Middleware as Code with mruby

Details of mruby usage in production

GMOペパボ株式会社

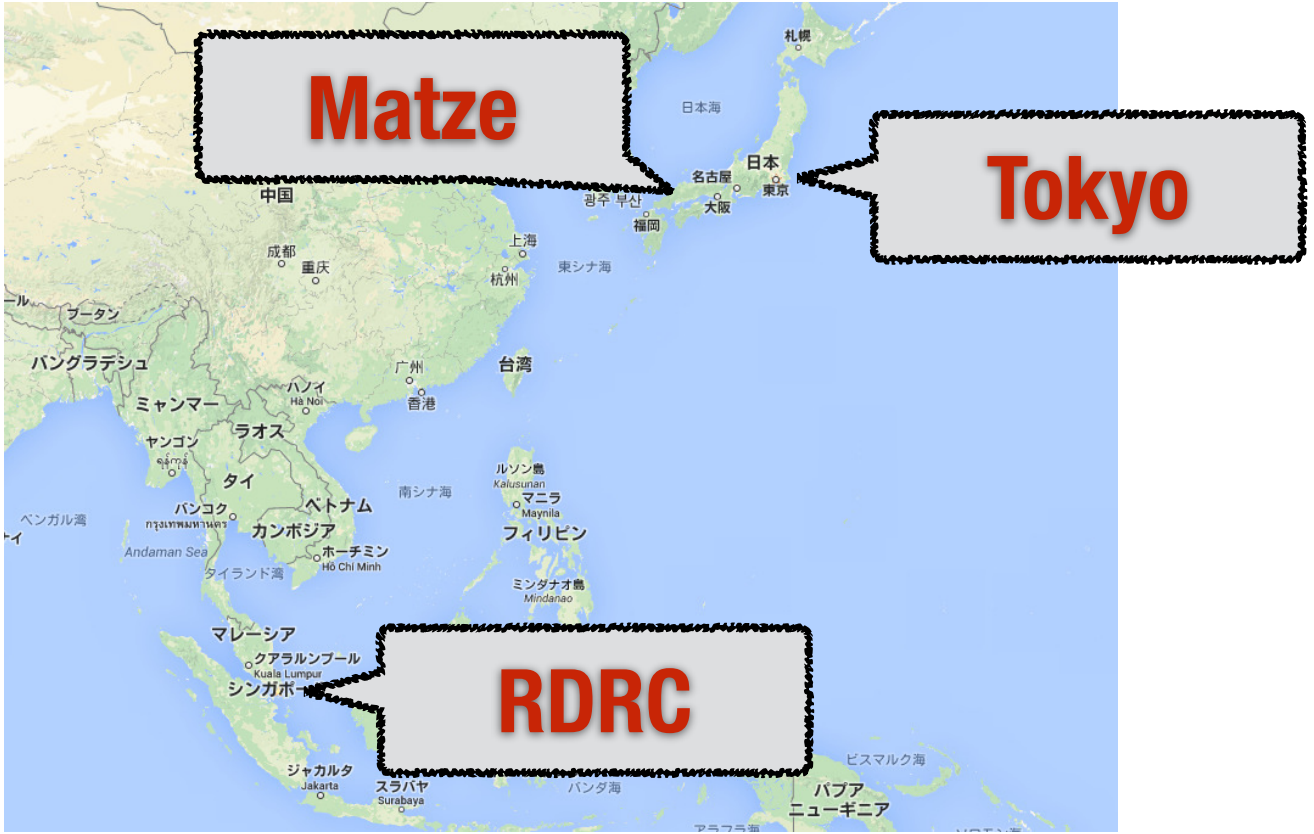


self.introduce



=>
{
  name: “SHIBATA Hiroshi”,
  nickname: “hsbt”,
  title: “Chief engineer at GMO Pepabo, Inc.”,
  commit_bits: [“ruby”, “rake”, “rubygems”, “rdoc”, “tdiary”, “hiki”, “railsgirls”, “railsgirls-jp”, “jenkins”],
  sites: [“ruby-lang.org”, “rubyci.com”, “railsgirls.com”, “railsgirls.jp”],
}

I’m from Tokyo, Japan



I’m from Asakusa.rb

Asakusa.rb is one of the most active meet-ups in Tokyo, Japan.

