

R 语言模型部署实战

徐静

2018-12-31

Contents

序言	5
关于我	7
1 httpuv	9
1.1 方法介绍	9
1.2 例子演示	12
2 opencpu	17
2.1 安装	17
2.2 API 文档	18
2.3 XGBoost 模型部署实例	26
3 plumber	33
3.1 快速开始	34
3.2 路由和输入	35
3.3 Rendering Output	39
3.4 部署	44
3.5 其他	48
4 jug	55
4.1 What is jug?	55
4.2 Install and Hello World	55
4.3 Middleware (中间件)	56
4.4 预定义的中间件	58
4.5 事件监听	60
4.6 预定义的事件侦听器	61
4.7 请求, 响应和错误对象	61
4.8 URL 调度	62
4.9 启动 jug 实例	62
4.10 线性回归模型的 API 举栗	62
4.11 官方栗子	64
5 fiery	65
6 Rserve	67
7 RestRserve	69
8 mailR	71
9 Rweixin(*)	75
10 参考文献	77
10.1 opencpu	77

10.2 plumber	77
10.3 jug	77

序言

我们的模型不能只停留在线下的分析报告中，训练好的 R 模型如何应用到生产环境？目前针对于 R 语言的模型生产环境应用的方式有很多，比如用其他语言去调用，Java, Python 等语言均可方便的调用 R 脚本；生成 PMML 文件，目前 R 中主流的一些 R 模型均支持 PMML 比如 xgboost,lightGBM 等，其他语言不需要调用 R 脚本只需调用统一的 PMML 文件就可以；还有就是 Web 端的部署，比如可以做成 REST API 供其他语言调用，或直接做成 web 应用供其他用户访问，本书主要针对于 R 语言模型的 Web 端的部署。过程中，我们会先后介绍 httpuv,opencpu,plumber,jug,fiery,Rserve,RestRserve, 等一些和模型线上化部署相关的 R 包（当然 shiny 也可以，但他不是我们本书的重点），最后会介绍 mailR 和 Rweixin 两个 R 和邮件与微信通信的 R 包，用于线上化部署的监测。当然会有其他的线上化部署方式。

欢迎进入 R 模型线上化部署的海洋！

关于我

徐静：

硕士研究生, 目前的研究兴趣主要包括: 数理统计, 统计机器学习, 深度学习, 网络爬虫, 前端可视化, R 语言和 Python 语言的超级粉丝, 多个 R 包和 Python 模块的作者, 现在正逐步向 Java 迁移。

Graduate students, the current research interests include: mathematical statistics, statistical machine learning, deep learning, web crawler, front-end visualization. He is a super fan of R and Python, and the author of several R packages and Python modules, and now gradually migrating to Java.

Chapter 1

httpuv

在 httpuv 的官网中, 有这么一段描述:

Allows R code to listen for and interact with HTTP and WebSocket clients, so you can serve web traffic directly out of your R process. Implementation is based on libuv and http-parser.

This is a low-level library that provides little more than network I/O and implementations of the HTTP and WebSocket protocols. For an easy way to create web applications, try Shiny instead.

我们可以通过 httpuv 搭建一个访问 R 模型的 web API, 但可能这不是最好的。

本部分我们首先介绍官方提供的一些方法, 然后解析官方提供的演示 Demo, 从而达到熟练使用 httpuv 的目的。

1.1 方法介绍

下面我们解析一下 httpuv 官方提供的一些调用方法, 并演示一些调用方法对应的实例

1. 使用 URI 编码/解码以与 Web 浏览器相同的方式对字符串进行编码/解码。

```
encodeURI (value)
encodeURIComponent (value)
decodeURI (value)
decodeURIComponent (value)
```

参数列表

- value 用于编码和解码的字符向量, UTF-8 字符编码

```
library (httpuv)
value <- "https://baidu.com/中国;?/"
encodeURI (value)

## [1] "https://baidu.com/%D6%D0%B9%FA;?/"

encodeURIComponent (value)

## [1] "https%3A%2F%2Fbaidu.com%2F%D6%D0%B9%FA%3B%3F%2F"

decodeURI (value)

## [1] "https://baidu.com/中国;?/"

decodeURIComponent (value)

## [1] "https://baidu.com/中国;?/"
```

注意: encodeURI 与 encodeURIComponent 是不一样的因为前者不对特殊字符: ;,/?:@&=+\$ 等进行 encode

2. 中断 httpuv 运行的环路

```
interrupt()
```

3. 检查 ip 地址的类型是 ipv4 还是 ipv6

```
ipFamily(ip)
```

参数列表

- ip 一个代表 IP 地址的字符串
- 返回值的意义: 如果是 IPv4 返回 4, 如果是 IPv6 返回 6, 如果不是 IP 地址返回 -1

```
ipFamily("127.0.0.1") # 4
```

```
## [1] 4
```

```
ipFamily("500.0.0.500") # -1
```

```
## [1] -1
```

```
ipFamily("500.0.0.500") # -1
```

```
## [1] -1
```

```
ipFamily(":::") # 6
```

```
## [1] 6
```

```
ipFamily(":::1") # 6
```

```
## [1] 6
```

3. 将原始向量转换为 BASE64 编码字符串

```
rawToBase64(x)
```

参数列表

- x 原始向量

```
set.seed(100)
result <- rawToBase64(as.raw(runif(19, min=0, max=256)))
#stopifnot(identical(result, "TkGNDnd7z16LK5/hR2bDqzRbXA=="))
result
```

```
## [1] "TkGNDnd7z16LK5/hR2bDqzRbXA=="
```

4. 运行一个 server

```
runServer(host, port, app, interruptIntervalMs = NULL)
```

参数列表

- host IPv4 地址, 或是 “0.0.0.0” 监听所有的 IP
- port 端口号
- app 一个定义应用的函数集合
- interruptIntervalMs 该参数不提倡使用, 1.3.5 版本后废除

```
app <- list(call = function(req) {
  list(status=200L,
    headers = list(
      'Content-Type' = 'text/html'
    ),
    body = "HelloWorld!")
}

runServer("0.0.0.0", 5000, app)
```

5. 过程请求

处理 HTTP 请求和 WebSocket 消息。如果 R 的调用堆栈上没有任何东西，如果 R 是在命令提示符下闲置，不必调用此函数，因为请求将自动处理。但是，如果 R 正在执行代码，则请求将不被处理。要么调用栈是空的，要么调用这个函数(或者，调用 run_now())。

```
service(timeoutMs = ifelse(interactive(), 100, 1000))
```

参数列表

- timeoutMs 返回之前运行的毫秒数。

6. 创建 HTTP/WebSocket 后台服务器（弃用）

```
startDaemonizedServer(host, port, app)
```

7. 创建 HTTP/WebSocket 服务器

```
startServer(host, port, app)
startPipeServer(name, mask, app)
```

参数列表

- host ip 地址
- port 端口号
- app 一个定义应用的函数集

```
app <- list(
  call = function(req) {
    list(
      status = 200L,
      headers = list(
        'Content-Type' = 'text/html'
      ),
      body = "Hello world!"
    )
  }
)
handle <- startServer("0.0.0.0", 5000, app)

# 此服务器的句柄，可以传递给 stopServer 以关闭服务器。
stopServer(handle)
```

