CS 683 Emerging Technologies
Fall Semester, 2008
Doc 7 Gen Server
Sept 23 2008

References

OTP Design Principles, Version 5.6.4, http://www.erlang.org/doc/design_principles/part_frame.html

gen_server docs, http://www.erlang.org/doc/man/gen_server.html

gen_server source code

Programming Erlang: Software for a Concurrent World, Armstrong, Chapter 16

Reading

Programming Erlang: Software for a Concurrent World, Armstrong, Chapter 16

Servers have common structure

```
start() -> spawn server process

rpc(Request) ->
    server ! {self(), Request},
    receive
        {server, Response} -> Response
    end.

loop() ->
    receive
        {From, Request} ->
            Response = compute response,
            From ! {server, Response},
            loop()
    end.
```

Toward a Generic Server

```
-module(serverA).
-export([start/1, rpc/1]).

start(Module) ->
    register(add_server, spawn(fun() -> loop(Module, Module:init()) end)).

rpc( Request) ->
    add_server ! {self(), Request},
    receive
        {add_server, Response} -> Response
    end.

loop( Module, State) ->
    receive
        {From, Request} ->
            {Response, NewState} = Module:handle(Request, State),
            From ! {add_server, Response},
            loop(Module, NewState)
    end.
```

Callback Module

```
-module (serverA_example).
-import (serverA, [rpc/1]).
-export ([init/0, handle/2,add/]).

add(Numbers) -> rpc({add, Numbers}).

init() -> 1.

handle({add, Numbers}, Count) ->
    Sum = lists:foldl(fun(X, PartialSum) -> X + PartialSum end,0, Numbers),
    {{sum,Sum,count,Count + 1}, Count + 1}.
```

```
43> serverA:start(serverA_example).
true
44> serverA_example:add([1, 2, 3]).
{sum,6,count,2}
45> serverA_example:add([1, 2, 3]).
{sum,6,count,3}
46> serverA_example:add([4, 5, 3]).
{sum,12,count,4}
```

Improvement 1

Don't Hard Code Server Name

```
-module(serverA).
-export([start/1, rpc/1]).

start(ServerName, Module) ->
    register(ServerName, spawn(fun() ->
          loop(ServerName, Module, Module:init()) end)).
```

Improvement 2

Transactions

```
rpc( Request) ->
    add_server ! {self(), Request},
    receive
        {add_server, crash} -> exit(rpc);
        {add_server, Response} -> Response
    end.
```

```
loop( Module, State) ->
    receive
            {From, Request} ->
          try Module:handle(Request, State) of
              {Response, NewState} ->
                  From ! {add_server, Response},
                  loop(Module, NewState)
          catch
              _:_ ->
                  From ! {add_server, crash},
                  loop(Module, State)
end.
```

Improvement 3

Hot Code Swapping

```
swap_code(NewModuleName) -> rpc({swap_code, NewModuleName

loop( Module, State) ->
    receive
      {From, {swap_code, NewModule}} ->
          From ! {add_server, ack},
          loop(NewModule, State);
            {From, Request} ->
          {Response, NewState} = Module:handle(Request, State),
          From ! {add_server, Response},
          loop(Module, NewState)
end.
```

Improvement 4

Let someone else write a generic server

gen_server

Using gen_server

Create client Interface

Implement callback routines

- init/1
- handle_call/3
- handle_cast/2
- handle_info/2
- termminate/2
- code_change/3

callback routines

init/1

Called when server started
Creates & returns state needed by server

handle_call/3

Called when server gets request
State passed as argument
Returns reply & server state

Example - Math Server

Interface

start()
    Start the server

stop()
    Stop the server

factorial(N)
    Compute N!

add(Numbers)
    Sum the numbers in the list Numbers

math_server.erl part 1

```
-module (math_server).

-behaviour (gen_server).

-export([start/0, stop/0, factorial/1, add/1]).
-export([init/1, handle_call/3, handle_cast/2, handle_info/2,
        terminate/2, code_change/3]).
```

```
start() -> gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).
stop()  -> gen_server:call(?MODULE, stop).

factorial(N)      -> gen_server:call(?MODULE, {factorial, N}).
add(Numbers)  -> gen_server:call(?MODULE, {add, Numbers}).
```


-behaviour (gen_server). is used by the compiler to make sure you implement all the methods required by gen_server. ?MODULE is a macro that expands to the modules name. We could use the module name (math_server) directly instead of the macro. This way there is only one place to change if we change the name of the module. Also start/stop can be copied to other servers as is.

gen_server:start_link

gen_server:start_link(Name, Mod, Args, Options)

