About evented-spec

Evented-spec is a set of helpers to help you test your asynchronous code.

EventMachine/Cool.io-based code, including asynchronous AMQP library is notoriously difficult to test. To the point that many people recommend using either mocks or synchronous libraries instead of EM-based libraries in unit tests. This is not always an option, however — sometimes your code just has to run inside the event loop, and you want to test a real thing, not just mocks.

em-spec gem made it easier to write evented specs, but it had several drawbacks. First, it is not easy to manage both EM.run and AMQP.start loops at the same time. Second, AMQP is not properly stopped and deactivated upon exceptions and timeouts, resulting in state leak between examples and multiple mystereous failures. amqp-spec, and, subsequently, evented-spec add more helpers to keep your specs from being bloated.

Usage

To get started with evented-spec you need to include one of the helper modules in your example groups, e.g.:

```
describe "eventmachine-based client" do
  include EventedSpec::SpecHelper
  it "should allow you to start a reactor" do
    em do
      EventMachine.reactor_running?.should be_true
      done
    end
  end
  context "nested contexts" do
    it "don't require another include" do
      em do
        EventMachine.add_timer(0.1) { @timer_run = true }
        done(0.3)
      end
      @timer_run.should be_true
    end
  end
end
```

Particular modules and methods are explained below.

#done

We have no means to know when your work with reactor is finished, so whatever it is you need to call done at some point. It optionally accepts a timeout and a block that is executed right before event reactor loop is stopped. If you don't call done, your specs are going to fail by timeout.

EventedSpec::SpecHelper

`EventedSpec::SpecHelper` is for semi-manual managing of reactor life-cycle. It includes three helpers: for EventMachine, Coolio and AMQP.

`em` stands for EventMachine. It takes a block, which is run after reactor starts.

`amqp` stands for AMQP. It takes a block, which is run after amqp connects with broker using given or default options.

`coolio` stands for cool.io. It takes a block, which is run after reactor starts.

All three accept a hash of options. Look into method documentation to learn more.

EventedSpec::EMSpec, EventedSpec::AMQPSpec, EventedSpec::CoolioSpec

`EventedSpec::EMSpec` wraps every example in em block, so it might save you a couple of lines per example. `EventedSpec::AMQPSpec` wraps every example in amqp block.

Also note that every example group including `EMSpec` or `AMQPSpec` automatically includes `SpecHelper`.

Example:

```
describe "eventmachine specs" do
  include EventedSpec::EMSpec
  it "should run in a reactor" do
    EventMachine.reactor_running?.should be_true
    done # don't forget to finish your specs properly!
  end
end
```

default_options, default_timeout

You can also pass some default options to specs (like amqp settings), they're specific to domain you're using evented-spec in.

`default_timeout` sets time (in seconds) for specs to time out

```
describe "using default_timeout" do
  include EventedSpec::SpecHelper
  default_timeout 0.5
  it "should prevent specs from hanging up" do
    em do
      1.should == 1 # this spec is going to fail with timeout error because #done is not called
    end
  end
end
```

Hooks

There are 6 hooks available to evented specs:

- `em_before` — launches after reactor started, before example runs

- `em_after` — launches right before reactor is stopped, after example runs
- `amqp_before` — launches after amqp connects, before example runs
- `amqp_after` — launches before amqp disconnects, after example runs
- `coolio_before` — launches after Cool.io starts, before example runs
- `coolio_after` — launches before Cool.io stops, after example runs

So, the order of hooks for an AMQP spec is as follows: `before(:all)`, `before(:each)`, `em_before`, `amqp_before`, example, `amqp_after`, `em_after`, `after(:each)`, `after(:all)`

```
describe "using amqp hooks" do
  include EventedSpec::AMQPSpec
  default_timeout 0.5
  amqp_before do
    AMQP.connection.should_not be_nil
  end
  let(:data) { "Test string" }
  it "should do something useful" do
    AMQP::Channel.new do |channel, _|
      exchange = channel.direct("amqp-test-exchange")
      queue = channel.queue("amqp-test-queue").bind(exchange)
      queue.subscribe do |hdr, msg|
        hdr.should be_an AMQP::Header
        msg.should == data
        done { queue.unsubscribe; queue.delete }
      end
      EM.add_timer(0.2) do
        exchange.publish data
      end
    end
  end
end
```

Words of warning on blocking the reactor

Evented specs are currently run inside of reactor thread. What this effectively means is that you **should not block** during spec execution.

For example, the following **will not** work:

```
describe "using amqp" do
  include EventedSpec::AMQPSpec
  it "should do something useful" do
    channel = AMQP::Channel.new
    sleep 0.2                    # voila, you're blocking the reactor
    channel.should be_open       # no, it should not
    done
  end
end
```

What you **should** do instead is use callbacks:

```
describe "using amqp" do
  include EventedSpec::AMQPSpec
  it "should do something useful" do
    AMQP::Channel.new do |channel, _|
      channel.should be_open
      done
```

```
    end
  end
end
```

You can also use #delayed helper method to maintain order of execution when callbacks are not an option.

```
describe "using amqp" do
  include EventedSpec::AMQPSpec
  it "should do something useful" do
    channel = AMQP::Channel.new
    @stage = 0
    delayed(0.2) {
      channel.should be_open
      @stage = 1
    }
    delayed(0.3) {
      @stage.should == 1
      done
    }
    delayed(0.4) {
      # this block is never going to be executed
      raise "Help me!"
    }
  end
end
```

I have an existing reactor running in separate thread, amqp specs won't work for me what should I do?

Unfortunately, right now there aren't many remedies to your problem, besides stopping the event loop in before(:all) hook like this:

```
describe "Example" do
  before(:all) { EM.stop_event_loop; sleep(0.1) }
  after(:all) { do_something_to_restart_the_eventmachine }
  include EventedSpec::SpecHelper
  it "should do something" do
    em { done }
  end
end
```

Reason is simple: if we don't restart event loop every spec example, all kinds of state leaks may occur: stale timers, delayed exceptions, weirdest errors and even segfaults. It isn't impossible but it certainly is very invasive.

Compatibility

EventedSpec is tested with RSpec >= 2.5.0, Cool.io ~> 1.0.0, EventMachine >= 0.12.10, and AMQP >= 0.8.0. Running it with RSpec 1.3 and/or AMQP 0.7.0 is not unheard of, although not tested in all its entirety.

See also

You can see evented-spec in use in spec suites for our amqp gems, amq-client and amqp.

Links

- cool.io
- amqp-spec
- eventmachine
- amqp
- amq-client

Generated on Sat Feb 22 04:33:32 2014 by yard 0.8.7.3 (ruby-2.0.0).