8. 停止所有应用

```
stopAllServers()
```

9. 在 UNIX 环境中停止运行的后台服务器（弃用）

```
stopDaemonizedServer(handle)
```

10. 停止一个服务

```
stopServer(handle)
```

1.2 例子演示

1. json-server

```
# Connect to this using websockets on port 9454
# Client sends to server in the format of {"data": [1,2,3]}
# The websocket server returns the standard deviation of the sent array
library(jsonlite)
library(httpuv)

# Server
app <- list(
  onWSOpen = function(ws) {
    ws$onMessage(function(binary, message) {
      # Decodes message from client
      message <- fromJSON(message)
      # Sends message to client
      ws$send(
        # JSON encode the message
        toJSON(
          # Returns standard deviation for message
          sd(message$data)
        )
      )
    })
  }
)
runServer("0.0.0.0", 9454, app, 250)
```

2.echo

```
library(httpuv)

app <- list(
  call = function(req) {
    wsUrl = paste(sep = "'",
                  "'",
                  "ws://",
                  ifelse(is.null(req$HTTP_HOST), req$SERVER_NAME, req$HTTP_HOST),
                  "'")
  }
)
list(
  status = 200L,
  headers = list(
    'Content-Type' = 'text/html'
  ),
  body = paste(
    sep = "\r\n",
```

```

    "<!DOCTYPE html>,
    "<html>,
    "<head>,
    '<style type='text/css'>',
    'body { font-family: Helvetica; }',
    'pre { margin: 0 }',
    '</style>',
    "<script>",
    sprintf("var ws = new WebSocket(%s);", wsUrl),
    "ws.onmessage = function(msg) {",
    '  var msgDiv = document.createElement('pre')',
    '  msgDiv.innerHTML = msg.data.replace(/&/g, "&").replace(/\\"/g, "<");',
    '  document.getElementById("output").appendChild(msgDiv);',
    "}",
    "function sendInput() {",
    "  var input = document.getElementById('input');",
    "  ws.send(input.value);",
    "  input.value = '';",
    "}",
    "</script>",
    "</head>",
    "<body>",
    '<h3>Send Message</h3>',
    '<form action="" onsubmit="sendInput(); return false">',
    '<input type="text" id="input"/>',
    '<h3>Received</h3>',
    '<div id="output"/>',
    '</form>',
    "</body>",
    "</html>"
  )
)
},
onWSOpen = function(ws) {
  ws$onMessage(function(binary, message) {
    ws$send(message)
  })
}
)

browseURL("http://localhost:9454/")
runServer("0.0.0.0", 9454, app, 250)

```

3.deamon-echo

```

library(httpruv)

.lastMessage <- NULL

app <- list(
  call = function(req) {
    wsUrl = paste(sep='',
      "'",

```

```

    "ws://",
    ifelse(is.null(req$HTTP_HOST), req$SERVER_NAME, req$HTTP_HOST),
    '')
}

list(
  status = 200L,
  headers = list(
    'Content-Type' = 'text/html'
  ),
  body = paste(
    sep = "\r\n",
    "<!DOCTYPE html>",
    "<html>",
    "<head>",
    '<style type="text/css">',
    'body { font-family: Helvetica; }',
    'pre { margin: 0 }',
    '</style>',
    "<script>",
    sprintf("var ws = new WebSocket(%s);", wsUrl),
    "ws.onmessage = function(msg) {",
    '  var msgDiv = document.createElement("pre");',
    '  msgDiv.innerHTML = msg.data.replace(/&/g, "&").replace(/\</g, "<");',
    '  document.getElementById("output").appendChild(msgDiv);',
    "}",
    "function sendInput() {",
    "  var input = document.getElementById('input');",
    "  ws.send(input.value);",
    "  input.value = '';;",
    "}",
    "</script>",
    "</head>",
    "<body>",
    '<h3>Send Message</h3>',
    '<form action="" onsubmit="sendInput(); return false">',
    '<input type="text" id="input"/>',
    '<h3>Received</h3>',
    '<div id="output"/>',
    '</form>',
    "</body>",
    "</html>"
  )
)
},
onWSOpen = function(ws) {
  ws$onMessage(function(binary, message) {
    .lastMessage <- message
    ws$send(message)
  })
}
)

server <- startDaemonizedServer("0.0.0.0", 9454, app)

```

```
# check the value of .lastMessage after echoing to check it is being updated
# call this after done
#stopDaemonizedServer(server)
```

```
4.

library(httputv)
app = list(call = function(req) {
  # 获取 POST 的参数
  postdata = req$rook$input$read_lines()
  qs = httr:::parse_query(gsub("^\\?\"", "", postdata))
  dat = jsonlite::fromJSON(qs$jsonData)
  print(dat)
  # 计算返回结果
  r = 0.3 + 0.1 * dat$v1 - 0.2 * dat$v2 + 0.1 * dat$v3
  output = jsonlite:::toJSON(list(message = 'suceess', result = r), auto_unbox = T)
  res = list(status = 200L, headers = list('Content-Type' = 'application/json'), body = o
      return(res)
})

# 启动服务
server = startServer("0.0.0.0", 1124L, app = a
while(TRUE) {
  service()
  Sys.sleep(0.001)
}
# stopServer(server)

RCurl:::postForm('127.0.0.1:1124',
style = 'post',
.params = list(jsonData = '{"v1":1,"v2":2,"v3":3}') )
```

httputv 是相对比较底层的包，熟练使用需要掌握前端知识，并且需要用到 RCurl, httr 相关爬虫包的一些知识去处理。本人不推荐这种方式进行模型的部署。

Chapter 2

opencpu

2.1 安装

关于 opencpu-server 的安装和本地单用户服务器的安装可以参考 <https://www.opencpu.org/download.html>, opencpu 官方推荐在 Ubuntu 18.04 / 16.04 中使用 opencpu-server, 关于 Fedora,Debian,CentOS 等系统的安装可以参考<https://github.com/opencpu>

2.1.1 Ubuntu 18.04/16.04 安装

```
# Requires Ubuntu 18.04 (Bionic) or 16.04 (Xenial)
sudo add-apt-repository -y ppa:opencpu/opencpu-2.1
sudo apt-get update
sudo apt-get upgrade

# Installs OpenCPU server
sudo apt-get install -y opencpu-server
# Done! Open http://yourhost/ocpu in your browser

# rstudio server 的安装 (不是必须的)
sudo apt-get install -y rstudio-server

# 安装完之后, 需要检查自己 R 中的 curl 包及 stringi 等包的版本,
# 同时如果需要 R markdown 还需要安装 pandoc

sudo apt-get install pandoc
```

2.1.2 本地单用户服务器

```
# Install OpenCPU
install.packages("opencpu")

# Run Apps directly from Github
library(opencpu)
ocpu_start_app("rwebapps/nabel")
ocpu_start_app("rwebapps/markdownapp")
ocpu_start_app("rwebapps/stockapp")
```

```
# Run Apps directly from library
library(opencpu)
ocpu_start_server()

# Install / remove apps
remove_apps("rwebapps/stockapp")
```

