Ruby Fundamentals
Day Three

Dan Klopp,
Coban Tun

Table of Contents

REST

- REpresentational Stateful Transfer
- Popular examples
  - Openstack and AWS
  - Sensu and Nagios
- A very common stateless architecture

Our Focus

- We are concerned with implementing a simple API
- We are NOT concerned with proper design
- We are concerned with terminology
- We are NOT concerned with security

HTTP Verbs

- PUT
  - Place the entity at the specified resource
- DELETE
  - Delete the specified resource
- GET
  - Get the entity at the specified resource
- POST
  - Instruct resource to act on enclosed entity

HTTP PUT

- PUT
  - Place the entity at the specified URI
  - curl -X PUT -d arg=val -d arg2=val2 localhost:8080
- If entity already exists at location, it updates the entity
- Example
  - PUT 95134 localhost/zipcode

HTTP DELETE

- Delete the URI
- Note: successful return code does not mean resource was deleted, only that the server intends to delete it
- Example:
  - DELETE localhost/zipcode/95134

HTTP GET

- Get the entity at the specified resource
- Should be idempotent
- Should only retrieve
  - This is a “guideline”
  - https://www.youtube.com/watch?v=jl0hMfqNQ-g
- Example
  - GET localhost/zipcode/95134

<https://www.ietf.org/rfc/rfc2616.txt>

HTTP POST

- Submit the enclosed entity to the URI handler
- This is fundamentally different from PUT
  - PUT places entity at URI
  - POST asks URI to act on entity
- Message is encoded in HTTP body, not URI
- Example
  - POST getallbusinessesat localhost/zipcode/95134

POST in Practice

- Theoretically POST should only be used for changes

POST in Practice

- Theoretically POST should only be used for changes
- In practice also used as a GET

POST in Practice

- Theoretically POST should only be used for changes
- In practice also used as a GET
  - GET is limited to 4000 characters or less (implementation dependent)
  - POST has no such limit

We Will Be Using GET

- It makes it simple to implement

We Will Be Using GET

- It makes it simple to implement
- Strictly speaking we are violating the standard

We Will Be Using GET

- It makes it simple to implement
- Strictly speaking we are violating the standard
- Properly implementing HTTP requests and responses is beyond the scope of a fundamentals course

AWS HTTP GET Example

- Querying AWS for metadata

```
# curl -X GET http://169.254.169.254/latest/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-action
```

- Polling the hostname

```
# curl -X GET http://169.254.169.254/latest/meta-data/hostname
ip-10-248-111-220.us-west-2.compute.internal
```

TAOS

Example Server

- Live Demonstration
- Source code at:

https://github.com/dsklopp/rest_example_server.git

Building A REST API

- First, let's design the API
- GET /
  - returns all keys
- GET /key
  - returns value of key "key"
- GET /key/value
  - sets value of key "key" to value

Building A REST API

- Where to begin?

Building A REST API

- Where to begin?
  - We'll create a stub and incrementally add functionality

The Sample API

- First step

```
class SocketListen
  def initialize port=8181
    @data = { 'apache' => 'apache',
      'bsd' => 'mit', 'chef' => 'apache' }
    puts @data
  end
end
SocketListen.new()
```

The Sample API

- Output

```
: tyrion @ Casterly Rock ; ruby rest_1.rb
{"apache"=>"apache", "bsd"=>"mit", "chef"=>"apache"}
: tyrion @ Casterly Rock ;
```

The Sample API

- Output

```
: tyrion @ Casterly Rock ; ruby rest_1.rb
{"apache"=>"apache", "bsd"=>"mit", "chef"=>"apache"}
: tyrion @ Casterly Rock ;
```

- Let's add a socket

The Sample API

- We'll use the TCPSocket Class of the Socket Module

```
require 'socket'

class SocketListen
  def initialize port=8181
    @data = { 'apache' => 'apache',
      'bsd' => 'mit', 'chef' => 'apache' }
    @port = port || 8181
    @server = TCPServer.new @port
  end
end
SocketListen.new()
```

The Sample API

- Create an infinite loop to listen for connections after creating the TCPServer

```
@server = TCPServer.new @port
loop {
  client = @server.accept
  request = client.gets
  puts request
  client.close
}
```

The Sample API

- Running that code and calling it via:

```
curl -X GET localhost:8181
```

- Will show the output of:

```
GET / HTTP/1.1
GET / HTTP/1.1
GET / HTTP/1.1
GET / HTTP/1.1
```

- See “rest_3.rb” for sample
- Live demonstration

TAOS

The Sample API

- It would be nice if the server could talk with the client...

```
@server = TCPServer.new @port
loop {
  client = @server.accept
  request = client.gets
  puts request
  client.puts request
  client.close
}
```

- This will echo the client's request back to the client
- Live demonstration, see "ruby_3b.rb"

TAOS

The Sample API

- When handling a request, we'll send the request to a dedicated method. We'll call the method "handle".
- The call is:

```
@server = TCPServer.new @port
loop {
  client = @server.accept
  request = client.gets
  puts request
  client.puts handle request
  client.close
}
```