@a_matsuda (Ruby/Rails committer, RubyKaigi organizer)
@kakutani (RubyKaigi organizer)
@ko1 (Ruby committer)
@takkanm (Ruby/Rails programmer)
@gunjisatoshi (Rubyist Magazine editor)
@hsbt (Me!)



I’m from the Ruby core team

We are working on the next version of Ruby, 2.3.0, now. However, the main feature is under “TBD” status. Some libraries will be omitted from stdlib such as rake, net-telnet, etc..

If you have any issue, please submit it to our issue tracker at http://bugs.ruby-lang.org

We hold the core developer meeting every months, and discuss various issues and ideas on Ruby. See https://bugs.ruby-lang.org/projects/ruby/wiki/#Developer-Meetings for details.

Middleware as Code with mruby



Hiroshi Shibata
Chief Engineer, GMO Pepabo, Inc.

HTTP Programming with mruby

mruby is the lightweight implementation of the Ruby language and was released about a year ago. Can we use mruby to write web services?

This answer is YES - our company used mruby in large scaled web services. Even with mruby, we were able to create web services with tests and gems, and it also helped to solve some problems using Ruby code outside of a Rails application. In essence, mruby also provides programming features like HTTP to us web programmers.

Speaker's Bio

CRuby committer and root operation engineer of ruby-lang.org. I am a full-stack developer at GMO Pepabo.

Today’s target

- Web engineer(Rails programmer)
- Operation engineer
- QA/test engineer
- mruby committer(Matz)

mruby

What's mruby?

“mruby is the lightweight implementation of the Ruby language complying to (part of) the ISO standard. Its syntax is Ruby 1.9 compatible.”

https://github.com/mruby/mruby#whats-mruby

Differences between mruby and CRuby

- The mruby runtime and libraries are embedded all into a single binary.
- By default, mruby provides just a minimum set of standard libraries such as String, Array, Hash, etc.
- Some of standard libraries in CRuby are NOT bundled in mruby, for example, IO, Regex, Socket, etc..
- mruby doesn't provide “require”, “sleep”, “p”, etc.

Advantages of mruby against CRuby

- Single binary without pure ruby files.
- Embeddable into middlewares like below:
  - apache/nginx
  - groonga
  - mysql
- Fun!!1 # most important thing

Dive into mruby build

You can declare prerequisite libraries in `build_config.rb`

```
MRuby::Build.new do |conf|

  toolchain :gcc

  conf.gembox 'full-core'

  conf.gem :github => 'iij/mruby-io'
  conf.gem :github => 'iij/mruby-env'
  (snip)
  conf.gem :github => 'matsumoto-r/mruby-uname'

  conf.gem '../mrbgems/ngx_mruby_mrblib'

end
```

mrbgem

See https://github.com/mruby/mruby/blob/master/doc/mrbgems/README.md :)

- mrbgem.rake
  Endpoint of mrbgem, put MRuby::Gem::Specification
- mrblib/
  Sources for pure ruby extension
- src/
  Sources for C extension



Demo



Middleware meets mruby

mruby has embeddable mechanism for middlewares like http server, search engine, etc..

Embedded mruby provides ruby runtime and syntax to middlewares. It’s so powerful programming environment for Rubyists.

ngx_mruby

Introduction to ngx_mruby

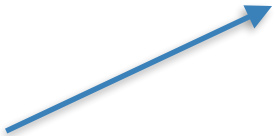
“ngx_mruby is A Fast and Memory-Efficient Web Server Extension Mechanism Using Scripting Language mruby for nginx.”

https://github.com/matsumoto-r/ngx_mruby#whats-ngx_mruby

```
location /hello {
    mruby_content_handler /path/to/hello.rb cache;
}
```



In “nginx.conf”!!!



```
location /proxy {
    mruby_set_code $backend '
        backends = [
          "test1.example.com",
          "test2.example.com",
          "test3.example.com",
        ]
        backends[rand(backends.length)]
    ';
}
```