2.2 API 文档

2.2.1 OpenCPU 中的 HTTP

OpenCPU 根路径

API 的 root 是动态的。默认为 /ocpu/，可以更改此设置。通过 R 启动的本地服务，可以通过 `ocpu_start_server(port = 5656, root = "/ocpu", workers = 2, preload = NULL, on_startup = NULL, no_cache = FALSE)` 中的 `root` 参数修改，`opencpu-server` 可以通过修改 /usr/lib/opencpu/rapache 中的文件进行修改。在下面的示例中，我们假设默认值 /ocpu/。

Debug

- `http://172.16.100.202/ocpu/info` opencpu 的一些详细的信息
- 该 `http://172.16.100.202/ocpu/test` URL 为您提供了一个方便的测试网页来执行服务器请求

Http 的方法

OpenCPU 目前只使用 HTTP 方法 GET 和 POST。GET 用于检索资源，POST 用于 RPC。POST 请求仅对脚本或功能 URL 有效。

Method	Target	Action	Arguments	Example
GET	object	read object	control output format	<code>GET /ocpu/library/MASS/R/cats/json</code>
POST	object	call function	function arguments	<code>POST /ocpu/library/stats/R/rnorm</code>
GET	file	read file	-	<code>GET /ocpu/library/MASS/NEWS</code> <code>GET /ocpu/library/MASS/scripts/</code>
POST	file	run script	control interpreter	<code>POST /ocpu/library/MASS/scripts/ch01.R</code> <code>POST /ocpu/library/knitr/examples/knitr-minimal.Rmd</code>

```
# Get 举栗
curl http://172.16.100.202/ocpu/library/MASS/data/Boston/json
curl http://172.16.100.202/ocpu/library/MASS/NEWS
curl http://172.16.100.202/ocpu/library/MASS/scripts/

# Post 举栗 -X POST or -d "arg=value"
curl http://172.16.100.202/ocpu/library/MASS/scripts/ch01.R -X POST
curl http://172.16.100.202/ocpu/library/stats/R/rnorm -d "n=10&mean=5"
```

Http 状态码

这些是 OpenCPU 返回的常见状态代码，客户端应该能够解释这些状态代码

HTTP Code	When	Returns
200 OK	On successful GET request	Resource content
201 Created	On successful POST request	Output location
302 Found	Redirect	Redirect Location
400 Bad Request	R raised an error.	Error message in <code>text/plain</code>
502 Bad Gateway	Nginx (opencpu-cache) can't connect to OpenCPU server.	(admin needs to look in error logs)
503 Bad Request	Serious problem with the server	(admin needs to look in error logs)

2.2.2 API 端点 (EndPoints)

The API Libraries

路径	什么
<code>/ocpu/library/{pkgnname}/</code>	R软件包安装在服务器上的一个全局库中。
<code>/ocpu/apps/{gituser}/{reponame}/</code>	<code>{reponame}</code> 从github用户安装在存储库根目录中的R包接口 <code>{gituser}</code> 。
<code>/ocpu/user/{username}/library/{pkgnname}/</code>	R包安装在Linux用户的主库中 <code>{username}</code> 。
<code>/ocpu/tmp/{key}/</code>	临时会话，用于保存函数/脚本RPC的输出。

```
#read packages 举栗
curl http://172.16.100.202/ocpu/library/
curl http://172.16.100.202/ocpu/apps/rwebapps/
curl http://172.16.100.202/ocpu/user/jeroen/library/
```

```
#read session 举栗
curl http://172.16.100.202/ocpu/tmp/x2c5ab8d4d6
```

The R package API

`/{package}/`库都支持以下端点：

路径	什么
<code>../{pkgname}/info</code>	显示有关此包的信息。
<code>../{pkgname}/www/</code>	包中包含的应用程序（如果有）
<code>../{pkgname}/R/</code>	此包导出的R对象。请参阅R对象API。
<code>../{pkgname}/data/</code>	此软件包中包含的数据。数据集是对象，请参阅R对象API。
<code>../{pkgname}/man/</code>	本软件包中包含的手册（帮助页面）。
<code>../{pkgname}/man/{topic}/{format}</code>	<code>topic</code> 以输出格式检索帮助页面 <code>format</code> 。手册的格式必须是一个 <code>text</code> , <code>html</code> , <code>pdf</code>
<code>../{pkgname}/html</code>	模拟R-base html帮助页面（为了向后兼容）。
<code>../{pkgname}/*</code>	对于其他所有，包安装目录中的文件的接口。

```
#package info
curl http://172.16.100.202/ocpu/library/MASS/

#package objects (mostly functions)
curl http://172.16.100.202/ocpu/library/MASS/R/
curl http://172.16.100.202/ocpu/library/MASS/R/rlm/print

#package data objects
curl http://172.16.100.202/ocpu/library/MASS/data/
curl http://172.16.100.202/ocpu/library/MASS/data/housing/json

#read manuals pages
curl http://172.16.100.202/ocpu/library/MASS/man/
curl http://172.16.100.202/ocpu/library/MASS/man/rilm/text
curl http://172.16.100.202/ocpu/library/MASS/man/rilm/html
curl http://172.16.100.202/ocpu/library/MASS/man/rilm/pdf

#read files included with this package
curl http://172.16.100.202/ocpu/library/MASS/scripts/
curl http://172.16.100.202/ocpu/library/MASS/scripts/ch01.R
curl http://172.16.100.202/ocpu/library/MASS/NEWS

#call a function (example from 'rlm' help page)
curl http://172.16.100.202/ocpu/library/MASS/R/rlm -d "formula=stack.loss ~ .&data=stackloss" -X POST

#run R script
curl http://172.16.100.202/ocpu/library/MASS/scripts/ch01.R -X POST

#read output (replace key with value returned from previous request)
curl http://172.16.100.202/ocpu/tmp/x0648ec526b/R/.val/print
```

The R object API

/R API 用来读取 R 对象或者调用 R 的方法。

路径	什么
<code>./R/</code>	列出此包或会话中的R对象。
<code>./data/</code>	列出包中的数据对象。
<code>./{R data}/{object}</code>	以默认格式读取对象。如果 <code>object</code> 是函数，则可以使用HTTP POST调用它。
<code>./{R data}/{object}/{format}</code>	以特定输出格式检索R对象（请参阅输出格式部分）。

```
#list objects and datasets from MASS
curl http://172.16.100.202/ocpu/library/MASS/R/
curl http://172.16.100.202/ocpu/library/MASS/data/

#retrieve objects
curl http://172.16.100.202/ocpu/library/MASS/R/truehist/print
curl http://172.16.100.202/ocpu/library/MASS/data/bacteria/print
curl http://172.16.100.202/ocpu/library/MASS/data/bacteria/json
curl http://172.16.100.202/ocpu/library/MASS/data/bacteria/csv
curl http://172.16.100.202/ocpu/library/MASS/data/bacteria/rda

#call a function
curl http://172.16.100.202/ocpu/library/MASS/R/truehist -d "data=[1,3,7,4,2,4,2,6,23,13,5]
```

