POE

Colin Bradford

Milton Keynes Perl Mongers, October 2010

Agenda

- Example problem
- POE
- Event Programming
- Solving the problem with POE
    - Architecture
    - Connection to CouchDB
    - Web server

Example Problem

- Performance Dashboard
  - Web based display of current application performance
  - Expensive to calculate
  - Needs to be as close to real time as practicable
  - Needs to be as lightweight as possible

First Solution

- Calculate the results, and push them to a cache
- Clients poll for updates
- Problems:
  - Freshness of data limited to poll time
  - Reducing poll time increases network and server load

POE

- POE is a Perl framework for reactive systems, cooperative multitasking, and network applications
- POE manages Sessions, which do work
- Sessions send and receive events, and should not block
- Many POE Components on CPAN for useful tasks

Event Programming

- Write code in pieces eg
  - Start a connection to server
  - When connected, send message
  - When receive a message, process

Solving the problem with POE

- Store the data in a CouchDB instance
- Listen for changes in a POE Session
- A web server process accepts web requests, with the id of the data that the client has
  - If the request does not have the current data, send the latest data
  - If the request has the current data, wait for the data to change, and send the new data

Connection to CouchDB

```
# Create a couch listener
POE::Component::Client::TCP->new(
    RemoteAddress    => 'localhost',
    RemotePort       => 5984,
    Connected        => \&couch_handle_connected,
    ServerInput      => \&couch_handle_server_input,
    Alias            => 'changeswatcher',
    InlineStates     => {
        get_data     => \&couch_get_data,
    }
);
```

Couch Connected event

```
sub couch_handle_connected {
    my ($heap) = $_[HEAP];
    print "Connected\n";
    $heap->{server}->put("GET /reporting/_changes?
since=0&include_docs=true&feed=continuous&heartbeat=28000
HTTP/1.0\n\n");
    say "Connected to changes server";
}
```

Couch Data event

```
sub couch_handle_server_input {
    my ($kernel, $heap, $input) = @_[KERNEL, HEAP, ARG0];
    # If it's a changes line, process it.
    if ($input =~ /^{/) {
        my $data = JSON::XS::decode_json($input);
        # If it's the right document, store the document.
        if ($data->{doc}->{_id} eq 'overview') {
            $heap->{lastdata} = $data->{doc};
            # and tell all the watchers that the document changed
            send_message($kernel, "Document changed");
        }
    }
}

# {"seq":7,"id":"overview","changes":[{"rev":"7-
ca902d4a99283171d8a451241d032a56"}],"doc":{"_id":"overview","_rev":"7-
ca902d4a99283171d8a451241d032a56","warehouse":15000,"cs":50,"receipts":25000}} ]}
```

Data accessor

```
# Allow other sessions to get our data, without exposing it as a global
sub couch_get_data {
    my ($kernel, $heap) = @_[KERNEL, HEAP];
    my $data = $heap->{lastdata} || {};
    return $data;
};
```

Web Server

```
# Create Web service sessions
POE::Component::Server::TCP->new(
    Alias    => 'web_server',
    Port     => 8080,
    ClientFilter => 'POE::Filter::HTTPD',
    ClientInput => \&web_client_input,
    InlineStates => {
        respond => \&web_respond,
    },
);
```

Connection handler

```
sub web_client_input {
    my ($kernel, $heap, $request, $session) = @_[KERNEL, HEAP, ARG0,
SESSION];
    # If we've got a response already, it's an error, so send it
    if ($request->isa('HTTP::Response')) {
        $heap->{client}->put($request);
        $kernel->yield('shutdown');
        return;
    }
    my $uri = $request->uri();
    my ($filename) = ($uri =~ m/^\/file\/([a-z0-9\.]+)$/);
    $filename = $uri if $uri eq '/favicon.ico';
    if (defined $filename) {
        web_send_file($heap->{client}, $filename);
        $kernel->yield('shutdown');
        return;
    }
```

Connection handler (cont)

```
    # It’s a request for data, so see if client has the latest
    my ($update) = ($uri =~ m/^\/overview\/(\d+-[a-z0-9]+)$/);
    # Get the current data
    my $data = $kernel->call('changeswatcher', 'get_data');
    # If the current revision doesn't match, send immediately
    if (!defined $update or $update ne $data->{_rev}) {
        $kernel->yield('respond');
    }

    # Start watching the changeswatcher
    start_watcher($session->ID, 'respond');

    # And set a timeout, in case we don't get a change
    $kernel->delay('respond' => 28);
}
```

Web responder

```
sub web_respond {
    my ($kernel, $session, $heap) = @_[KERNEL, SESSION, HEAP];
    # Clear the reasons for the call
    stop_watcher($session->ID); # Stop watching for a response
    $kernel->delay('respond'); # Clear any timer that might be outstanding
    # Create a response with the appropriate data
    my $response = HTTP::Response->new(200);
    $response->push_header('Content-type' => 'application/json');
    my $text = JSON::XS::encode_json( $kernel->call('changeswatcher', 'get_data')
);
    $response->content($text);
    # Send the response, and close the session.  Sometimes we get a race, and the
client has already been replied to.
    if (defined $heap->{client}) {
        $heap->{client}->put($response);
    }
    $kernel->yield('shutdown');
}
```

Notifying changes

```
my %watcher;
sub start_watcher {
    my ($sessionID, $callback_event) = @_;
    $watcher{$sessionID} = $callback_event;
}

sub stop_watcher {
    my ($sessionID) = @_;
    delete $watcher{$sessionID};
}

sub send_message {
    my ($kernel, $message) = @_;
    foreach my $sessionID (keys %watcher) {
        $kernel->post($sessionID, $watcher{$sessionID}, $message);
    }
}
```

Thank you!