How to build ngx_mruby (and mruby)

I suggest to try it on OS X or Linux environment. You can change embedded mgem via “build_config.rb” in ngx_mruby repository.

```
$ git clone https://github.com/matsumoto-r/ngx_mruby
$ git clone https://github.com/nginx/nginx
$ cd ngx_mruby
$ git submodule init && git submodule update

comment-out mruby-redis and mruby-vedis

$ ./configure —with-ngx-src-root=../nginx
$ make build_mruby
$ make
$ cd ../nginx
$ ./objs/nginx -V
```

mruby_content_handler

It's basic usage of ngx_mruby. These handlers are invoked at every requests

```
location /hello {
    mruby_content_handler /path/to/hello.rb cache;
}
```

```
location /hello {
    mruby_content_handler_code '
        Nginx.rputs "hello"
        Nginx.echo "world!"
    ';
}
```

mruby_set

mruby_set sets the return value from mruby code to nginx variable

```
location /proxy {
    mruby_set $backend /path/to/proxy.rb cache;
}
```

```
location /proxy {
    mruby_set_code $backend '
        backends = [
          "test1.example.com",
          "test2.example.com",
          "test3.example.com",
        ]
        backends[rand(backends.length)]
    ';
```

mruby_init

It’s invoked when nginx master process launched.

```
http {
  mruby_init /path/to/init.rb;

  server {
    location / {
      mruby_content_handler /path/to/handler.rb;
    }
  }
}
```

mruby_init_worker/mruby_exit_worker

It's invoked when nginx "worker" process is launched.

```
http {
  mruby_init /path/to/init.rb;

  mruby_init_worker /path/to/init_worker.rb;
  mruby_exit_worker /path/to/exit_worker.rb;

  server {
    location / {
      mruby_content_handler /path/to/handler.rb;
    }
  }
}
```

Sample code of ngx_mruby

```
class ProductionCode
  def initialize(r, c)
    @r, @c = r, c
  end

  def allowed_ip_addresses
    %w[
      128.0.0.1
    ]
  end

  def allowed?
    if (allowed_ip_addresses &
        [@c.remote_ip, @r.headers_in['X-Real-IP'], @r.headers_in['X-Forwarded-For']].compact).size > 0
     return true
    end
    (snip for memcached)
  end
 return false
end
ProductionCode.new(Nginx::Request.new, Nginx::Connection.new).allowed?
```

Sample configuration of nginx

```
location /path {

  mruby_set $allowed ‘/etc/nginx/handler/production_code.rb' cache;

  if ($allowed = 'true'){
    proxy_pass http://upstream;
  }

  if ($allowed = 'false'){
    return 403;
  }
}
```

Use cases of ngx_mruby

- Calculation of digest hash for authentication.
- Data sharing with Rails application.
- To replace ugly complex nginx.conf with clean, simple, and TESTABLE ruby code.