The R session API

会话(session)是一个容器，用于保存从远程函数/脚本调用（RPC）创建的资源。

路径	什么
<code>/ocpu/tmp/{key}/</code>	列出此会话的可用输出。
<code>/ocpu/tmp/{key}/R</code>	存储在此会话中的R对象。使用 R对象API的接口 ，与包中的对象相同。
<code>/ocpu/tmp/{key}/graphics/</code>	存储在此会话中的图形（图表）。
<code>/ocpu/tmp/{key}/graphics/{n}/{format}</code>	{n} 以输出格式检索绘图编号 {format}。格式通常是一个 <code>png</code> 、 <code>pdf</code> 或 <code>svg</code> 。该 {n} 是一个整数或“最后一个”。
<code>/ocpu/tmp/{key}/source</code>	读取此会话的输入源代码。
<code>/ocpu/tmp/{key}/stdout</code>	显示在此会话中打印到STDOUT的文本。
<code>/ocpu/tmp/{key}/console</code>	显示此会话的控制台输入/输出（组合源和标准输出）
<code>/ocpu/tmp/{key}/zip</code>	以zip存档的形式下载整个会话。
<code>/ocpu/tmp/{key}/tar</code>	以gzip压缩包下载整个会话。
<code>/ocpu/tmp/{key}/files/*</code>	在会话的工作目录中与文件API的接口。

```
#A POST will create a temporary session
#Use the returned key for the subsequent calls below
curl http://172.16.100.202/ocpu/library/MASS/scripts/ch01.R -X POST

#Look at console input/output
```

```

curl http://172.16.100.202/ocpu/tmp/x05b85461/console/text

#We read session R objects
curl http://172.16.100.202/ocpu/tmp/x05b85461/
curl http://172.16.100.202/ocpu/tmp/x05b85461/R/
curl http://172.16.100.202/ocpu/tmp/x05b85461/R/dd/csv
curl http://172.16.100.202/ocpu/tmp/x05b85461/R/t.stat/print

#Or even call a function
curl http://172.16.100.202/ocpu/tmp/x05b85461/R/t.stat -d "x=[1,0,0,1,1,1,0,1,1,0]"

#Download file from the working dir
curl http://172.16.100.202/ocpu/tmp/x05b85461/files/ch01.pdf

#Check sessionInfo
curl http://172.16.100.202/ocpu/tmp/x05b85461/info/print

```

2.2.3 输入，输出：数据和格式

R 对象的输出格式

可以以各种输出格式检索任何 R 对象（包括记录的图形）

Format	Content-type	Encoder (+args)	Data Frame	Matrix	List	Graphics	Other (function, S4)
print	text/plain	base::print	✓	✓	✓	✗	✓
json	application/json	jsonlite::toJSON	✓	✓	✓	✗	✗
ndjson	application/ndjson	jsonlite::stream_out	✓	✗	✗	✗	✗
csv	text/csv	utils:::write.csv	✓	✓	✗	✗	✗
tab	text/plain	utils:::write.table	✓	✓	✗	✗	✗
md	text/markdown	pander::pander	✓	✓	✗	✗	✗
rda	application/octet-stream	base::save	✓	✓	✓	✓	✓
rds	application/octet-stream	base::saveRDS	✓	✓	✓	✓	✓
pb	application/x-protobuf	protolite::serialize_pb	✓	✓	✓	✗	✗
feather	application/feather	feather::write_feather	✓	✗	✗	✗	✗
png	image/png	grDevices::png	✗	✗	✗	✓	✗
pdf	application/pdf	grDevices::pdf	✗	✗	✗	✓	✗
svg	image/svg+xml	grDevices::svg	✗	✗	✗	✓	✗

```

#read R objects from packages.
curl http://172.16.100.202/ocpu/library/datasets/R/mtcars/json?digits=0
curl http://172.16.100.202/ocpu/library/datasets/R/mtcars/csv
curl http://172.16.100.202/ocpu/library/datasets/R/mtcars/tab?sep="|"
curl http://172.16.100.202/ocpu/library/MASS/R/loglm/print

```

```
#create a simple plot
curl http://172.16.100.202/ocpu/library/graphics/R/plot -d "x=cars"

#replace session id with returned one
curl http://172.16.100.202/ocpu/tmp/x0468b7ab/graphics/last/png
curl http://172.16.100.202/ocpu/tmp/x0468b7ab/graphics/1/png?width=1000
curl http://172.16.100.202/ocpu/tmp/x0468b7ab/graphics/last/svg
curl http://172.16.100.202/ocpu/tmp/x0468b7ab/graphics/last/pdf?width=8
```

R 函数调用的参数格式 (仅限 HTTP POST)

调用函数时，我们需要传递参数。OpenCPU 接受以下类型的参数

Type	What	Examples
Primitives	Numbers, strings, booleans	3.1415, false, "mytitle"
JSON object	An array or object	{"foo": [1,2,3], "bar": {"name": "jerry"}}
R code	plain R code	paste("useR", "2013"), mtcars
File upload	File argument will be temporary file path.	(multipart only)
Temp key (session)	Session ID from previous function call.	x2ac58d69

执行函数功能时, Post 的数据(请求体)可以使下面面的任意形式: multipart/form-data, application/x-www-form-urlencoded, application/json 或者 application/x-protobuf. 并非每个 content-type 都支持任何参数格式:

Content-type	Primitives	Data structures	R code	File upload	Temp key (session)
application/x-www-form-urlencoded	✓	✓ (inline json)	✓	✗	✓
multipart/form-data	✓	✓ (inline json)	✓	✓	✓
application/json	✓	✓	✗	✗	✗
application/x-protobuf	✓	✓	✗	✗	✗

```
#call some functions
curl http://172.16.100.202/ocpu/library/stats/R/rnorm -d "n=10&mean=5"
curl http://172.16.100.202/ocpu/library/graphics/R/hist -d "x=[2,3,2,3,4,3,3]&breaks=10"
curl http://172.16.100.202/ocpu/library/graphics/R/plot -d "x=cars&main='test'"
curl http://172.16.100.202/ocpu/library/base/R/identity -d "x=coef(lm(speed~dist, data=cars))"

#upload local file mydata.csv
curl http://172.16.100.202/ocpu/library/utils/R/read.csv -F "file=@mydata.csv"

#replace session id with returned one above
curl http://172.16.100.202/ocpu/tmp/x067b4172/R/.val/print
curl http://172.16.100.202/ocpu/library/base/R/summary -d "object=x067b4172"

#post arguments in json
curl http://cloud.opencpu.org/ocpu/library/stats/R/rnorm \
-H "Content-Type: application/json" -d '{"n":10, "mean": 10, "sd":10}'
```

运行脚本和可重现的文档

我们可以通过对文件执行 HTTP POST 来运行脚本。该脚本根据其（不区分大小写）文件扩展名进行解释。任何 HTTP POST 参数都会传递给解释函数。支持以下类型：

