céu-rabbitmq

carlos mattoso :: 03/novembro/2016

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

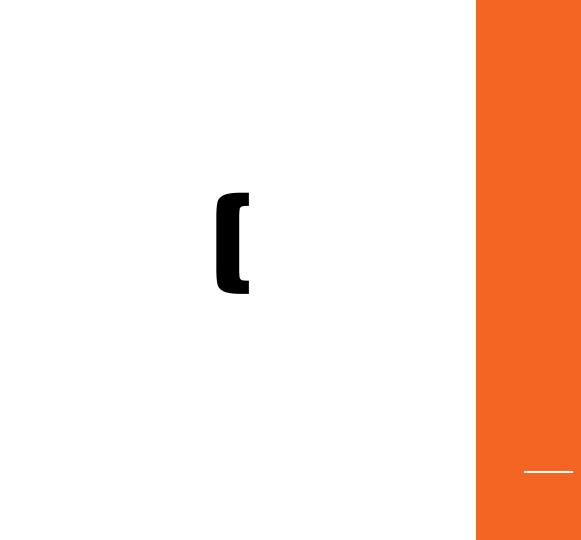
every C entity abstracted from the user (mostly)

- 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 frame max = 131072;
    var int
                 sasl_method = _AMQP_SASL_METHOD_PLAIN;
    var plain string user = "guest";
    var plain string password = "guest";
end
```

connection



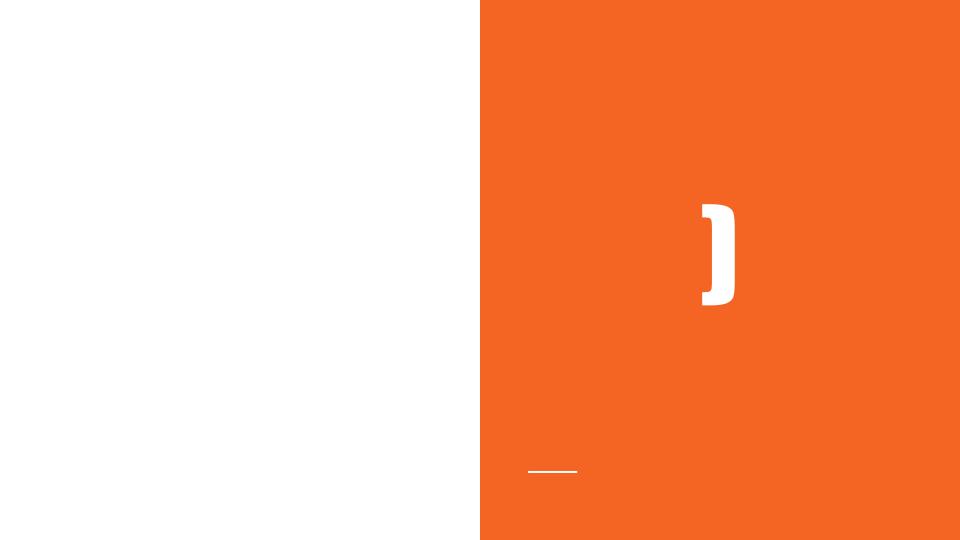
organisms are dead :(

code/await

"The **code/tight** and **code/await** declarations create new subprograms that can be <u>invoked</u> from arbitrary points in programs"

code/await

```
code/tight Absolute (var int v) -> int do
                                               code/await Hello_World (void) -> FOREVER
// declares the prototype for "Absolute"
                                               do
      if v > 0 then
                                                 every 1s do
      // implements the behavior
                                                _printf("Hello World!\n"); // prints
      escape v;
                                               "Hello World!" per 1s
      else
                                                     end
                                               End
      escape -v;
      end
end
                                               await Hello World();
var int abs = call Absolute(-10);
      // invokes "Absolute" (yields 10)
```



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

channel

channel

channel: consumption

-> channel can consume by default

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

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

exchange

```
data Exchange with
    var amap 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 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

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

queue

```
data Queue with
    var amap 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 _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& Oueue a)
                         -> 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);
```

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

bind_queue

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

subscribe_queue

```
data SubscribeContext with
     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
```

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

sample: publish in Céu

sample: publish in 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& Oueue 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

```
conn = Bunny.new(:automatically recover => false)
conn.start
ch = conn.create channel
    = 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

```
amap_basic_consume(conn, 1, queuename, amap_empty_bytes, 0, 1, 0, amap_empty_table);
for (;;) {
    amap_rpc_reply_t res;
    amap_envelope_t envelope;

    amap_maybe_release_buffers(conn);

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

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

    amap_destroy_envelope(&envelope);
}
// cleanup...
```