Middleware as Code

Our use cases

Data sharing with Rails & Restricted access

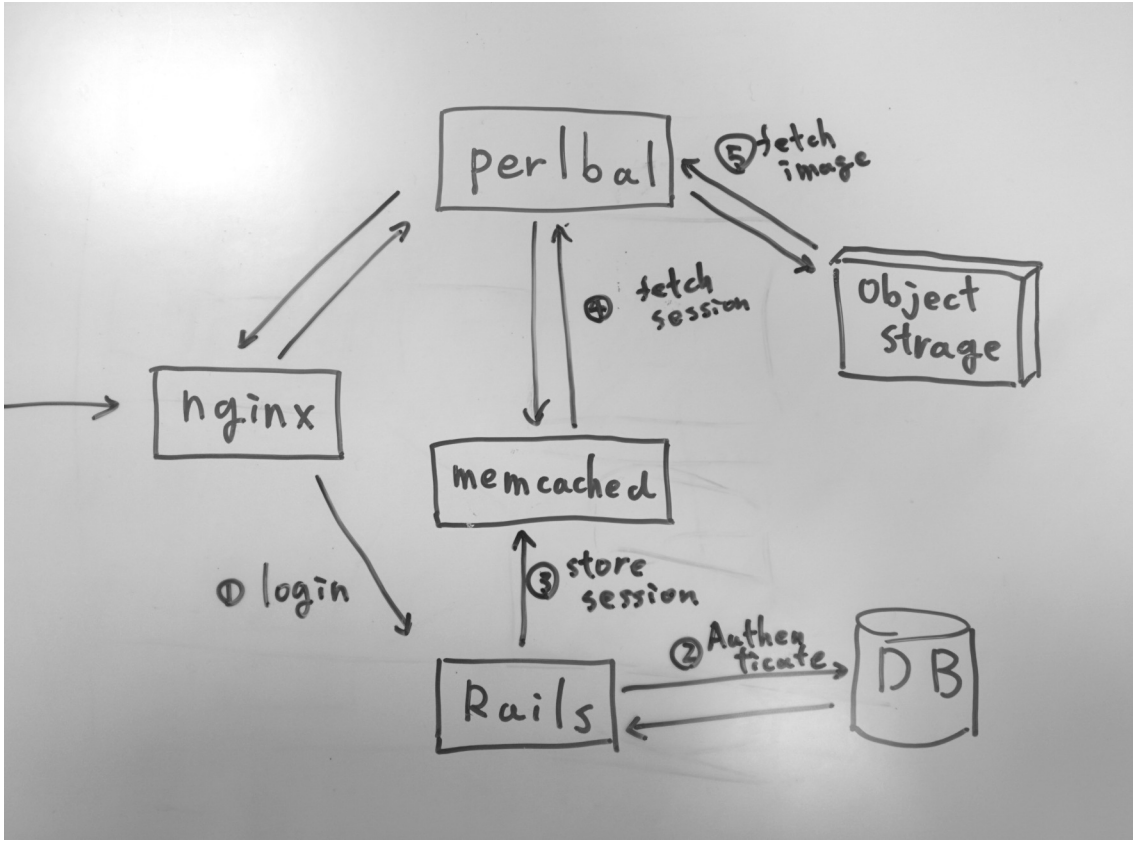
We have photo sharing service named “30days album”

This service concept is private photo sharing.

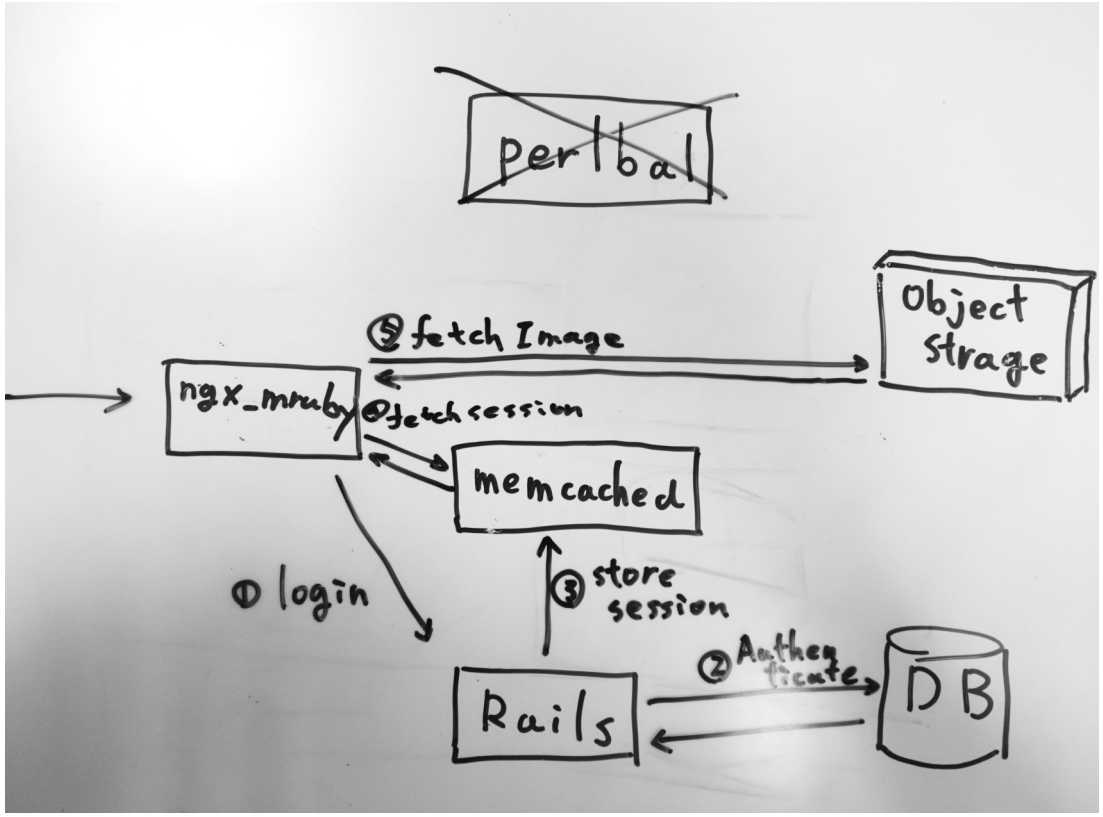
We need to have restrict access mechanism for image files and share data of Rails to http middleware.



