Testing Ratpack Applications

Dan Hyun `<@LSpacewalker>`

Ratpack is a developer friendly and productivity focused web framework. That's quite a claim to make. We'll explore how Ratpack's rich testing facilities strongly support this statement.

1. Intro

- Test framework Agnostic (Spock, JUnit, TestNG)
- Core fixutres in Java 8+, first-class Groovy Support available
- Most fixtures implement `java.lang.AutoCloseable`
  - Need to either close yourself or use in `try-with-resources`
  - Provides points of interaction that utilize an execute around pattern in cases where you need the fixture once.

2. Hello World

2.1. Dependencies

testing-ratpack-apps.gradle

```
plugins { ❶
  id 'io.ratpack.ratpack-groovy' version '1.2.0' ❷
}

repositories {
  jcenter()
}

dependencies {
  runtime "org.apache.logging.log4j:log4j-slf4j-impl:${log4j}"
  runtime "org.apache.logging.log4j:log4j-api:${log4j}"
  runtime "org.apache.logging.log4j:log4j-core:${log4j}"
  runtime 'com.lmax:disruptor:3.3.2'

  testCompile ratpack.dependency('groovy-test') ❸
```

```
  testCompile ('org.spockframework:spock-core:1.0-groovy-2.4') {
    exclude module: "groovy-all"
  }

  testCompile 'junit:junit:4.12'
  testCompile 'org.testng:testng:6.9.10'
}
```

❶ Use Gradle's incubating Plugins feature

❷ Pull in and apply Ratpack's Gradle plugin from Gradle's Plugin Portal

❸ Pull in `'io.ratpack:ratpack-groovy-test` from Bintray

2.2. Hello World Under Test

ratpack.groovy

```
import static ratpack.groovy.Groovy.ratpack

ratpack {
  handlers {
    get {
      render 'Hello Greach 2016!'
    }
  }
}
```

MainClassApp

```
import groovy.transform.CompileStatic
import ratpack.func.Action
import ratpack.groovy.handling.GroovyContext
import ratpack.groovy.handling.GroovyHandler
import ratpack.handling.Chain
import ratpack.server.RatpackServer
import ratpack.server.RatpackServerSpec

@CompileStatic
class MainClassApp {
  public static void main(String[] args) throws Exception {
    RatpackServer.start({ RatpackServerSpec serverSpec -> serverSpec
      .handlers({ Chain chain ->
        chain.get({GroovyContext ctx ->
          ctx.render 'Hello Greach 2016!'
        } as GroovyHandler)
      } as Action<Chain>)
```

```
    } as Action<RatpackServerSpec>)
  }
}
```

2.3. Verify

```
$ curl localhost:5050
Hello Greach 2016!
```

3. `assert true`

3.1. Spock Hello World

HelloWorldSpec.groovy

```
import ratpack.groovy.test.GroovyRatpackMainApplicationUnderTest
import ratpack.test.MainClassApplicationUnderTest
import spock.lang.AutoCleanup
import spock.lang.Shared
import spock.lang.Specification
import spock.lang.Unroll

class HelloWorldSpec extends Specification {

  // tag::GroovyScriptAUT[]
  @AutoCleanup
  @Shared
  GroovyRatpackMainApplicationUnderTest groovyScriptApplicationunderTest = new
 GroovyRatpackMainApplicationUnderTest()
  // end::GroovyScriptAUT[]

  // tag::MainClassAUT[]
  @AutoCleanup
  @Shared
  MainClassApplicationUnderTest mainClassApplicationUnderTest = new
 MainClassApplicationUnderTest(MainClassApp)
  // end::MainClassAUT[]

  @Unroll
  def 'Should render \'Hello Greach 2016!\' from #type'() {
    when:
    def getText = aut.httpClient.getText()

    then:
```

```
    getText == 'Hello Greach 2016!'

    where:
    aut                               | type
    groovyScriptApplicationunderTest | 'ratpack.groovy'
    mainClassApplicationUnderTest    | 'MainClassApp.groovy'
  }
}
```

