céu-rabbitmq

carlos mattoso :: 03/novembro/2016

concepts

- connection
- channel
- exchange
- queue
- bindings
- subscription
- sample pub/sub

every C entity abstracted from the user (mostly)

concepts

- **connection**
- channel
- exchange
- queue
- bindings
- subscription
- sample: pub/sub

connection

```
data Connection with
     var& _amqp_connection_state_t_ state;
     var int channel_id;
end

data ConnectionContext with
     var _plain_string hostname = "localhost";
     var int          port       = 5672;
     var _plain_string vhost     = "/";
     var int          channel_max = 0;
     var int          frame_max   = 131072;
     var int          sasl_method = _AMQP_SASL_METHOD_PLAIN;
     var _plain_string user     = "guest";
     var _plain_string password = "guest";
end
```

connection

```
code/await New_Connection (var ConnectionContext ctx)
                            -> (var& Connection conn, event& void ok)
                                 -> FOREVER

var ConnectionContext ctx = val ConnectionContext(_,_,_,_,_,_,_,_);
// or: … = _; // -> even simpler

// defs...

spawn New_Connection(ctx) -> (&conn, &conn_ok);
await conn_ok;

// at this point we have a connection!
```

(

organisms are dead
:(

code/await

*“The **code/tight** and **code/await** declarations create new subprograms that can be invoked from arbitrary points in programs”*

code/await

```
code/tight Absolute (var int v) -> int do
// declares the prototype for "Absolute"
      if v > 0 then
      // implements the behavior
      escape  v;
      else
      escape -v;
      end
end

var int abs = call Absolute(-10);
      // invokes "Absolute" (yields 10)
```

```
code/await Hello_World (void) -> FOREVER
do
  every 1s do
  _printf("Hello World!\n");  // prints
"Hello World!" per 1s
      end
End

await Hello_World();
```

]

concepts

- connection
- **channel**
- exchange
- queue
- bindings
- subscription
- sample: pub/sub

channel

```
data Channel with
     var& Connection conn;
     var  int        id;
end
```

-> abstracts *id* handling from user!

```
code/await New_Channel (var& Connection conn)
                            -> (var& Channel ch)
                                 -> FOREVER
do
     // ...
end
```

channel

```
data Channel with
     var& Connection conn;
     var  int        id;
end
```

-> abstracts *id* handling from user!

```
code/await New_Channel (var& Connection conn)
                            -> (var& Channel ch)
                                 -> FOREVER
```

-> create a new channel

```
spawn New_Channel(&conn) -> (&ch);
```

channel: consumption

-> channel can consume by default

```
code/await New_Channel_Consume (var& Connection conn, pool&[] LowHandler handlers)
                                 -> (var& Channel channel)
                                      -> FOREVER
// ...

await async/thread (channel_, message) do
      loop do
            _amqp_consume_message(&&channel_.conn.state, &&message, 0, 0);
      end
      spawn LowHandler (message) in handlers;
end
```

channel: handlers

```
-> handlers take care of incoming messages
-> messages carry a tag that associates them with the inbound queue
-> tag taken when queue is created (hold on a bit)

code/await LowHandler (var _amqp_envelope_t message) -> void
do
  // takes care of destroying the message for the user
  do finalize with
      _amqp_destroy_envelope(&&message);
  end

  // user provided handler; it’s mandatory!
  await Handler (message);
end
```

channel: handlers

```
#include "amqp_base.ceu" // -> basic definitions for lib clients

code/await Handler (var _amqp_envelope_t message) -> void do
     _printf("Received message for ctag `%s`.\n",
             _stringify_bytes(message.consumer_tag));
     _amqp_dump(message.message.body.bytes,
                message.message.body.len);
end
```

concepts

- connection
- channel
- **exchange**
- queue
- bindings
- subscription
- sample: pub/sub

exchange

```
data Exchange with
      var _amqp_bytes_t name_bytes;
end

data ExchangeContext with
      var _plain_string name;
      var _plain_string type        = AMQ_DEFAULT_TYPE;
      var bool          passive     = false;
      var bool          durable     = false;
      var bool          auto_delete = true;
      var bool          internal    = false;
      var _amqp_table_t arguments   = _amqp_empty_table;
end

code/await New_Exchange (var& Channel channel, var ExchangeContext ctx)
                              -> (var& Exchange ex)
                              -> FOREVER
```

exchange: defaults

```
// All new default exchanges! Save time and money by using these.
var Exchange amq_default = val Exchange(_amqp_cstring_bytes(AMQ_DEFAULT)),
             amq_direct  = val Exchange(_amqp_cstring_bytes(AMQ_DIRECT)),
             amq_topic   = val Exchange(_amqp_cstring_bytes(AMQ_TOPIC)),
             amq_fanout  = val Exchange(_amqp_cstring_bytes(AMQ_FANOUT)),
             amq_headers = val Exchange(_amqp_cstring_bytes(AMQ_HEADERS));

// or create your own… note: default exchanges are not created
// also note: non-default exchanges are destroyed when code gets out of scope
spawn New_Exchange(&channel, ExchangeContext("test_exchange_topic", "topic",
                                             _,_,true,_,_amqp_empty_table))
                  -> (&e);
```

concepts

- connection
- channel
- exchange
- **queue**
- bindings
- subscription
- sample: pub/sub

queue

```
data Queue with
      var _amqp_bytes_t name_bytes;
end

data QueueContext with
      var _plain_string name;
      var bool           passive     = false;
      var bool           durable     = false;
      var bool           exclusive   = false;
      var bool           auto_delete = true;
      var _amqp_table_t arguments   = _amqp_empty_table;
end
```