Before ngx_mruby



Current architecture using ngx_mruby



Data sharing with rails using mruby

You can share data via persisted storage like memcahed/redis

```
allowed = false
memcached = Memcache.new("127.0.0.1")

allowed_data = memcached.get(session_id)
```

In this case, We share data using rails session key via cookie data.

Issue of connection leak and fd open cost

We tried to connect memcached at each request using “mruby_handler”

This approach open and close network connection each request. We faced problem of connection overflow

We discussed the issue with the ngx_mruby author, @matsumoto-r. He solved this issue in a few days. He provided us new handler named “mruby_init_worker”.

OSS 

Share connection in worker process

nginx.conf

```
http {
    (snip)
    mruby_init_worker /etc/nginx/handler/session_connect.rb cache;
    mruby_exit_worker /etc/nginx/handler/session_disconnect.rb cache;
    (snip)
}
```

session_connect.rb

```
userdata = Userdata.new("memcached_#{Process.pid}")
userdata.memcached = Memcached.new('128.0.0.1')
```

session_disconnect.rb

```
userdata = Userdata.new("memcached_#{Process.pid}")
userdata.memcached.close if userdata.memcached
```

Restrict access to image asset

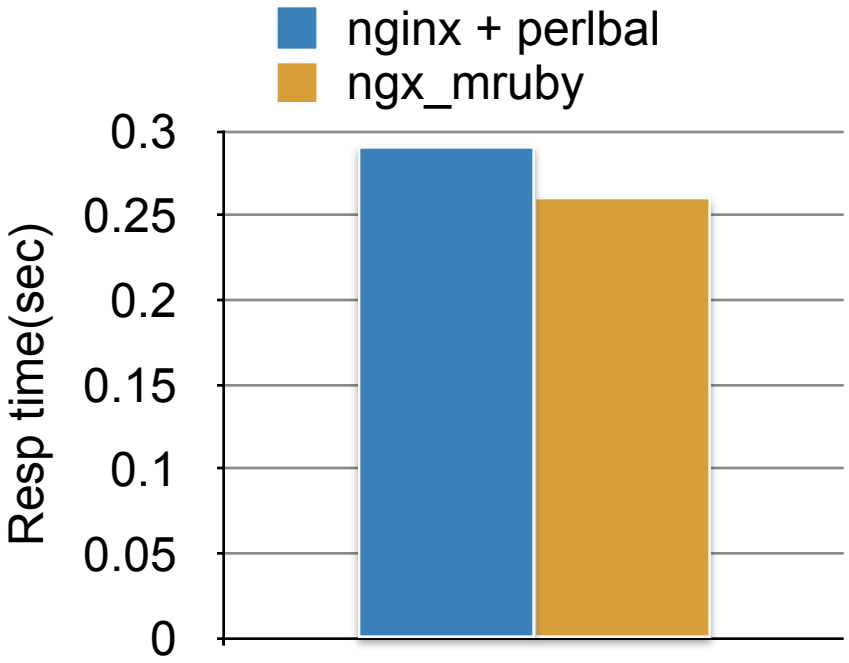
Allowing uri string in session is compared accessing uri. If it matches, ngx_mruby allows this request to access image asset.

