ''STDERR'' forces logging to stderr. See Net::Daemon::Log for details.

mode (**--mode=modename**)
The server can run in three different modes, depending on the environment.

If you are running Perl 5.005 and did compile it for threads, then the server will create a new thread for each connection. The thread will execute the server's *Run()* method and then terminate. This mode is the default, you can force it with ''--mode=threads''.

If threads are not available, but you have a working *fork()*, then the server will behave similar by creating a new process for each connection. This mode will be used automatically in the absence of threads or if you use the ''--mode=fork'' option.

Finally there's a single-connection mode: If the server has accepted a connection, he will enter the *Run()* method. No other connections are accepted until the *Run()* method returns (if the client disconnects). This operation mode is useful if you have neither threads nor *fork()*, for example on the Macintosh. For debugging purposes you can force this mode with ''--mode=single''.

pidfile (**--pidfile=file**)
(UNIX only) If this option is present, a PID file will be created at the given location. Default is to not create a pidfile.

user (**--user=uid**)
After doing a *bind()*, change the real and effective UID to the given. This is useful, if you want your server to bind to a privileged port (<1024), but don't want the server to execute as root. See also the --group and the --chroot options.

UID's can be passed as group names or numeric values.

version (**--version**)
: Suppresses startup of the server; instead the version string will be printed and the program exits immediately.

SHUTDOWN

DBI::ProxyServer is built on RPC::PlServer which is, in turn, built on Net::Daemon.

You should refer to Net::Daemon for how to shutdown the server, except that you can't because it's not currently documented there (as of v0.43). The bottom-line is that it seems that there's no support for graceful shutdown.

CONFIGURATION FILE

The configuration file is just that of *RPC::PlServer* or *Net::Daemon* with some additional attributes in the client list.

The config file is a Perl script. At the top of the file you may include arbitrary Perl source, for example load drivers at the start (useful to enhance performance), prepare a chroot environment and so on.

The important thing is that you finally return a hash ref of option name/value pairs. The possible options are listed above.

All possibilities of Net::Daemon and RPC::PlServer apply, in particular

Host and/or User dependent access control
Host and/or User dependent encryption
Changing UID and/or GID after binding to the port
Running in a *chroot()* environment

Additionally the server offers you query restrictions. Suggest the following client list:

```
'clients' => [
{ 'mask' => 'admin\.company\.com$',
'accept' => 1,
'users' => [ 'root', 'wwwrun' ],
},
{
'mask' => 'admin\.company\.com$',
'accept' => 1,
'users' => [ 'root', 'wwwrun' ],
'sql' => {
'select' => 'SELECT * FROM foo',
'insert' => 'INSERT INTO foo VALUES (?, ?, ?)'
}
}
```

then only the users root and wwwrun may connect from admin.company.com, executing arbitrary queries, but only wwwrun may connect from other hosts and is restricted to

```
$sth->prepare("select");
```

or

```
$sth->prepare("insert");
```

which in fact are "SELECT * FROM foo" or "INSERT INTO foo VALUES (?, ?, ?)".

Proxyserver Configuration file (bigger example)

This section tells you how to restrict a DBI-Proxy: Not every user from every workstation shall be able to execute every query.

There is a perl program "dbiproxy" which runs on a machine which is able to connect to all the databases we wish to reach. All Perl-DBD-drivers must be installed on this machine. You can also reach databases for which drivers are not available on the machine where you run the program querying the database, e.g. ask MS-Access-database from Linux.

Create a configuration file “proxy_oracle.cfg” at the dbproxy-server:

```
{
# This shall run in a shell or a DOS-window
# facility => 'daemon',
pidfile => 'your_dbiproxy.pid',
logfile => 1,
debug => 0,
mode => 'single',
localport => '12400',

# Access control, the first match in this list wins!
# So the order is important
clients => [
# hint to organize:
# the most specialized rules for single machines/users are 1st
# then the denying rules
# then the rules about whole networks

# rule: internal_webserver
# desc: to get statistical information
{
# this IP-address only is meant
mask => '10\.95\.81\.243$',
# accept (not defer) connections like this
accept => 1,
# only users from this list
# are allowed to log on
users => [ 'informationdesk' ],
# only this statistical query is allowed
# to get results for a web-query
sql => {
alive => 'select count(*) from dual',
statistic_area => 'select count(*) from e01admin.e01e203 where geb_bezei like ?',
}
},

# rule: internal_bad_guy_1
{
mask => '10\.95\.81\.1$',
accept => 0,
},

# rule: employee_workplace
# desc: get detailed information
{
# any IP-address is meant here
mask => '10\.95\.81\.(\d+)$',
# accept (not defer) connections like this
accept => 1,
# only users from this list
# are allowed to log on
users => [ 'informationdesk', 'lippmann' ],
# all these queries are allowed:
sql => {
search_city => 'select ort_nr, plz, ort from e01admin.e01e200 where plz like ?',
search_area => 'select gebiettyp, geb_bezei from e01admin.e01e203 where geb_bezei like ? or
}
},

# rule: internal_bad_guy_2
# This does NOT work, because rule "employee_workplace" hits
# with its ip-address-mask of the whole network
{
# don't accept connection from this ip-address
mask => '10\.95\.81\.5$',
accept => 0,
}
]
}
```

Start the proxyserver like this:

```
rem well-set Oracle_home needed for Oracle
set ORACLE_HOME=d:\oracle\ora81
dbiproxy --configfile proxy_oracle.cfg
```

Testing the connection from a remote machine

Call a program "dbish" from your commandline. I take the machine from rule "internal_webserver"

```
dbish "dbi:Proxy:hostname=oracle.zdf;port=12400;dsn=dbi:Oracle:e01" informationdesk xxx
```