3.2. Junit Test

HelloJunitTest.groovy

```
import groovy.transform.CompileStatic
import org.junit.Assert
import org.junit.AfterClass
import org.junit.BeforeClass
import org.junit.Test
import ratpack.groovy.test.GroovyRatpackMainApplicationUnderTest
import ratpack.test.MainClassApplicationUnderTest
import ratpack.test.http.TestHttpClient

import static org.junit.Assert.assertEquals

@CompileStatic
class HelloJunitTest {

  static GroovyRatpackMainApplicationUnderTest groovyScriptApplicationunderTest
  static MainClassApplicationUnderTest mainClassApplicationUnderTest

  @BeforeClass
  static void setup() {
    groovyScriptApplicationunderTest = new GroovyRatpackMainApplicationUnderTest()
    mainClassApplicationUnderTest = new MainClassApplicationUnderTest(MainClassApp)
  }

  @AfterClass
  static void cleanup() {
    groovyScriptApplicationunderTest.close()
    mainClassApplicationUnderTest.close()
  }

  @Test
  def void testHelloWorld() {
    [
      groovyScriptApplicationunderTest,
      mainClassApplicationUnderTest
```

```
    ].each { aut ->
      TestHttpClient client = aut.httpClient
      assertEquals('Hello Greach 2016!', client.getText())
    }
  }
}
```

4. Unit testing

4.1. `GroovyRequestFixture`

Testing Standalone Handlers

ImportantHandler.groovy

```
import groovy.transform.CompileStatic
import ratpack.groovy.handling.GroovyContext
import ratpack.groovy.handling.GroovyHandler

@CompileStatic
class ImportantHandler extends GroovyHandler {
  @Override
  protected void handle(GroovyContext context) {
    context.render 'Very important handler'
  }
}
```

ratpack.groovy

```
import static ratpack.groovy.Groovy.ratpack

ratpack {
  handlers {
    get(new ImportantHandler()) ❶
  }
}
```

❶ Bind our `ImportantHandler` to `GET /`

Failing test

```
def 'should render \'Very important handler\''() {
  when:
  HandlingResult result = GroovyRequestFixture.handle(new ImportantHandler()) {}
```

```
  then:
  result.bodyText == 'Very important handler' (1)
}
```

(1) Consult the `HandlingResult` for response body

WARN: This test will fail

What happened?

`Context#render(Object)` uses Ratpack's rendering framework. `GroovyRequestFixture` does not actually serialize rendered objects to `Response` of `HandlingResult`. For this test to pass you must either modify the Handler or modify the expectation:

Modify the handler:

ImportantSendingHandler.groovy

```
import groovy.transform.CompileStatic
import ratpack.groovy.handling.GroovyContext
import ratpack.groovy.handling.GroovyHandler

@CompileStatic
class ImportantSendingHandler extends GroovyHandler {
  @Override
  protected void handle(GroovyContext context) {
    context.response.send('Very important handler')
  }
}
```

Modify the expectation:

ImportantHandlerUnitSpec.groovy

```
  def 'should render \'Very important handler\''() {
    when:
    HandlingResult result = GroovyRequestFixture.handle(new ImportantHandler()) {}

    then:
    String rendered = result.rendered(String) (1)
    rendered == 'Very important handler'
  }
```

❶ Retrieve the rendered object by type from `HandlingResult`

Everday use:

Modify request attributes

HeaderExtractionHandler.groovy

```
import groovy.transform.CompileStatic
import ratpack.groovy.handling.GroovyContext
import ratpack.groovy.handling.GroovyHandler

@CompileStatic
class HeaderExtractionHandler extends GroovyHandler {
  @Override
  protected void handle(GroovyContext context) {
    String specialHeader = context.request.headers.get('special') ?: 'not
 special' ❶
    context.render "Special header: $specialHeader"
  }
}
```

❶ Extract HTTP header and render a response to client

HeaderExtractionHandlingSpec.groovy

```
import ratpack.test.handling.HandlingResult
import spock.lang.Specification
import ratpack.groovy.test.handling.GroovyRequestFixture
import spock.lang.Unroll

class HeaderExtractionHandlingSpec extends Specification {

  @Unroll
  def 'should render #expectedValue with special header value'() {
    when:
    HandlingResult result = GroovyRequestFixture
      .handle(new HeaderExtractionHandler(), requestFixture)

    then:
    def rendered = result.rendered(CharSequence)
    rendered == "Special header: $expectedValue"

    where:
    expectedValue | requestFixture
    'greach2016'  | { header('special', 'greach2016') } ❶
    'not special' | {}
```

```
  }
}
```

❶ You can get a chance to configure the properties of the request to be made, can configure HTTP method, headers, request body, etc.