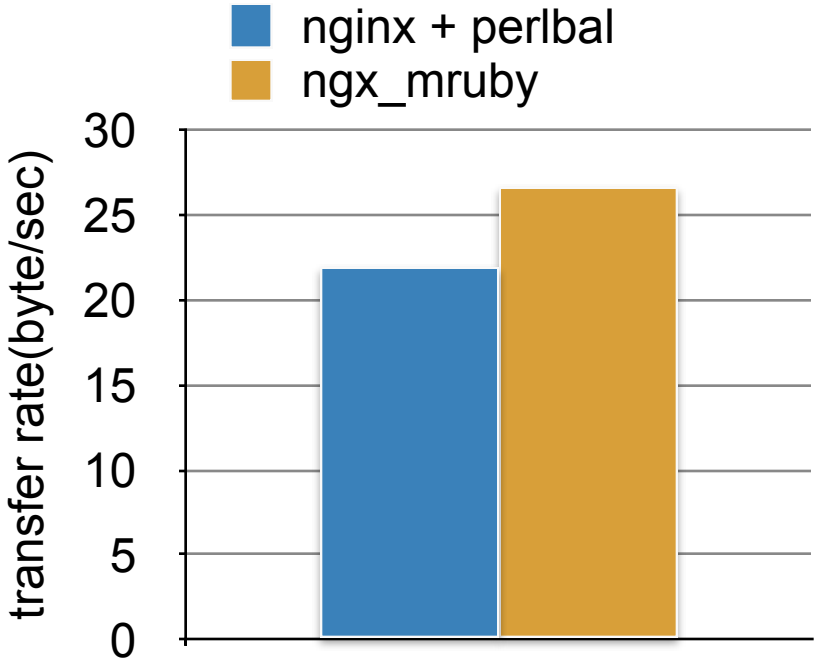
```
allowed = false
userdata = Userdata.new("memcached_#{Process.pid}")

if allowed_data = userdata.memcached.get(session_id)
  if @r.uri =~ /\/image\/#{allowed_data}/
    allowed = true
  end
end

allowed
```

Comparison of performance





 STAR IT 

https://github.com/matsumoto-r/ngx_mruby

Testing code of mruby

Someone said “Test is redundant”

What’s motivation

- We are using ngx_mruby in production.
- We should test every production code.
- Testing mruby code is a cutting edge technical issue.

Prototype concept

- Use CRuby(version independent: 2.0.0, 2.1, 2.2)
- Use test-unit
- Test “ruby code” without real world behavior.

Sample code of ngx_mruby

```
class ProductionCode
  def initialize(r, c)
    @r, @c = r, c
  end

  def allowed_ip_addresses
    %w[
      128.0.0.1
    ]
  end

  def allowed?
    if (allowed_ip_addresses &
        [@c.remote_ip, @r.headers_in['X-Real-IP'], @r.headers_in['X-Forwarded-For']].compact).size > 0
      return true
    end
    (snip for memcached)
  end
 return false
end
ProductionCode.new(Nginx::Request.new, Nginx::Connection.new).allowed?
```

Dummy class of ngx_mruby

```
class Nginx
  class Request
    attr_accessor :uri, :headers_in, :args, :method, :hostname
    def initialize
      @uri = nil
      @headers_in = {}
      @args = nil
      @method = 'GET'
      @hostname = nil
    end
  end

  class Connection
    attr_accessor :remote_ip
    def initialize
      @remote_ip = nil
    end
  end
```

Dummy class of mgem

```
Memcached = MemCache

class Memcached
  def close
    servers.each(&:close)
  end
end

class Userdata
  def initialize(*args)
  end

  def memcached
    Memcached.new('127.0.0.1:11211')
  end
end
```

