DBIPlugin

The Persistent Database Connection Plugin

NOTE: This module requires SpamAssassin 3.1+

If you use this module and find it useful, please send parkerm ~~at~~ pobox ~~dot~~ com a note to let him know. I would like to eventually get this moved into the core SpamAssassin code and any feedback will help in that direction.

This plugin module provides persistent database connections. It uses the DBI interface in much the same way that Apache::DBI does, in fact a large portion of the inspiration comes from that module.

To use, all you need to do is load the plugin module, via loadplugin, and it will automatically step in and handle your database connections.

Due to the use of Apache::DBI code this module is placed under the Artistic License.

Code

Add the following to your local.cf or any .pre file:

```
loadplugin Mail::SpamAssassin::Plugin::DBI /path/to/Mail/SpamAssassin/Plugin/DBI.pm
```

DBI.pm:

```
=head1 NAME

Mail::SpamAssassin::Plugin::DBI - Provide persistent database connections

=head1 SYNOPSIS

  loadplugin Mail::SpamAssassin::Plugin::DBI

=head1 DESCRIPTION

This plugin module provides persistent database connections.  It uses the DBI interface
in much the same way that Apache::DBI does, in fact a large portion of the inspiration
comes from that module.

To use, all you need to do is load the plugin module, via loadplugin, and it will
automatically step in and handle your database connections.

=cut

package Mail::SpamAssassin::Plugin::DBI;

use strict;
use warnings;
use bytes;

use DBI ();

require_version DBI 1.00;

use Mail::SpamAssassin::Plugin;
use Mail::SpamAssassin::Logger;

use vars qw(@ISA);
@ISA = qw(Mail::SpamAssassin::Plugin);

my %CONNECTIONS;
my %ROLLBACKS;
my %PingTimeOut;
my %LastPingTime;
my $IN_CHILD_P = 0;

$DBI::connect_via = "Mail::SpamAssassin::Plugin::DBI::connect";

# constructor: register the eval rule
```

```
sub new {
  my $class = shift;
  my $mailsaobject = shift;

  # some boilerplate...
  $class = ref($class) || $class;
  my $self = $class->SUPER::new($mailsaobject);
  bless ($self, $class);

  return $self;
}

# supposed to be called in a startup script.
# stores the timeout per data_source for the ping function.
# use a DSN without attribute settings specified within !
# Note currently used, but preserved for future use.

sub setPingTimeOut {
  my ($class, $data_source, $timeout) = @_;
  if ($data_source =~ /dbi:\w+:.*/ and $timeout =~ /\-*\d+/) {
    $PingTimeOut{$data_source} = $timeout;
  }
}


# the connect method called from DBI::connect
# borrowed largely from Apache::DBI

sub connect {
  my $class = shift;
  unshift @_, $class if ref $class;
  my $drh    = shift;
  my @args   = map { defined $_ ? $_ : "" } @_;
  my $dsn    = "dbi:$drh->{Name}:$args[0]";

  my $connectkey = join $;, $args[0], $args[1], $args[2];

  # the hash-reference differs between calls even in the same
  # process, so de-reference the hash-reference
  if (3 == $#args and ref $args[3] eq "HASH") {
    # should we default to '__undef__' or something for undef values?
    map { $connectkey .= "$;$_=" .
            (defined $args[3]->{$_}
             ? $args[3]->{$_}
             : '')
          } sort keys %{$args[3]};
  } elsif (3 == $#args) {
    pop @args;
  }

  unless ($IN_CHILD_P) {
    dbg("dbiplugin: Creating uncached database handle to '$connectkey'");
    return $drh->connect(@args);
  }


  my $autocommit_p = ($connectkey =~ /AutoCommit[^\d]+0/) ? 1 : 0;
  if(!$ROLLBACKS{$connectkey} and $autocommit_p) {
    $ROLLBACKS{$connectkey} = 1;
  }

  # do we need to ping the database ?
  $PingTimeOut{$dsn}  = 0 unless $PingTimeOut{$dsn};
  $LastPingTime{$dsn} = 0 unless $LastPingTime{$dsn};
  my $now = time;
  my $needping = (($PingTimeOut{$dsn} == 0 or $PingTimeOut{$dsn} > 0)
                  and (($now - $LastPingTime{$dsn}) >= $PingTimeOut{$dsn})
                 ) ? 1 : 0;
  my $needpingtxt = $needping == 1 ? "yes" : "no";
  dbg("dbiplugin: $dsn need ping? $needpingtxt");
  $LastPingTime{$dsn} = $now;
```

```
  # check first if there is already a database-handle cached
  # if this is the case, possibly verify the database-handle
  # using the ping-method. Use eval for checking the connection
  # handle in order to avoid problems (dying inside ping) when
  # RaiseError being on and the handle is invalid.
  if ($CONNECTIONS{$connectkey} and (!$needping or eval{$CONNECTIONS{$connectkey}->ping})) {
    dbg("dbiplugin: Returning already connected database handle to '$connectkey'");
    return (bless $CONNECTIONS{$connectkey}, 'Mail::SpamAssassin::Plugin::DBI::db');
  }

  # either there is no database handle-cached or it is not valid,
  # so get a new database-handle and store it in the cache
  delete $CONNECTIONS{$connectkey};
  $CONNECTIONS{$connectkey} = $drh->connect(@args);
  unless ($CONNECTIONS{$connectkey}) {
    dbg("dbiplugin: Failed to create database handle for '$connectkey', returning undef");
    return undef;
  }

  # return the new database handle
  dbg("dbiplugin: Creating new database handle to '$connectkey'");
  return (bless $CONNECTIONS{$connectkey}, 'Mail::SpamAssassin::Plugin::DBI::db');
}


sub spamd_child_init {
  dbg("dbiplugin: In spamd_child_init");
  $IN_CHILD_P = 1;
  # Eventually, we can have a mechanism that will automatically connect to certain DSNs
  # at child init time, rather than when requested.
}

sub spamd_child_post_connection_close {
  dbg("dbiplugin: In spamd_child_post_connection_close");
  foreach my $connectkey (keys %CONNECTIONS) {
    my $dbh = $CONNECTIONS{$connectkey};
    if ($ROLLBACKS{$connectkey} and $dbh and $dbh->{Active}
        and !$dbh->{AutoCommit} and eval {$dbh->rollback}) {
      dbg("dbiplugin: signal_child_post_connection_close called rollback for $connectkey");
    }
    delete $ROLLBACKS{$connectkey};
  }
}

sub finish {
  dbg("dbiplugin: In finish()");
}

@Mail::SpamAssassin::Plugin::DBI::st::ISA = ('DBI::st');


# overload disconnect

{ package Mail::SpamAssassin::Plugin::DBI::db;
  no strict;
  @ISA=qw(DBI::db);
  use strict;
  use Mail::SpamAssassin::Logger;
  sub disconnect {
    dbg("dbiplugin: disconnect (overloaded) \n");
    1;
  };
}


1;
```

How To Use It

Just load the plugin and it will start caching your database connections right away.