File extension	Type	Interpreter	Arguments
<code>file.r</code>	R script	<code>evaluate::evaluate</code>	-
<code>file.tex</code>	latex	<code>tools::texi2pdf</code>	-
<code>file.rnw</code>	knitr/sweave	<code>knitr::knit + tools::texi2pdf</code>	-
<code>file.md</code>	markdown	<code>knitr::pandoc</code>	<code>format</code> (see ?pandoc)
<code>file.rmd</code>	knitr/markdown	<code>knitr::knit + knitr::pandoc</code>	<code>format</code> (see ?pandoc)
<code>file.brew</code>	brew	<code>brew::brew</code>	<code>output</code> (see ?brew)

```
#run scripts which exist in packages
curl http://172.16.100.202/ocpu/library/MASS/scripts/ch01.R -X POST
curl http://172.16.100.202/ocpu/library/brew/featurefull.brew -X POST
curl http://172.16.100.202/ocpu/library/brew/brew-test-2.brew -d "output=output.html"
curl http://172.16.100.202/ocpu/library/knitr/examples/knitr-minimal.Rmd -X POST
curl http://172.16.100.202/ocpu/library/knitr/examples/knitr-minimal.Rmd -d "format=docx"
curl http://172.16.100.202/ocpu/library/knitr/examples/knitr-minimal.Rmd -d "format=html"
curl http://172.16.100.202/ocpu/library/knitr/examples/knitr-minimal.Rnw
```

JSON I/O RPC (又名数据处理单元)

对于客户端只对 JSON 格式的函数调用的输出数据感兴趣的常见特殊情况，可以使用后置修复 HTTP POST 请求 URL /json。在这种情况下，成功调用将返回状态 200 (而不是 201)，并且响应主体直接包含 JSON 中返回的对象；无需额外的 GET 请求。

```
curl http://cloud.opencpu.org/ocpu/library/stats/R/rnorm/json -d n=2
[
-1.2804,
-0.75013
]
```

我们可以将它与 application/jsonR 函数上的完整 JSON RPC 的请求内容类型相结合

```
curl http://cloud.opencpu.org/ocpu/library/stats/R/rnorm/json \
-H "Content-Type: application/json" -d '{"n":3, "mean": 10, "sd":10}'
[
4.9829,
6.3104,
11.411
]
```

上面的请求调用以下 R 函数调用：

```
library(jsonlite)
args <- fromJSON('{"n":3, "mean": 10, "sd":10}')
output <- do.call(stats::rnorm, args)
toJSON(output)
```

在这种情况下，相当于：

```
rnorm(n=3, mean=10, sd=10)
```

2.2.4 其他功能

OpenCPU 应用程序

OpenCPU 应用程序是包含在 R 包中的静态网页（html, css, js）。它们通过 OpenCPU API 连接此包中的 R 函数。按照惯例，这些应用程序放在 /inst/www/R 源包的目录中。有关应用页面的更多信息，请参阅。

```
rnorm(n=3, mean=10, sd=10)
```

Github CI Hook

OpenCPU 云服务器包括对持续集成（CI）的支持。因此，每次将提交推送到主分支时，可以将 Github 存储库配置为在 OpenCPU 服务器上自动安装程序包。要利用此功能，需要：

1. R 源包位于存储库的根目录中。（例）
2. Github 用户帐户有一个公共电子邮件地址

要设置 CI，/ocpu/webhook 请在 Github 存储库中将服务器的 URL 添加为“WebHook”。例如，要使用公共演示服务器，请添加带有以下 URL 的 webhook（您可以保持 Content-type 和 Secret 字段不变）

```
https://cloud.opencpu.org/ocpu/webhook
```

要触发 build，请将提交推送到主分支。build 将显示在您的 github webhook 页面中的 Recent Deliveries 下，如果安装成功，您应该收到一封电子邮件（在您的垃圾邮件文件夹中）。如果是，则该程序包将直接可用于 /ocpu/apps/{username}/{package}/ 服务器上的远程使用。

The screenshot shows the GitHub repository settings page for managing webhooks. The left sidebar has tabs for Code, Issues (28), Pull requests (1), Projects (0), Pulse, Graphs, and Settings (highlighted with a red arrow). The main area is titled 'Webhooks / Manage webhook'. It explains that a POST request will be sent to the URL below with event details. It also mentions specifying data format (JSON, x-www-form-urlencoded, etc.) and provides developer documentation. A red arrow points to the 'Payload URL' input field, which contains the URL 'https://cloud.opencpu.org/ocpu/webhook'. Below it is a 'Content type' dropdown set to 'application/json'. There is a 'Secret' input field and a note about SSL verification with a 'Disable SSL verification' button.

如果您使用的是公共服务器，则该程序包也可以通过 `https://{{yourname}}.ocpu.io/{{package}}/`。如果您正在运行自己的 OpenCPU 云服务器，则必须配置 SMTP 服务器才能使电子邮件通知正常工作。

2.3 XGBoost 模型部署实例

本节将基于垃圾邮件的分类任务，训练一个 XGBoost 模型，基于训练的 XGBoost 模型构建一个 R 包，该 R 包的功能用于预测一封邮件是垃圾邮件的概率，并构建一个前端的页面，测试前端页面在 opencpu 中是可用的。其步骤如下：

2.3.1 训练模型

```
library(xgboost)
library(ElemStatLearn)
# 用于构建 R 包使用
data <- save(spam, file='mail_data.rda')

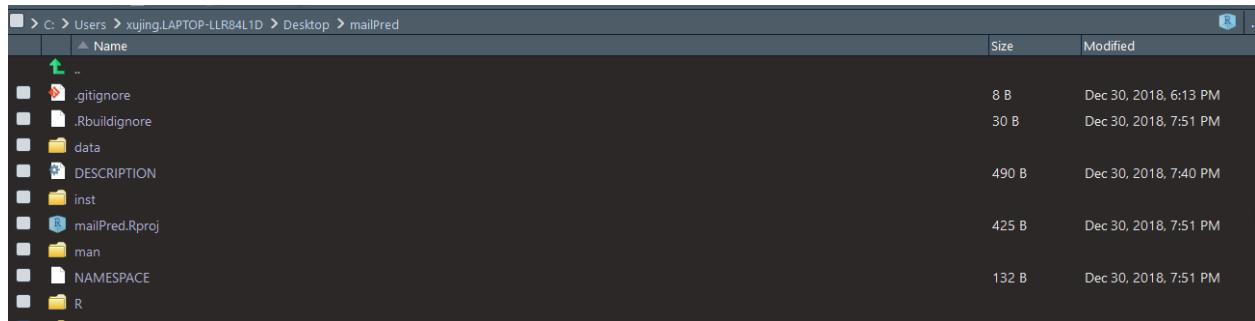
x = as.matrix(spam[, -ncol(spam)])
y = as.numeric(spam$spam) - 1
xgbmodel = xgboost(data = x, label = y, nrounds = 5, objective = 'binary:logistic')
# 这里模型保存的路径可以自己设置
# 保存的训练模型用于构建 R 包
save(xgbmodel, file="xgb.rda")

xgb_model <- load("xgb.rda")
xgbmodel

data_spam <- as.matrix(spam[1, -ncol(spam)])
pred <- predict(xgbmodel, data_spam)
# pred <- xgboost:::predict.xgb.Booster(object = xgbmodel, newdata = data_spam)
```