Modify and make assertions against context registry:

ProfileLoadingHandler.groovy

```
import groovy.transform.Canonical
import groovy.transform.CompileStatic
import ratpack.groovy.handling.GroovyContext
import ratpack.groovy.handling.GroovyHandler
import ratpack.registry.Registry

@CompileStatic
class ProfileLoadingHandler extends GroovyHandler {
  @Override
  protected void handle(GroovyContext context) {
    String role = context.request.headers.get('role') ?: 'guest' ❶
    String secretToken = context.get(String) ❷
    context.next(Registry.single(new Profile(role: role, token: secretToken))) ❸
  }
}

@Canonical
class Profile {
  String role
  String token
}
```

❶ Extract role from request header, defaulting to 'guest'

❷ Extract a String from the context registry

❸ Delegate to the next Handler in the chain and pass a new single Registry with a newly constructed Profile object

We can make use of `RequestFixture` to populate the Registry with any entries our stand-alone Handler may be expecting, such as a token in the form of a String.

ProfileLoadingHandlingSpec.groovy

```
  def 'handler should populate context registry with Profile'() {
    when:
```

```
    HandlingResult result = GroovyRequestFixture.handle(new ProfileLoadingHandler())
{
      header('role', 'admin') ❶
      registry { add(String, 'secret-token') } ❷
    }

    then:
    result.registry.get(Profile) == new Profile('admin', 'secret-token') ❸
  }
```

❶ Use `RequestFixture#header` to add Headers to the HTTP Request

❷ Use `RequestFixture#registry` to add a `String` to the Context registry

❸ Consult the HandlingResponse to ensure that the context was populated with a `Profile` object and that it meets our expectations

Let's put our `ProfileLoadingHandler` in a chain with a dummy Map renderer:

ProfileLoadingHandlingSpec.groovy

```
  def 'should be able to render Profile as map from Registry'() {
    when:
    HandlingResult result = GroovyRequestFixture.handle(new GroovyChainAction() { ❶
      @Override
      void execute() throws Exception {
        all(new ProfileLoadingHandler()) ❷
        get { ❸
          Profile profile = get(Profile)
          render([profile: [role: profile.role, token: profile.token]])
        }
      }
    }) {
      header('role', 'admin')
      registry { add(String, 'secret-token') }
    }

    then:
    result.rendered(Map) == [profile: [role: 'admin', 'token': 'secret-token']] ❹
  }
```

5. `GroovyEmbeddedApp`

`GroovyEmbeddedApp` represents an isolated subset of functionality that stands up a full Ratpack server.

It represents a very bare server that binds to an ephemeral port and has no base directory by default. `GroovyEmbeddedApp` is also `AutoCloseable`. If you plan on making more than a few interactions it may help to grab a `TestHttpClient` from the server, otherwise you can make use of `EmbeddedApp#test(TestHttpClient)` which will ensure that the `EmbeddedApp` is shut down gracefully. Javadocs for Ratpack are 100% tested and make use of `EmbeddedApp` to demonstrate functionality.

The `EmbeddedApp` is also useful in creating a test fixture that represents some network based resource that returns canned or contrived responses.

EmbeddedAppSpec

```
def 'can create simple hello world'() {
  expect:
  GroovyEmbeddedApp.fromHandler { ❶
    render 'Hello Greach 2016!'
  } test {
    assert getText() == 'Hello Greach 2016!' ❷
  }
}
```

❶ Creates a full Ratpack server with a single handler

❷ Ratpack provides us with a `TestHttpClient` that is configured to submit requests to `EmbeddedApp`. When the closure is finished executing Ratpack will take care of cleaning up the `EmbeddedApp`.

6. TestHttpClient

For testing, Ratpack provides `TestHttpClient` which is a blocking, synchronous http client for making requests against a running `ApplicationUnderTest`. This is intentionally designed in order to make testing deterministic and predictable.

EmbeddedAppSpec

```
def 'demonstrate ByMethodSpec'() {
  given:
  GroovyEmbeddedApp app = GroovyEmbeddedApp.fromHandlers { ❶
      path {
        byMethod {
          get {
            render 'GET'
          }
          post {
```

```
            render 'POST'
          }
        }
      }
    }

  and:
  TestHttpClient client = app.httpClient ❷

  expect: ❸
  client.getText() == 'GET'
  client.postText() == 'POST'

  client.put().status.code == 405
  client.delete().status.code == 405

  cleanup: ❹
  app.close()
}
```

❶ Create `GroovyEmbeddedApp` from a chain

❷ Retrieve a configured `TestHttpClient` for making requests against the `EmbeddedApp`

❸ Make some assertions about the application as described by the chain

❹ Have Spock invoke `EmbeddedApp#close` to gracefully shutdown the server.