Skeleton of test-case

```
require_relative '../lib/production/code/path/mruby.rb'

class MRubyTest < Test::Unit::TestCase
  def setup
    @r = Nginx::Request.new
    @c = Nginx::Connection.new
  end

  def test_discard_access
    assert !ProductionCode.new(@r, @c).allowed?
  end
end
```

Restrict requests with cookie session

```
require_relative '../lib/production/code/path/mruby.rb'

class MRubyTest < Test::Unit::TestCase
  def setup
    @r = Nginx::Request.new
    @c = Nginx::Connection.new
  end

  def test_session_access
    MemCache.new('127.0.0.1').set 'a77a2a0cc91b739438dfc9dc47c5dd36'
    @r.headers_in['cookie'] = '_session=a77a2a0cc91b739438dfc9dc47c5dd36;'

    @r.uri = '/secret/file/path'
    assert ProductionCode.new(@r, @c).allowed?
  end
end
```

Run test

```
% ruby test/production_code_test.rb
Loaded suite test/production_code_test
Started
.........

Finished in 0.031017 seconds.
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------
9 tests, 15 assertions, 0 failures, 0 errors, 0 pendings, 0 omissions, 0 notifications
100% passed
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------------------------------------
------------------
290.16 tests/s, 483.61 assertions/s
```

Test is back!!

Our concerns on CRuby testing

- We can test “ruby code”. But it’s not fulfill testing requirements. We need to test ngx_mruby behavior.
- We use a lot of mock/stub classes. It’s ruby’s dark-side.
- We need to make easy task runner.

Testing code of mruby using mruby

Use mruby directly instead of CRuby

mruby-mtest

build_config.rb

```
MRuby::Build.new do |conf|

  (snip)

  conf.gem :github => 'matsumoto-r/mruby-uname'

  # ngx_mruby extended class
  conf.gem '../mrbgems/ngx_mruby_mrblib'

  con.gem :github => 'iij/mruby-mtest'

  (snip)
end
```

test_4m_test.rb

```
class Test4MTest < MTest::Unit::TestCase
  def test_assert
    assert(true)
    assert(true, 'true sample test')
  end
end

MTest::Unit.new.run
```

Inline testing for mruby-mtest

```
class ProductionCode
  (snip)
end

if Object.const_defined?(:MTest)
  class Nginx
    (snip)
  end

  class TestProductionCode < MTest::Unit::TestCase
    (snip)
  end

  MTest::Unit.new.run
else
  ProductionCode.new(Nginx::Request.new, Nginx::Connection.new).allowed?
end
```

Build mruby for mruby testing

You need to get mruby binary before embed ngx_mruby.

```
$ cd ngx_mruby/mruby
$ cp ../build_config.rb .
$ make
$ cp bin/mruby /path/to/test/bin
```

```
% ./path/to/test/bin/mruby -v
mruby 1.1.0 (2014-11-19)
^C
```

Test runner for mruby-mtest

```
require 'rake'

desc 'Run mruby-mtest'
task :mtest do
  target = "modules/path/to/production/code"
  mruby_binary = File.expand_path("../#{target}/test_bin/mruby", __FILE__)
  mruby_files = FileList["#{target}/**/*.rb"]
  mruby_files.each do |f|
    absolute_path = File.expand_path("../#{f}", __FILE__)
    system "#{mruby_binary} #{absolute_path}"
  end
end
```

Advantage of mruby testing

Rapid!

```
% rake mtest

# Running tests:

.........

Finished tests in 0.007924s, 1135.7900 tests/s, 1892.9833 assertions/s.

9 tests, 15 assertions, 0 failures, 0 errors, 0 skips
```

Deployment

Deployment strategy

We need to prepare following things for production use:

- Build target binary ngx_mruby in production environment
- Write manifest file of puppet/chef
- Test binary and mruby code continuously

Deploy!