2.3.2 构建 R 包

关于 R 包的构建，本教程不做详细的介绍，如果个人感兴趣可以参考 R 官网中的教程或 Hadley Wickham 的《R packages》



Name	Size	Modified
..		
.gitignore	8 B	Dec 30, 2018, 6:13 PM
.Rbuildignore	30 B	Dec 30, 2018, 7:51 PM
data		
DESCRIPTION	490 B	Dec 30, 2018, 7:40 PM
inst		
mailPred.Rproj	425 B	Dec 30, 2018, 7:51 PM
man		
NAMESPACE	132 B	Dec 30, 2018, 7:51 PM
R		

```
#' @title Mail predict test for opencpu
#' @description Mail predict test for opencpu.
#' @name mailPred
#' @aliases mailPred
#' @author Xu Jing
#' @usage mailPreds(id)
#' @param id The row of mial data
#'
#' @import xgboost
#' @import jsonlite
```

```
'#' @import stats
'#' @import utils
'#'
'#' @export mailPreds
mailPreds <- function(id = '1') {

  message("XGBoost model test on opencpu!")

  spam <- NULL
  xgbmodel <- NULL

  spam_path <- system.file("extdata", "mail_data.rda", package = "mailPred")
  xgbmodel_path <- system.file("extdata", "xgb.rda", package = "mailPred")

  data_spam <- load(spam_path)
  model_xgb <- load(xgbmodel_path)

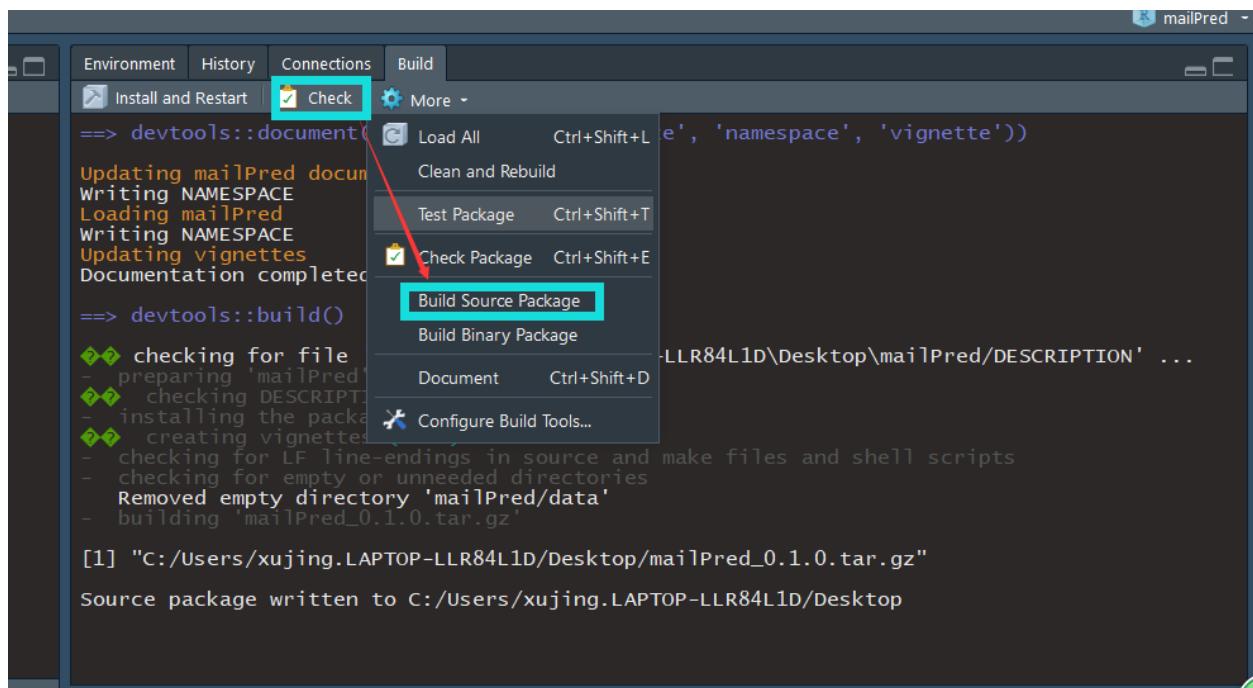
  x = as.matrix(spam[as.integer(id), -ncol(spam)])
  # y = as.numeric(spam[as.integer(id), ncol(spam)]) - 1
  pred <- predict(xgbmodel,x)
  # print(spam)

  return(list(class = pred,id=id))
}

# DESCRIPTION

Package: mailPred
Type: Package
Title: XGBoost Predict Mail
Version: 0.1.0
Author: XuJing
Maintainer: XuJing <274762204@qq.com>
Description: Mail predict test for opencpu.
License: GPL (>= 2)
Encoding: UTF-8
LazyData: false
Date: 2018-12-30
URL: https://github.com/DataXujing/mailPred
BugReports: https://github.com/DataXujing/mailPred/issues
RoxygenNote: 6.1.1
Depends: R (>= 3.4.0)
Imports: xgboost, jsonlite, stats, utils
Suggests: rmarkdown, knitr
VignetteBuilder: knitr
```