There will be a shell-prompt:

```
informationdesk@dbi...> alive

Current statement buffer (enter '/'...):
alive

informationdesk@dbi...> /
COUNT(*)
'1'
[1 rows of 1 fields returned]
```

Testing the connection with a perl-script

Create a perl-script like this:

```
# file: oratest.pl
# call me like this: perl oratest.pl user password

use strict;
use DBI;

my $user = shift || die "Usage: $0 user password";
my $pass = shift || die "Usage: $0 user password";
my $config = {
dsn_at_proxy => "dbi:Oracle:e01",
proxy => "hostname=oechsle.zdf;port=12400",
};
my $dsn = sprintf "dbi:Proxy:%s;dsn=%s",
$config->{proxy},
$config->{dsn_at_proxy};

my $dbh = DBI->connect( $dsn, $user, $pass )
|| die "connect did not work: $DBI::errstr";

my $sql = "search_city";
printf "%s\n%s\n%s\n", "="x40, $sql, "="x40;
my $cur = $dbh->prepare($sql);
$cur->bind_param(1,'905%');
&show_result ($cur);

my $sql = "search_area";
printf "%s\n%s\n%s\n", "="x40, $sql, "="x40;
my $cur = $dbh->prepare($sql);
$cur->bind_param(1,'Pfarr%');
$cur->bind_param(2,'Bronnamberg%');
&show_result ($cur);

my $sql = "statistic_area";
printf "%s\n%s\n%s\n", "="x40, $sql, "="x40;
my $cur = $dbh->prepare($sql);
$cur->bind_param(1,'Pfarr%');
&show_result ($cur);

$dbh->disconnect;
exit;


sub show_result {
my $cur = shift;
unless ($cur->execute()) {
print "Could not execute\n";
return;
}

my $rownum = 0;
while (my @row = $cur->fetchrow_array()) {
printf "Row is: %s\n", join(", ",@row);
if ($rownum++ > 5) {
print "... and so on\n";
last;
}
}
$cur->finish;
}
```

The result

```
C:\>perl oratest.pl informationdesk xxx
========================================
search_city
========================================
Row is: 3322, 9050, Chemnitz
Row is: 3678, 9051, Chemnitz
Row is: 10447, 9051, Chemnitz
Row is: 12128, 9051, Chemnitz
Row is: 10954, 90513, Zirndorf
Row is: 5808, 90513, Zirndorf
Row is: 5715, 90513, Zirndorf
... and so on
========================================
search_area
========================================
Row is: 101, Bronnamberg
Row is: 400, Pfarramt Zirndorf
Row is: 400, Pfarramt Rosstal
Row is: 400, Pfarramt Oberasbach
Row is: 401, Pfarramt Zirndorf
Row is: 401, Pfarramt Rosstal
========================================
statistic_area
========================================
DBD::Proxy::st execute failed: Server returned error: Failed to execute method CallMethod: U
Could not execute
```

How the configuration works

The most important section to control access to your dbi-proxy is "client=>" in the file "proxy_oracle.cfg":

Controlling which person at which machine is allowed to access

- "mask" is a perl regular expression against the plain ip-address of the machine which wishes to connect _or_ the reverse-lookup from a nameserver.
- "accept" tells the dbiproxy-server whether ip-adresse like in "mask" are allowed to connect or not (0/1)
- "users" is a reference to a list of usernames which must be matched, this is NOT a regular expression.

Controlling which SQL-statements are allowed

You can put every SQL-statement you like in simply omitting "sql => ...", but the more important thing is to restrict the connection so that only allowed queries are possible.

If you include an sql-section in your config-file like this:

```
sql => {
alive => 'select count(*) from dual',
statistic_area => 'select count(*) from e01admin.e01e203 where geb_bezei like ?',
}
```

The user is allowed to put two queries against the dbi-proxy. The queries are _not_ "select count(*)...", the queries are "alive" and "statistic_area"! These keywords are replaced by the real query. So you can run a query for "alive":

```
my $sql = "alive";
my $cur = $dbh->prepare($sql);
...
```

The flexibility is that you can put parameters in the where-part of the query so the query are not static. Simply replace a value in the where-part of the query through a question mark and bind it as a parameter to the query.

```
my $sql = "statistic_area";
my $cur = $dbh->prepare($sql);
$cur->bind_param(1,'905%');
# A second parameter would be called like this:
# $cur->bind_param(2,'98%');
```

The result is this query:

```
select count(*) from e01admin.e01e203
where geb_bezei like '905%'
```

Don't try to put parameters into the sql-query like this:

```
# Does not work like you think.
# Only the first word of the query is parsed,
# so it's changed to "statistic_area", the rest is omitted.
# You _have_ to work with $cur->bind_param.
my $sql = "statistic_area 905%";
my $cur = $dbh->prepare($sql);
...
```

Problems

- I don't know how to restrict users to special databases.
- I don't know how to pass query-parameters via dbish

SECURITY WARNING

RPC::PlServer used underneath is not secure due to serializing and deserializing data with Storable module. Use the proxy driver only in trusted environment.

AUTHOR

```
Copyright (c) 1997 Jochen Wiedmann
Am Eisteich 9
72555 Metzingen
Germany

Email: joe@ispsoft.de
Phone: +49 7123 14881
```

SEE ALSO

dbiproxy, DBD::Proxy, DBI, RPC::PlServer, RPC::PlClient, Net::Daemon, Net::Daemon::Log, Sys::Syslog, Win32::EventLog, syslog