-> note: queues are destroyed when code gets out of scope

```
code/await New_Queue (var& Channel channel, var QueueContext ctx)
                        -> (var& Queue q)
                              -> FOREVER
```

queue

```
spawn New_Queue(&channel,
QueueContext("test_queue",_,_,_,true,_amqp_empty_table))
      -> (&q);

spawn New_Queue(&channel, QueueContext("",_,_,_,true,_amqp_empty_table))
      -> (&q2);

spawn New_Queue(&channel, QueueContext(_,_,_,_,true,_amqp_empty_table))
      -> (&q_null_name);
```

concepts

- connection
- channel
- exchange
- queue
- **bindings**
- subscription
- sample: pub/sub

bind_queue

-> note: binding is destroyed when code gets out of scope

```
code/await Bind_Queue (var& Channel       channel,
                      var& Queue         queue,
                      var& Exchange      exchange,
                      var  _plain_string binding_key,
                      var  _amqp_table_t arguments)
                         -> void
```

-> note convenience of using default exchange

```
spawn Bind_Queue(&channel, &queue, &amq_direct, "baloney", _amqp_empty_table);
```

***Obs*: in AMQP every queue is automatically bound to the default exchange with a routing key equal to the former’s name**

concepts

- connection
- channel
- exchange
- queue
- bindings
- **subscription**
- sample: pub/sub

subscribe_queue

```
data SubscribeContext with
      var bool          no_local  = false;
      var bool          no_ack    = true;
      var bool          exclusive = false;
      var _amqp_table_t arguments = _amqp_empty_table;
end
```

-> note: unsubscribes when code gets out of scope
-> produces consumer tag, if none given
-> user should map queues and consumer-tags, so messages can be handled in *handler.ceu*

```
code/await Subscribe_Queue (var& Channel channel, var& Queue queue,
                            var SubscribeContext ctx,
                            var _plain_string consumer_tag)
                                -> (var& _amqp_bytes_t consumer_tag_bytes)
                                    -> FOREVER
```

concepts

- connection
- channel
- exchange
- queue
- bindings
- subscription
- **sample: pub/sub**

sample: publish in Céu

```
var& Connection conn;
event& void conn_ok;

spawn New_Connection(ConnectionContext(_,_,_,_,_,_,_,_))
     -> (&conn, &conn_ok);
await conn_ok;

var& Channel channel;
spawn New_Channel(&conn) -> (&channel);

spawn Publish(&channel, &amq_default,
              PublishContext("hello", "Hello from Ceu!",
                             _,_,default_props));
await 2s;
```

sample: publish in Ruby

```ruby
conn = Bunny.new(:automatically_recover => false)
conn.start

ch   = conn.create_channel
q    = ch.queue("hello")

ch.default_exchange.publish("Hello World!", :routing_key => q.name)
puts " [x] Sent 'Hello World!'"

conn.close
```

sample: publish in C

```
// Create connection
conn = amqp_new_connection();
socket = amqp_tcp_socket_new(conn);
status = amqp_socket_open(socket, hostname, port);
amqp_login(conn, "/", 0, 131072, 0, AMQP_SASL_METHOD_PLAIN, "guest",
"guest"), "Logging in");

// Create channel
amqp_channel_open(conn, 1);

// Publish
amqp_basic_publish(...);

// cleanup...
```

sample: consume in Céu

```
var& Connection conn;
event& void conn_ok;

spawn New_Connection(ConnectionContext(_,_,_,_,_,_,_,_)) -> (&conn, &conn_ok);
await conn_ok;

pool[] LowHandler handlers;
var& Channel channel;
spawn New_Channel_Consume(&conn, &handlers) -> (&channel);

var& Queue queue;
spawn New_Queue(&channel, QueueContext("hello",_,_,_,_,_amqp_empty_table)) -> (&queue);

var& _amqp_bytes_t consumer_tag;
spawn Subscribe_Queue(&channel, &queue, SubscribeContext(_,true,_,_amqp_empty_table),
null) -> (&consumer_tag);

await 2s;
```

sample: consume in Ruby

```ruby
conn = Bunny.new(:automatically_recover => false)
conn.start

ch   = conn.create_channel
q    = ch.queue("hello")

begin
  puts " [*] Waiting for messages. To exit press CTRL+C"
  q.subscribe(:block => true) do |delivery_info, properties, body|
    puts " [x] Received #{body}"
  end
rescue Interrupt => _
  conn.close

  exit(0)
end
```

sample: consume in C

```
conn = amqp_new_connection();
socket = amqp_tcp_socket_new(conn);
status = amqp_socket_open(socket, hostname, port);
amqp_login(conn, "/", 0, 131072, 0, AMQP_SASL_METHOD_PLAIN, "guest", "guest"), "Logging
in");

amqp_channel_open(conn, 1);

amqp_queue_declare_ok_t *r = amqp_queue_declare(conn, 1, amqp_empty_bytes, 0, 0, 0, 1,
amqp_empty_table);
queuename = amqp_bytes_malloc_dup(r->queue);
```

sample: consume in C

```
amqp_basic_consume(conn, 1, queuename, amqp_empty_bytes, 0, 1, 0, amqp_empty_table);

for (;;) {
  amqp_rpc_reply_t res;
  amqp_envelope_t envelope;

  amqp_maybe_release_buffers(conn);

  res = amqp_consume_message(conn, &envelope, NULL, 0);

  amqp_dump(envelope.message.body.bytes, envelope.message.body.len);

  amqp_destroy_envelope(&envelope);
}

// cleanup...
```