2.3.3 编译 R 包并安装

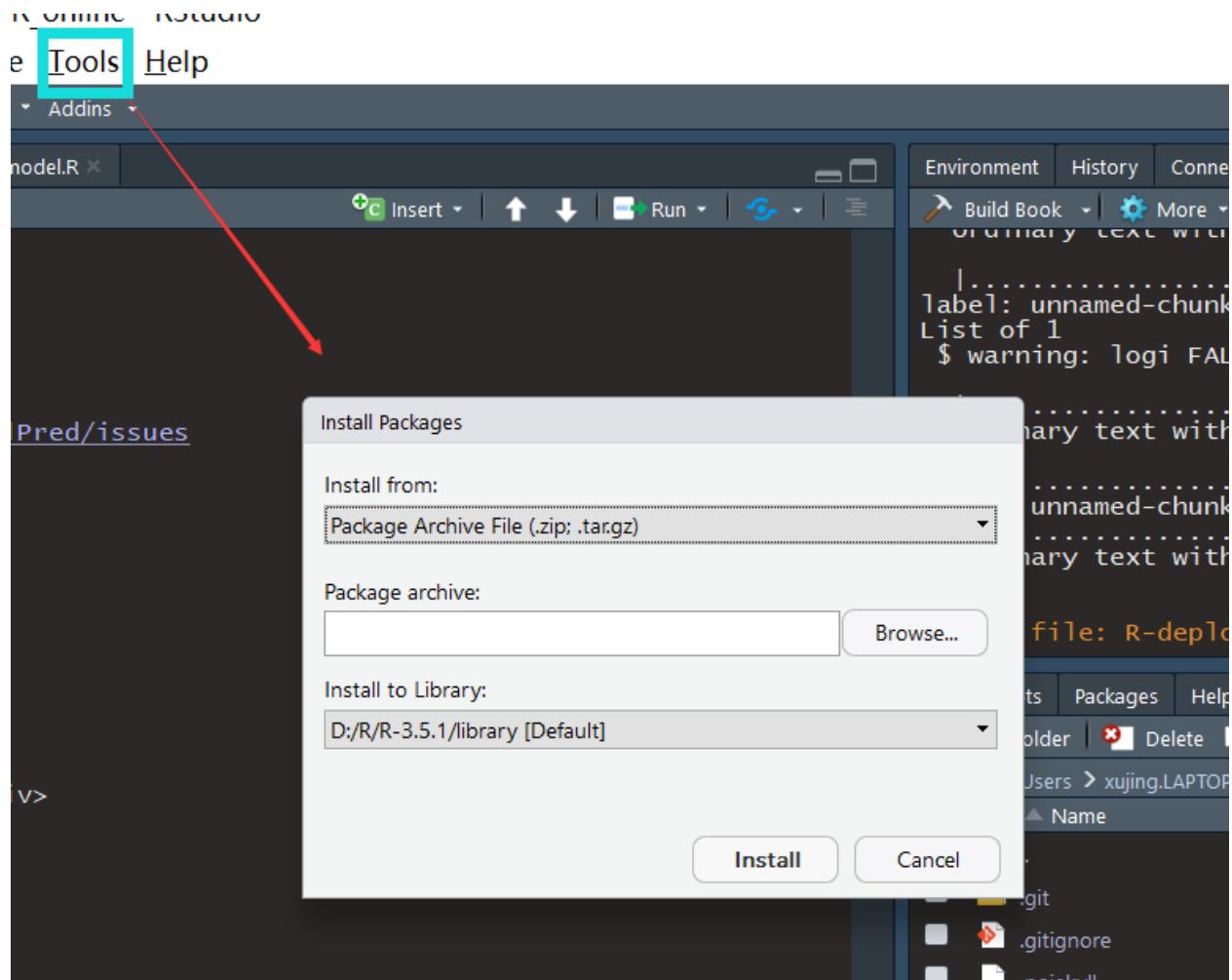


The screenshot shows the RStudio interface with the 'mailPred' project selected. The 'Build' menu is open, and the 'Build Source Package' option is highlighted with a red arrow pointing to it. The terminal window below shows the process of building the package, including documentation and vignette creation, followed by the build command and resulting tar.gz file.

```
--> devtools:::document()
Updating mailPred document
Writing NAMESPACE
Loading mailPred
Writing NAMESPACE
Updating vignettes
Documentation completed
--> devtools:::build()
  checking for file
  - preparing 'mailPred'
  - checking DESCRIPTION
  - installing the package
  - creating vignettes
  - checking for LF line-endings in source and make files and shell scripts
  - checking for empty or unneeded directories
    Removed empty directory 'mailPred/data'
  - building 'mailPred_0.1.0.tar.gz'

[1] "C:/Users/xujing.LAPTOP-LLR84L1D/Desktop/mailPred_0.1.0.tar.gz"
Source package written to C:/Users/xujing.LAPTOP-LLR84L1D/Desktop
```

- Rstudio 中离线安装



- Ubuntu 中离线安装

```
R CMD INSTALL mailPred_0.1.0.tar.gz
```

2.3.4 opencpu的一些配置

修改 opencpu 服务参数，文件位于 /etc/opencpu/server.conf，增加预加载的包

```
"preload": ["jsonlite", "xgboost", "glmnet", "mailPred", "ocputest"]
```

2.3.5 本地 opencpu 服务测试

```
library(opencpu)
ocpu_start_server(root='xj')

http://localhost:5656/xj/test/
http://localhost:5656/xj/library/mailPred/info
http://localhost:5656/xj/library/mailPred/www/mailpred.html
```

The screenshot shows the OpenCPU API Explorer interface. In the top navigation bar, the URL is `localhost:5656/xj/test/`. Below the navigation bar, there are several links: Application, 百度一下, 你就知道, Google, 微软 Bing 搜索 - 国, R, Python, SQL, 数据科学, Github, 其他, Tensorflow, Mxnet, and Other Bookmarks. The main content area has a title "HTTP request options". It shows a POST method selected and the endpoint `../library/mailPred/R/mailPreds/json`. A note says "Method and Endpoint are required. Click below to add additional parameters." Below this, there is a parameter entry field with "id" set to "4". There are buttons for "+ Add parameter" and "+ Add file". A green button labeled "Ajax request" is present. To the right, a status message "HTTP 201 Created" is displayed with the JSON response:

```
{ "class": [0.8351], "id": [4] }
```

. At the bottom left, a welcome message says: "Welcome! Use this simple page to poke around at the API. Specify HTTP method, URL and parameters, and click on Ajax Request. Note that this page requires a browser with HTML5 support." On the right side, there is a sidebar with system information: x-ocpu-time: 2018-12-30 22:10:08 CST, cache-control: max-age=300, public, access-control-allow-origin: http://localhost:5656, and x-ocpu-locale: Chinese (Simplified) China.936.

The screenshot shows a web page titled "垃圾邮件分类opencpu测试" (Spam Email Classification opencpu Test) by "XuJing" on December 30, 2018. The page content includes an "XGBoost分类" section. At the top, there is a navigation bar with links: Home, Blog, API, Download, Apps, Cloud, JS, and Papers. The OpenCPU logo is visible in the top right corner. The main content area features a large image of industrial port cranes against a blue sky with white clouds.

2.3.6 Ubuntu 16.04 opencpu-server 部署测试

The screenshot shows the OpenCPU API Explorer interface. On the left, there's a sidebar with various icons. The main area has tabs for 'OpenCPU API Explorer', 'API docs', 'Example Apps', 'JavaScript Client', 'PDF manual', and 'Source Code'. Below these tabs, a sub-section titled 'HTTP request options' is visible. It shows a 'Method' dropdown set to 'POST' and an 'Endpoint' input field containing '..//library/mailPred/R/mailPreds/json'. Below this, a note says 'Method and Endpoint are required. Click below to add additional parameters.' A table for parameters shows one row with 'Param Name' 'id' and 'Param Value' '2'. Buttons for '+ Add parameter' and '+ Add file' are present. A large green button at the bottom right says 'Ajax request'. To the right of this form, a green box displays the response: 'HTTP 201 Created' followed by a JSON object: { "class": [0.8875], "id": [2] }.

The screenshot shows a web browser window titled '垃圾邮件分类opencpu测试'. The page content includes the author 'XuJing' and the date '2018-12-30'. Below this, the heading 'XGBoost分类' is displayed. The main content area shows a large image of three industrial cranes against a blue sky. At the top of the image, there's a navigation bar with links for 'OpenCPU' (with a logo), 'Blog', 'API', 'Download', 'Apps', 'Cloud', 'JS', and 'P'. On the right side of the image, there are several small status icons, including battery level (92%), signal strength, and memory usage (11.4K/11.2Ki).