The `TestHttpClient` has some basic support for manipulating request configuration as well as handling redirects and cookies.

EmbeddedAppSpec

```
def 'should handle redirects and cookies'() {
  expect:
  GroovyEmbeddedApp.fromHandlers { ❶
    get {
      render request.oneCookie('foo') ?: 'empty'
    }
    get('set') {
      response.cookie('foo', 'foo')
      redirect '/'
    }
    get('clear') {
      response.expireCookie('foo')
      redirect '/'
    }
```

```
    } test {
      assert getText() == 'empty' ❷
      assert getCookies('/')*.name() == []
      assert getCookies('/')*.value() == []

      assert getText('set') == 'foo'
      assert getCookies('/')*.name() == ['foo']
      assert getCookies('/')*.value() == ['foo']

      assert getText() == 'foo'

      assert getText('clear') == 'empty'
      assert getCookies('/')*.name() == []
      assert getCookies('/')*.value() == []

      assert getText() == 'empty'
      assert getCookies('/')*.name() == []
      assert getCookies('/')*.value() == []
    }
  }
```

❶ Create sample app that reads and writes cookies

❷ Issue requests that ensures cookie setting/expiring and redirect functionality

7. Async Testing

Ratpack is asynchronous and non-blocking from the ground up. This means that not only is Ratpack's api asynchronous but it expects that your code should be asynchronous as well.

Let's say we have a `ProfileService` that's responsible for retrieving `Profile`s:

ProfileService.groovy

```
import groovy.transform.Canonical
import ratpack.exec.Operation
import ratpack.exec.Promise

class ProfileService {
  final List<Profile> profiles = []
  Promise<List<Profile>> getProfiles() {
    Promise.value(profiles)
  }

  Operation add(Profile p) {
    profiles.add(p)
    Operation.noop()
```

```
  }

  Operation delete() {
    profiles.clear()
    Operation.noop()
  }
}

@Canonical
class Profile {
  String role
  String token
}
```

If you were to test this Service without any assistance from Ratpack you will run into the well known `UnmanagedThreadException`:

```
ratpack.exec.UnmanagedThreadException: Operation attempted on non Ratpack managed
 thread
```

7.1. ExecHarness

`ExecHarness` is the utility that Ratpack provides to test any kind of asynchronous behavior. Unsurprisingly `ExecHarness` is also an `AutoCloseable`. It utilizes resources that manage an `EventLoopGroup` and an `ExecutorService` so it's important to make sure these resources get properly cleaned up.

ProfileServiceExec.groovy

```
import ratpack.exec.ExecResult
import ratpack.exec.Promise
import ratpack.test.exec.ExecHarness
import spock.lang.AutoCleanup
import spock.lang.Specification

class ProfileServiceSpec extends Specification {

  @AutoCleanup
  ExecHarness execHarness = ExecHarness.harness() ❶

  def 'can add/retrieve/remove profiles from service'() {
    given:
    ProfileService service = new ProfileService()

    when:
```

```
    ExecResult<Promise<List<Profile>>> result = execHarness.yield
 { service.profiles } ❷

    then:
    result.value == []

    when:
    execHarness.yield { service.add(new Profile(role: 'admin', token: 'secret')) }
    and:
    List<Profile> profiles = execHarness.yield { service.profiles }.value

    then: profiles == [new Profile(role: 'admin', token: 'secret')]

    when:
    execHarness.yield { service.delete() }

    then:
    execHarness.yield { service.profiles }.value == []

  }
}
```

❶ Create an `ExecHarness` and tell Spock to clean up when we are finished

❷ Use `ExecHarness#yield` to wrap all of our service calls so that our Promises and Operations can be resolved on a Ratpack managed thread.