Build on docker

https://gist.github.com/hsbt/f5a3a83ec2ebf8169f38

```
FROM centos:7
MAINTAINER hsbt

RUN yum -y install --enablerepo=extras epel-release
RUN yum -y groupinstall "Development Tools"
RUN yum -y install git libffi-devel libevent-devel cyrus-sasl-devel openssl-devel pcre-devel tar zlib-devel rake
(snip)
RUN git clone --depth 1 --branch v1.8.11 --recursive https://github.com/matsumoto-r/ngx_mruby.git /usr/local/src/ngx_mruby
ADD build_config.rb /usr/local/src/ngx_mruby/build_config.rb

RUN git clone --depth 1 --branch v1.9.0 https://github.com/nginx/nginx.git /usr/local/src/nginx
RUN cd /usr/local/src/ngx_mruby && NGINX_SRC_ENV=/usr/local/src/nginx NGINX_CONFIG_OPT_ENV="--prefix=/etc/nginx …'" sh build.sh
```

Build on docker

You can modify build_config.rb. After that, You run “docker build”

```
$ docker build --tag=nginx_mruby:centos7 --file=Dockerfile.centos7 .
```

You got ngx_mruby binary for centos7 with “docker run”

```
$ export DIR=`pwd` && docker run --volume="$DIR:/tmp:rw" --user=root "nginx_mruby:centos7" "cp"
"-a" "/usr/local/src/nginx/objs/nginx" "/tmp"
```

official nginx package + ngx_mruby

```
% rpm -qlp nginx-1.9.0-1.el7.ngx.x86_64.rpm
/etc/logrotate.d/nginx
/etc/nginx
/etc/nginx/conf.d
/etc/nginx/conf.d/default.conf
(snip)
/usr/sbin/nginx
/usr/share/nginx
/usr/share/nginx/html
/usr/share/nginx/html/50x.html
/usr/share/nginx/html/index.html
/var/cache/nginx
/var/log/nginx
```

Replace ngx_mruby binary from official package

example of puppet

```
class ngx_mruby::install {
 include ngx_mruby::params
 (snip)
 file { '/usr/local/src/nginx.tar.gz':
    source  => "puppet:///modules/ngx_mruby/usr/local/src/nginx-${::platform}.tar.gz",
    notify  => Exec['cleanup and unzip ngx_mruby'],
  }

  $install_dir = dirname($ngx_mruby::params::nginx)

  exec { 'cleanup and unzip ngx_mruby':
    path        => ['/bin', '/usr/bin'],
    command     => "tar xf /usr/local/src/nginx.tar.gz -C ${install_dir}",
    require     => File['/usr/local/src/nginx.tar.gz'],
    refreshonly => true,
  }
 (snip)
}
```

Next challenge

- mruby binary can have different library from one in production.
- For continuous integration, we need to prepare cross-compile or live compile environment.
- Replace nginx.conf with mruby code backed by test code.

cross-compile and CI

mruby supports cross-compile configuration.

We made “https://github.com/matsumoto-r/mruby-cross-compile-on-mac-osx”

```
MRuby::Build.new do |conf|

  # ... (snip) ...

  # the last line of conf.gem
  load '/path/to/mruby-cross-compile-on-mac-osx/mrbgem.rake'
end
```

mruby
in the future

mruby-ipvs

“mruby-ipvs is an interface to the IP Virtual Server(IPVS) for mruby.”

https://github.com/rrreeeyyy/mruby-ipvs

```
# Create IPVS::Service instance.
s = IPVS::Service.new({
  'addr' => '10.0.0.1:80',
  'port' => 80,
  'sched_name' => 'wrr'
})

# apply to IPVS.
s.add_service
```

```
# Create IPVS::Dest instance.
d = IPVS::Dest.new({
  'addr' => '192.168.0.1',
  'port' => 80,
  'weight' => 1
})

# Add destination to IPVS::Service instance.
s.add_dest(d)
```

mruby_cgroup

mruby cgroup module using libcgroup

https://github.com/matsumoto-r/mruby-cgroup

```
rate = Cgroup::CPU.new "test"
core = Cgroup::CPUSET.new "test"

rate.cfs_quota_us = 30000
core.cpus = core.mems = "0"

rate.create
core.create

# CPU core 0 and rate 30%
puts "attach /test group with cpu core 0 and rate 30%"
rate.attach
core.attach
```

We should use mruby!