QINIU
七牛云存储

七牛如何做HTTP服务测试？

许式伟

2015-4-18

HTTP服务测试

- 单元测试
  - 某个独立子服务的测试
- 集成测试
  - 整个集群对外业务API的测试
    - Stage环境
    - Product环境

怎么测？

- 七牛早期做法
  - 实现服务逻辑（Service Implementation）
  - 实现客户端SDK（Client Implementation）
  - 基于客户端SDK写测试案例（Test Case）
- 问题
  - 客户端SDK修改导致测试案例编不过
  - 客户端SDK通常是使用方友好，而不是测试方友好
  - 让服务端与客户端SDK耦合，容易过早陷入客户端SDK如何抽象更合理的细节，而不能专注于测试服务逻辑本身

换个角度

- 直接基于协议测试呢？
  - 比如，基于 http.Client 类直接写测试案例
- 问题
  - 代码相对冗长
  - 业务逻辑表达不直观
    - 写一些辅助函数能够略为改观，不过会有逐步写测试专用SDK的倾向

七牛当前做法

- 引入 httptest DSL 文法
- 更接近基于 http.Client 写测试案例的思路
  - 但努力让代码更直白体现测试用意

Hello, world!

QINIU 七牛云存储

hello.qtf ● quick_start.qtf ×

```
1  #!/usr/bin/env qiniutest
2
3  #
4  # 这个例子算 qiniu httptest 工具的 Hello world 程序吧。
5  # 执行预期：下载 www.qiniu.com 首页，要求返回 200。
6  # 如果返回非 200，测试失败；否则测试通过，并打印返回的 response body（对于测试来说无价值，通常用于调试）。
7  #
8
9  get http://www.qiniu.com/
10 ret 200
11 echo $(resp.body)
12
```

Quick start

hello.qtf ● quick_start.qtf ×

```
#!/usr/bin/env qiniutest

#
# qiniutest 整体基于命令行文法。其中 `...` 代表子命令，'...' 或 "" 方便传递复杂参数。
# 以下是单HTTP请求的测试文法。如果你有 HTTP 协议的基础，理解这样一段测试代码所代表的含义并不困难：
#
# req <http-method> <url>
# header <key1> <val11> <val12> ...
# header <key2> <val21> <val22> ...
# auth <authorization>
# body <content-type> <body-data>
# ret <expected-status-code>
# header <resp-key1> <expected-val11> <expected-val12> ...
# header <resp-key2> <expected-val21> <expected-val12> ...
# body <expected-content-type> <expected-body-data>
#
# 上面的 req 和 body 指令，有诸多简写形式。比如：
#
# req GET http://www.qiniu.com/ 可以简写为：get http://www.qiniu.com/
# body 'application/json' '{"a": 1, "b": 2}' 可以简写为：json '{"a": 1, "b": 2}'
#

auth qiniutest `qiniu <AccessKey> <SecretKey>` #预先给auth取个别名，只是为了让下面写auth语句可以更简洁一些

post http://foo.com/objects
auth qiniutest  #等价于：  auth `qiniu <AccessKey> <SecretKey>`
json '{
    "a": "value1", "b": 1
}'
ret 200
json '{
    "id": $(id1)  #重要！暂时先体会下，后面有详细的机制解析
}'

get http://foo.com/objects/$(id1)
auth qiniutest
ret 200
json '{
    "a": "value1", "b": 1
}'

```

语法结构

- 基于命令行文法

  command switch1 switch2 … arg1 arg2 …

- 转义
  - 如果参数包含空格或其他特殊字符，则可以：
    - 用 \ 转义
      - 比如 '\ ' 表示 ' ' (空格)，'\t' 表示 TAB 字符，等等
    - 用 '…' 或 "…" 包含
      - '…' 中不支持用 \ 转义，也不支持子命令，出现任何内容都当作普通字符对待

语法结构

- 区别于Linux Shell的地方
  - 参数类型不只是字符串，有完整类型系统（支持且仅支持所有json的数据类型）
    - string (如："a"、application/json)，在不引起歧义的情况下，可以省略双引号
    - number (如：3.14159)
    - boolean (如：true)
    - array (如：["a", 200, {"b": 2}])
    - dictionary/object (如：{"a": 1, "b": 2})
  - 子命令相当于函数，返回任意类型的数据
    - 比如 `qiniu f2weae23e6c9f jg35fae526kbce` 返回一个 auth object，用字符串无法表达

http Request 的表达

req <http-method> <url>
header <key1> <val11> <val12> ...
header <key2> <val21> <val22> ...
auth <authorization>
body <content-type> <body-data>

样例

- 无授权的GET请求

```
req GET http://www.qiniu.com/
```

- 带授权的POST请求

```
req POST http://foo.com/objects
auth `qiniu f2weae23e6c9f jg35fae526kbce`
body application/json '{
    "a": "hello1", "b": 2
}'
```