Name ::= {local, atom()} | {global, atom()}

Mod ::= atom(), callback module implementing the 'real' server

Args ::= term(), init arguments (to Mod:init/1)

Options ::= [{timeout, Timeout} | {debug, [Flag]}]

Flag ::= trace | log | {logfile, File} | statistics | debug
(debug == log && statistics)

Returns: {ok, Pid} |
{error, {already_started, Pid}} |
{error, Reason}

math_server.erl part 2

Some Computation

```
compute_factorial(1) -> 1;
compute_factorial(N) when N < 1 ->
      throw({factorialNonPositiveArgument, N});
compute_factorial(N) ->
      N * compute_factorial(N - 1).

message_count({Pid,_MesageId}, Table) ->
      case ets:lookup(Table, Pid) of
            [] ->
                  ets:insert(Table, {Pid,1}),
                  1;
            [{Pid,OldCount}] ->
                  ets:insert(Table, {Pid,OldCount+1}),
                  OldCount+1
      end.
```

math_server.erl part 3

```
init([]) -> {ok, ets:new(?MODULE,[])}.

handle_call({factorial,N}, From, Table) ->
    {reply, {factorial,compute_factorial(N), messages, message_count(From,Table)}, Table};

handle_call({add, List}, From, Table) ->
    Sum = lists:foldl(fun(X, PartialSum) -> X + PartialSum end,0, List),
    {reply, {add,Sum, messages, message_count(From,Table)}, Table};

handle_call(stop,_From, Table) ->
    {stop, normal, server_stopped, Table}.
```

init(Args)

gen_server:start_link triggers a call to init

Args

Value from gen_server:start_link(Name, Mod, Args, Options)

Return Values

{ok, State}
{ok, State, Timeout}
ignore
{stop, Reason}

handle_call(Msg, {From, Tag}, State)

Msg

Message from client

From

Client process PID

Tag

Unique message ID

State

Server state from init/previous handle_call

gen_server:call triggers a call to handle_call

Return Values

{reply, Reply, State}

{reply, Reply, State, Timeout}

{noreply, State}

{noreply, State, Timeout}

{stop, Reason, Reply, State}

Reason = normal | shutdown | Term

terminate(State) is called

math_server.erl part 4

```
handle_cast(_Msg, State) -> {noreply, State}.

handle_info(_Info, State) -> {noreply, State}.

terminate(_Reason, State) ->
        ets:delete(State),
        ok.

code_change(_OldVsn, State, Extra) -> {ok, State}.
```

Unit Tests

```
-include_lib("eunit/include/eunit.hrl").
-module(math_server_tests).
-import(math_server).

all_tests_test_() ->
    {setup,
        fun test_setup/0,
        fun test_tear_down/1,
        fun server_tests/1
    }.

server_tests(_X) ->
      [ ?_assert({factorial, 6, messages, 1} == math_server:factorial(3)),
      ?_assert({factorial, 2, messages, 2} == math_server:factorial(2)),
      ?_assert({add, 6, messages, 3} == math_server:add([1, 2, 3]))].

test_setup() -> math_server:start().

test_tear_down(_X) -> math_server:stop().
```

handle_cast(Msg, State)

gen_server:cast triggers call to handle_cast

Used for one-way message from client to server

Return Values

{noreply, State}
{noreply, State, Timeout}
{stop, Reason, State}

Reason = normal | shutdown | Term
terminate(State) is called

terminate(Reason, State)

Called when

handle_call returns {stop, Reason, Reply, State}

handle_cast returns {stop, Reason, State}

gen_server is about to terminate

handle_info(Info, State)

Called when

Timeout occurs

gen_server receives exit message

code_change(OldVsn, State, Extra)

gen_server:system_code_change triggers call to code_change

Deals with updating/rollback of code changes

Timeout Example

```
start() -> gen_server:start_link({local, ?MODULE}, ?MODULE, [], []).
stop()  -> gen_server:call(?MODULE, stop).

add(Numbers)  -> gen_server:call(?MODULE, {add, Numbers}).

init([]) -> {ok, ets:new(?MODULE,[]), 1000}.

handle_call({add, List}, From, Table) ->
      Sum = lists:foldl(fun(X, PartialSum) -> X + PartialSum end,0, List),
      {reply, {add,Sum}, Table, 1000};

handle_call(stop,_From, Table) -> {stop, normal, server_stopped, Table}.

handle_cast(_Msg, State) -> {noreply, State}.
handle_info(_Info, State) -> {noreply, State, 1000}.
terminate(_Reason, State) ->  ok.
code_change(_OldVsn, State, Extra) -> {ok, State}.
```