The Sample API

- For those more comfortable with parenthesis around method calls, this is equivalent code:

```
@server = TCPServer.new @port
loop {
  client = @server.accept
  request = client.gets()
  puts(request)
  client.puts(handle(request))
  client.close()
}
```

The Sample API

- The handle method only returns “GET” for the GET HTTP verb.

```
def handle arg
  verb = arg.split()[0]
  arg=arg.split()[1..-1].join(" ")
  if verb == 'GET'
    return "GET    "
  else
    return "NOT IMPLEMENTED"
  end
end
```

- Live demonstration, see “rest_4.rb”

TAOS

The Sample API

- So far we have
  - A socket that listens for requests
  - A program that returns the request over the socket
  - Trivial request validation
- Next, act on GET URI requests

Building A REST API

- Refresher, remember we are responding to:
  - GET /
    - returns all keys
  - GET /key
    - returns value of key "key"
  - GET /key/value
    - sets value of key "key" to value

The Sample API

- Modify

```
def handle arg
  verb = arg.split()[0]
  arg=arg.split()[1..-1].join(" ")
  if verb == 'GET'
    return "GET    "
  else
    return "NOT IMPLEMENTED"
  end
end
```

- to...

The Sample API

- We are deferring action on the URI to another method

```
def handle arg
  verb = arg.split()[0]
  arg=arg.split()[1..-1].join(" ")
  if verb == 'GET'
    return handle_get arg
  else
    return "NOT IMPLEMENTED"
  end
end
```

TAOS

The Sample API

- How do we parse the URI?

The Sample API

- How do we parse the URI?
- We'll convert the relevant portion into an array

The Sample API

- How do we parse the URI?
- We'll convert the relevant portion into an array

```
>> arg="GET /bsd HTTP/1.1"
=> "GET /bsd HTTP/1.1"
```

The Sample API

- How do we parse the URI?
- We'll convert the relevant portion into an array

```
>> arg="GET /bsd HTTP/1.1"
=> "GET /bsd HTTP/1.1"
>> arg.split()
=> ["GET", "/bsd", "HTTP/1.1"]
```

The Sample API

- How do we parse the URI?
- We'll convert the relevant portion into an array

```
>> arg="GET /bsd HTTP/1.1"
=> "GET /bsd HTTP/1.1"
>> arg.split()
=> ["GET", "/bsd", "HTTP/1.1"]
>> # We've already split the string earlier in handle()
?> # Handle did this and then passed it on to handle_get
?> arg=arg.split()[1..-1].join(" ")
=> "/bsd HTTP/1.1"
```

The Sample API

```
?> arg=arg.split()[1..-1].join(" ")
=> "/bsd HTTP/1.1"
```

The Sample API

```
?> arg=arg.split()[1..-1].join(" ")
=> "/bsd HTTP/1.1"
>> # Isolate the URI
?> arg.split()[0]
=> "/bsd"
```

The Sample API

```
?> arg=arg.split()[1..-1].join(" ")
=> "/bsd HTTP/1.1"
>> # Isolate the URI
?> arg.split()[0]
=> "/bsd"
>> # Break up each argument
?> (arg.split[0]).split('/')
=> ["", "bsd"]
```

The Sample API

```
?> arg=arg.split()[1..-1].join(" ")
=> "/bsd HTTP/1.1"
>> # Isolate the URI
?> arg.split()[0]
=> "/bsd"
>> # Break up each argument
?> (arg.split[0]).split('/')
=> ["", "bsd"]
>> # Remove the empty string before the first /
?> (arg.split[0]).split('/')[1..-1]
=> ["bsd"]
```

The Sample API

- First handle “GET /”

```
def handle_get arg
  path = (arg.split[0]).split('/')[1..-1]
  if path.nil?
    keys=[]
    @data.each do |key,value|
      keys << key
    end
    puts keys
    return keys
  end
end
```

TAOS

The Sample API

- Now handle “GET /$key”

```
    else
      puts path[0] + " => " + @data[path[0]]
      return @data[path[0]] + "    "
```

The Sample API

- Now handle “GET /$key/$value”

```
    elsif path.size == 2
      @data[path[0].chomp] = path[1]
      puts path[1]
      return path[1]
```

The Sample API

- Now handle “GET /$key”

```
def handle_get arg
  path = (arg.split[0]).split('/')[1..-1]
  if path.nil?
    keys=[]
    @data.each do |key,value|
      keys << key
    end
    puts keys
    return keys
  end
end
```

Scope

Code References

- The sample REST API for reporting licenses

https://github.com/dsklopp/rest_example_server

- A functional HAProxy Management tool

https://github.com/dsklopp/haproxy_rest_rubyclass

- All slide related code

https://github.com/dsklopp/ruby_class_2015_samples

In Class Exercise / Homework

- As demonstrated in class, add an API to your haproxy management tool.
  - The calls it must honor are:

    GET /backend - show the active backend

    GET /backends - show all available backends

    GET /backend/$backend - set the backend to $backend and restart haproxy

In Class Exercise / Homework Comments

- This will not be easy
- Since the class is small we'll help the class out one on one as needed
- AWS instances will be available until Monday (Feb 2)

Questions?

TAOS