简写

- 无授权的GET请求

  get http://www.qiniu.com/

- 带授权的POST请求

  post http://foo.com/objects

  auth `qiniu f2weae23e6c9f jg35fae526kbce`

  json '{

  "a": "hello1", "b": 2

  }'

http Response 匹配

ret <expected-status-code>
header <key1> <expected-val11> <expected-val12> ...
header <key2> <expected-val21> <expected-val22> ...
body <expected-content-type> <expected-body-data>

ret 指令

- ret
  - 发起 http Request 请求，并将 http Response 存储到 $(resp) 变量中

- ret <expected-status-code>
  - 等价于

    ret

    match <expected-status-code> $(resp.code)

匹配(match)

- 这几乎是这套 DSL 中最核心的概念
  - match <expected> <source>
    - 要求 <expected> 必须和 <source> **匹配**
    - <source> 中不允许出现未绑定的变量
    - <expected> 中允许存在未绑定的变量
      - 如果 <expected> 中出现了已绑定的变量，则要求该变量必须匹配 <source> 中对应的值
      - 如果 <expected> 中出现了未绑定的变量，则该变量会被赋值为 <source> 中对应的值
  - **匹配**
    - 对于 number/string/boolean/array 类型
      - match A B 意味着要求 A == B
    - 对于 object(dictionary) 类型
      - match A B 意味着 A 中出现的 item，在 B 中必须出现并且匹配

例子

- 例子1

  match $(a.b) 1

  match $(a.c) hello

  结果 $(a) 的值为 {"b": 1, "c": "hello"}

- 例子2

  match $(a.b) 1

  match $(a.b) 1　#可以匹配，因为$(a.b)的值的确为1

  match $(a.b) 2　#失败，1和2不相等

- 例子3

  match '{"c": {"d": $(d)}}' '{"c": {"d": "hello", "e": "world"}, "f": 1}'

  结果 $(d) 的值为 "hello"

理解 http Response 匹配

- ret <expected-status-code>
  - 等价于

    ret

    match <expected-status-code> $(resp.code)
- header <key> <expected-val1> <expected-val2> ...
  - 等价于

    match '[<expected-val1>, <expected-val2>, ...]' $(resp.header.<key>)
- body <expected-content-type> <expected-body-data>
  - 等价于

    match '[<expected-content-type>]' $(resp.header.Content-Type)

    match <expected-body-data> $(resp.body)

断言

- equal <expected> <source>
  - 与 match 不同，<expected>, <source> 中都不允许出现未绑定的变量
  - 与 match 不同，equal 要求 <expected>, <source> 的值精确相等
- equalSet <expected> <source>
  - Set 是指集合
  - 与 equal 不同，equalSet 要求 <expected>, <source> 都是 array，并且对 array 的元素进行排序后两者精确相等

例子

- 例子1

```
get http://foo.com/objects/a325gea2kgfd
auth qiniutest
ret 200
equal '{"a": "hello1", "b": 2}' $(resp.body)
match '{"a": "hello1", "b": 2}' $(resp.body)
```

- 例子2

```
get http://foo.com/objects
auth qiniutest
ret 200
equalSet $(resp.body), '[
    {"a": "hello1", "b": 2},
    {"a": "world2", "b": 4}
]'
```

测试环境

- 如何让 stage 和 product 环境共享测试案例？
  - 测试环境参数化
  - 也方便测试脚本入库（不入库 User/Password、AK/SK 这种敏感信息）

host 指令

- 服务地址参数化

```
host foo.com 127.0.0.1:8888

get http://foo.com/objects/a325gea2kgfd
auth qiniutest
ret 200
json '{"a": "hello1", "b": 2}'
```

Host、AK/SK 参数化

```
match $(testenv) `env QiniuTestEnv`
match $(env) `envdecode QiniuTestEnv_$(testenv)`

host foo.com $(env.FooHost)
auth qiniutest `qiniu $(env.AK) $(env.SK)`

post http://foo.com/objects
auth qiniutest
json '{"a": "hello1", "b": 2}'
ret 200
json '{"id": $(id1)}'

get http://foo.com/objects/$(id1)
auth qiniutest
ret 200
json '{"a": "hello1", "b": 2}'
```

env、envdecode 指令

- env <key>
  - 取环境变量 <key> 对应的值
- decode <encodedval>
  - 将一个 json 字符串 decode 为对象(object)
- envdecode <key>
  - 取环境变量 <key> 对应的值，并且把它当做 json 字符串 decode 为一个对象(object)
  - 等价于

```
match $(encodedval) `env <key>`
decode $(encodedval)
```

配置测试环境

- 配置 stage、product 环境

```
export QiniuTestEnv_stage='{
    "FooHost": "192.168.1.10:8888",
    "AK": "...",
    "SK": "..."
}'

export QiniuTestEnv_product='{
    "FooHost": "foo.com",
    "AK": "...",
    "SK": "..."
}'
```

执行测试案例

- 测试 stage 环境

  QiniuTestEnv=stage qiniutest ./testfoo.qtf

- 测试 product 环境

  QiniuTestEnv=product qiniutest ./testfoo.qtf

未覆盖的内容...

- echo/println – 打印变量内容(调试)
- auth – 授权的详细解剖
- base64 – 对字符串 base64 encode/decode
- case/setUp/tearDown – 如何定义多个案例
- ...

Q & A

QINIU
七牛云存储

@许式伟

@七牛云存储