8. Functional testing

8.1. MainClassApplicationUnderTest

`GroovyRatpackMainApplicationUnderTest`
: For testing `ratpack.groovy` backed applications

```
  @AutoCleanup
  @Shared
  GroovyRatpackMainApplicationUnderTest groovyScriptApplicationunderTest = new
 GroovyRatpackMainApplicationUnderTest()
```

`MainClassApplicationUnderTest`
: For testing class backed applications

```
  @AutoCleanup
```

```
  @Shared
  MainClassApplicationUnderTest mainClassApplicationUnderTest = new
 MainClassApplicationUnderTest(MainClassApp)
```

Our sample Ratpack application for testing:

ratpack.groovy

```
import static ratpack.groovy.Groovy.ratpack

ratpack {
  serverConfig {
    sysProps() ❶
    require('/api', ApiConfig) ❷
  }
  bindings {
    bind(ConfService) ❸
  }
  handlers {
    get { ConfService confService ->
      confService.conferences.map { ❹
        "Here are the best conferences: $it"
      } then(context.&render)
    }
  }
}
```

❶ Pull configuration from System properties

❷ Create an ApiConfig object and put into the registry

❸ Bind `ConfService` using Guice

❹ Use `ConfService` to retrieve list of awesome Groovy Conferences

ApiConfig.groovy

```
class ApiConfig {
  String url
}
```

Simple object to contain our configuration data related to an API

ConfService

```
import com.google.inject.Inject
import ratpack.exec.Promise
import ratpack.http.client.HttpClient
```

```
class ConfService {
  final HttpClient httpClient
  final ApiConfig apiConfig

  @Inject
  ConfService(ApiConfig apiConfig, HttpClient httpClient) { ❶
    this.apiConfig = apiConfig
    this.httpClient = httpClient
  }

  Promise<List<String>> getConferences() { ❷
    httpClient.get(new URI(apiConfig.url))
      .map { it.body }
      .map { it.text.split(',').collect { it } }
  }
}
```

❶ Receive `ApiConfig` and `HtpClient` from Guice

❷ Define an asynchronous service method to retrieve data from remote service

8.2. Configuration

We can take advantage of system properties to change how the Ratpack application configures its services.

FunctionalSpec.groovy

```
import ratpack.groovy.test.GroovyRatpackMainApplicationUnderTest
import ratpack.groovy.test.embed.GroovyEmbeddedApp
import ratpack.test.ApplicationUnderTest
import ratpack.test.http.TestHttpClient
import spock.lang.AutoCleanup
import spock.lang.Shared
import spock.lang.Specification

class FunctionalSpec extends Specification {

  @Shared
  @AutoCleanup
  ApplicationUnderTest aut = new GroovyRatpackMainApplicationUnderTest() ❶

  @Delegate
  TestHttpClient client = aut.httpClient ❷
```

```
  @Shared
  @AutoCleanup
  GroovyEmbeddedApp api = GroovyEmbeddedApp.fromHandler { ❸
    render 'Greach, GR8conf, Gradle Conf'
  }

  def setup() {
    System.setProperty('ratpack.api.url', api.address.toURL().toString()) ❹
  }

  def 'can get best conferences'() { ❺
    when:
    get()

    then:
    response.statusCode == 200

    and:
    response.body.text.contains('Greach')
  }
}
```

❶ Create our `ApplicationUnderTest` and tell Spock to clean up when we're done

❷ Retrieve `TestHttpClient` and make use of `@Delegate` to make tests very readable

❸ Create a simple service that response with a comma separated list of Groovy Conferences

❹ Set system property to point to our stubbed service

❺ Write a simple test to assure that our Ratpack app can make a succcessful call to the remote api

8.3. Impositions

`Impositions` allow a user to provide overrides to various aspects of the Ratpack application bootstrap phase.

- `ServerConfigImposition` allows to override server configuration
- `BindingsImposition` allows to provide Guice binding overrides
- `UserRegistryImposition` allows you to provide alternatives for items in the registry

ImpositionSpec

```
import ratpack.exec.Promise
```

```
import ratpack.groovy.test.GroovyRatpackMainApplicationUnderTest
import ratpack.impose.UserRegistryImposition
import ratpack.impose.ImpositionsSpec
import ratpack.test.ApplicationUnderTest
import ratpack.test.http.TestHttpClient
import spock.lang.AutoCleanup
import spock.lang.Shared
import spock.lang.Specification
import ratpack.guice.Guice

class ImpositionSpec extends Specification {

  @Shared
  @AutoCleanup
  ApplicationUnderTest aut = new GroovyRatpackMainApplicationUnderTest() {
    @Override
    protected void addImpositions(ImpositionsSpec impositions) { ❶
      impositions.add(
        UserRegistryImposition.of(Guice.registry {
          it.add(new ConfService(null, null) {
            Promise<List<String>> getConferences() {
              Promise.value(['Greach'])
            }
          })
        }))
    }
  }

  @Delegate
  TestHttpClient client = aut.httpClient

  def 'can get list of gr8 conferences'() {
    when:
    get()

    then:
    response.statusCode == 200

    and:
    response.body.text.contains('Greach')
  }
}
```

❶ Override `addImpositions` method to provide a `UserRegistryImposition` that supplies our own dumb implementation of `ConfService` that does not need to make any network connections

8.4. RemoteControl

Authored by Luke Daley; originally for Grails

Used to serialize commands to be executed on the `ApplicationUnderTest`

build.gradle

```
testCompile ratpack.dependency('remote-test')
```

Here we add a test compile dependency on `io.ratpack:ratpack-remote-test` which includes a dependency on `remote-control`

RemoteControlSpec.groovy

```
import io.remotecontrol.client.UnserializableResultStrategy
import ratpack.groovy.test.GroovyRatpackMainApplicationUnderTest
import ratpack.guice.BindingsImposition
import ratpack.impose.ImpositionsSpec
import ratpack.remote.RemoteControl
import ratpack.test.ApplicationUnderTest
import ratpack.test.http.TestHttpClient
import spock.lang.AutoCleanup
import spock.lang.Shared
import spock.lang.Specification

class RemoteControlSpec extends Specification {

  @Shared
  @AutoCleanup
  ApplicationUnderTest aut = new GroovyRatpackMainApplicationUnderTest() {
    @Override
    protected void addImpositions(ImpositionsSpec impositions) {
      impositions.add(BindingsImposition.of {
        it.bindInstance RemoteControl.handlerDecorator() ❶
      })
    }
  }

  @Delegate
  TestHttpClient client = aut.httpClient

  ratpack.test.remote.RemoteControl remoteControl = new
ratpack.test.remote.RemoteControl(aut, UnserializableResultStrategy.NULL) ❷
```

```
  def 'should render profiles'() {
    when:
    get()

    then:
    response.body.text == '[]'

    when:
    remoteControl.exec { ❸
      get(ProfileService)
        .add(new Profile('admin'))
    }

    and:
    get()

    then:
    response.body.text.startsWith('[{')
  }
}
```

❶ We use `BindingsImposition` here to add a hook into the running `ApplicationUnderTest` that allows us to run remote code on the server

❷ We tell `RemoteControl` not to complain if the result of the command is not Serializable

❸ We use remote control here to grab the `ProfileService` and manually add a profile

8.5. EphemeralBaseDir

A utility that provides a nice way to interact with files that would provide the basis of a base directory for Ratpack applications. It is also an `AutoCloseable` so you'll need to make sure to clean up after use.

EphemeralSpec.groovy

```
import ratpack.groovy.test.embed.GroovyEmbeddedApp
import ratpack.test.embed.EphemeralBaseDir
import spock.lang.Specification

import java.nio.file.Path

class EphemeralSpec extends Specification {
  def 'can supply ephemeral basedir'() {
    expect:
    EphemeralBaseDir.tmpDir().use { baseDir ->
      baseDir.write("mydir/.ratpack", "")
```

```
      baseDir.write("mydir/assets/message.txt", "Hello Ratpack!")
      Path mydir = baseDir.getRoot().resolve("mydir")

      ClassLoader classLoader = new URLClassLoader((URL[])
 [mydir.toUri().toURL()].toArray())
      Thread.currentThread().setContextClassLoader(classLoader);

      GroovyEmbeddedApp.of { serverSpec ->
        serverSpec
          .serverConfig { c -> c.baseDir(mydir) }
          .handlers { chain ->
          chain.files { f -> f.dir("assets") }
        }
      }.test {
        String message = getText("message.txt")
        assert "Hello Ratpack!" == message
      }
    }
  }
}
```

9. Resources

- (book) O'Reilly: Learning Ratpack by Dan Woods[1]
- (javadocs) Ratpack Test[2]
- (javadocs) Ratpack Groovy Test[3]
- (javadocs) Ratpack Remote[4]
- (javadocs) Ratpack Remote Test[5]

[1] http://shop.oreilly.com/product/0636920037545.do
[2] https://ratpack.io/manual/current/api/ratpack/test/package-summary.html
[3] https://ratpack.io/manual/current/api/ratpack/groovy/test/package-summary.html
[4] https://ratpack.io/manual/current/api/ratpack/remote/package-summary.html
[5] https://ratpack.io/manual/current/api/ratpack/test/remote/package-summary.html