2.3.7 Github 部署测试

详细的参考 Github CI Hook 部分及<https://github.com/DataXujing/mailPred>

2.3.8 官方例子

官方提供了更多的例子，详细的可以参考<https://www.opencpu.org/apps.html>

Chapter 3

plumber



HTTP API 已成为软件通信的主要语言。通过创建 HTTP API，您将使您的 R 代码能够被其他服务利用 - 无论它们是在您的组织内部还是托管在世界的另一端。以下是在您将 R 代码包装在 Plumber API 中时向您打开的一些想法：

- 在您的组织中使用其他语言编写的软件可以运行您的 R 代码。您公司的 Java 应用程序现在可以引入您按需生成的自定义 `ggplot2` 图表，或者 Python 客户端可以查询 R 中定义的预测模型。
- 您可以代表您使用某些第三方接收电子邮件，然后在新邮件到达时通知您的 Plumber 服务。您可以在 Slack 上注册一个“Slash Command”，使您能够执行 R 功能以响应 Slack 中输入的命令。
- 您可以编写从访问者的 Web 浏览器查询 Plumber API 的 JavaScript 代码。更进一步，您可以将 Plumber 专

门用作交互式 Web 应用程序的后端。

Plumber 托管在 CRAN 上，因此您可以通过运行以下命令下载并安装最新的稳定版本及其所有依赖项：

```
install.packages("plumber")
```

或者，如果您想运行最新的不稳定开发版本 plumber，可以使用该软件包从其 GitHub 存储库安装它 devtools。

```
install.packages("devtools")
devtools::install_github("rstudio/plumber")
```

3.1 快速开始

Plumber 允许您通过仅使用特殊注释装饰现有 R 代码来创建 API(## 或 #')。下面的示例显示了一个名为 plumber.R (Plumber API 的常规名称) 的文件，该文件定义了 API。

```
# plumber.R

#' Echo the parameter that was sent in
#' @param msg The message to echo back.
#' @get /echo
function(msg=""){
  list(msg = paste0("The message is: '", msg, "'"))
}

#' Plot out data from the iris dataset
#' @param spec If provided, filter the data to only this species (e.g. 'setosa')
#' @get /plot
#' @png
function(spec){
  myData <- iris
  title <- "All Species"

  # Filter if the species was specified
  if (!missing(spec)){
    title <- paste0("Only the '", spec, "' Species")
    myData <- subset(iris, Species == spec)
  }

  plot(myData$Sepal.Length, myData$Petal.Length,
       main=title, xlab="Sepal Length", ylab="Petal Length")
}
```

此文件定义了两个 Plumber“端点”。一个托管在路径上/echo，只是回显传入的消息；另一个托管在路径上/plot 并返回一个显示简单 R 图的图像。

你可以使用 plumber::plumb 函数来此 R 档转换成管道工 API：

```
pr <- plumber::plumb("plumber.R")
```

该 pr 对象现在封装了 plumber.R 文件中表示的所有逻辑。下一步是使用以下 run 方法使 API 生效：

```
pr$run()
```

不妨运行试一下!!!

在前面的示例中，您看到了一个呈现为 JSON 的端点和一个生成图像的端点。除非另有说明，否则 Plumber 将尝试呈现您的端点函数返回的任何内容作为 JSON。但是，您可以指定备用“序列化程序”，指示 Plumber 将输出呈现为其他格式，如 HTML (@html)，PNG (@png) 或 JPEG (@jpeg)

```
#' @get /hello
#' @html
function() {
  "<html><h1>hello world</h1></html>"
}
```

访问时，此端点将生成类似以下内容的内容。它还设置了适当的 Content-Type 标题，以便访问此页面的浏览器知道将结果呈现为 HTML。

```
<html><h1>hello world</h1></html>
```

我们将在 API 的输出中详细讲解。

3.2 路由和输入

传入的 HTTP 请求必须“路由”到一个或多个 R 函数。Plumber 有两个不同的函数系列：端点和过滤器。



通常，当请求到达 Plumber 路由器时，Plumber 首先将该请求通过其过滤器。一旦所有过滤器处理了请求，路由器将查找可满足传入请求的端点。如果找到一个，它将调用端点并使用端点返回的值响应传入的请求。如果没有端点与请求匹配，则将 404 Not Found 返回错误（其行为可由 `set404Handler` 方法控制）详见自定义路由。

3.2.1 端点

此批注指定此函数负责生成对任何 GET 请求的响应/hello。从函数返回的值将用作对请求的响应（在通过序列化程序运行以将响应转换为 JSON 之后）。在这种情况下，GET 响应/hello 将返回 [“hello world”] 带有的内容 `Content-Type`。

生成端点的注释包括：

- `@get`
- `@post`
- `@put`
- `@delete`
- `@head`

请注意，单个端点可以支持多个动词。下面的函数将被用来处理任何传入的 GET，POST 或 PUT 请求/cars

```
#' @get /cars
#' @post /cars
#' @put /cars
function() {
  ...
}
```

3.2.2 过滤器

过滤器可用于定义用于处理传入请求的“管道”。这允许 API 作者将复杂的逻辑分解为一系列独立的，可理解的步骤。与端点不同，请求可以在生成响应之前通过多个 Plumber 过滤器。

通常，Plumber 路由器会在尝试查找满足请求的端点之前，通过所有已定义的过滤器传递请求。

过滤器可以在处理请求时执行以下三种操作之一：

1. 可能在改变请求之后将控制转发到下一个处理程序。
2. 返回响应本身而不是转发给后续处理程序
3. 抛出一个错误

3.2.2.1 转发给另一个处理程序

过滤器最常见的行为是在改变传入请求或调用某些外部副作用后将请求传递给下一个处理程序。一个常见的用例是使用过滤器作为请求记录器：

```
/* Log some information about the incoming request
#* @filter logger
function(req) {
  cat(as.character(Sys.time()), "-",
    req$REQUEST_METHOD, req$PATH_INFO, "-",
    req$HTTP_USER_AGENT, "@", req$REMOTE_ADDR, "\n")
  plumber::forward()
}
```

此过滤器是直接的：它调用外部操作（日志记录），然后调用 `forward()` 将控制权传递给管道中的下一个处理程序（另一个过滤器或端点）

类似的过滤器可能会改变它给出的请求或响应对象上的某些状态

```
#* @filter setuser
function(req) {
  un <- req$cookies$user
  # Make req$username available to endpoints
  req$username <- un

  plumber::forward()
}
```

3.2.2.2 返回响应

过滤器也可以返回响应。您可能希望检查请求是否满足某些约束（如身份验证），并且 - 在某些情况下 - 返回响应而不调用任何其他处理程序。例如，可以使用过滤器来检查用户是否已经过身份验证。

```
#* @filter checkAuth
function(req, res) {
  if (is.null(req$username)) {
    res$status <- 401 # Unauthorized
    return(list(error = "Authentication required"))
  } else {
    plumber::forward()
  }
}
```

Plumber API 中出现错误的常见原因是忘记 `forward()` 在过滤器中调用。在这样的过滤器中，最后一行的结果将作为对传入请求的响应以静默方式返回。这可能会导致您的 API 表现出非常奇怪的行为，具体取决于返回的内容。当您使用过滤器时，请务必仔细审核所有代码路径，以确保您正在调用 `forward()`，导致错误或故意返回值。

3.2.2.3 抛出错误

最后，过滤器可能会抛出错误。如果在定义过滤器的代码中出错或者过滤器故意调用 `stop()` 以触发错误，则会发生这种情况。在这种情况下，任何后续处理程序都不会处理请求，并且会立即将其发送到路由器的错误处理程序。有关如何自定义此错误处理程序的详细信息，请参阅路由器自定义。

3.2.3 动态路由

除了具有类似硬编码的路由之外 /hello，Plumber 端点还可以具有动态路由。动态路由允许端点定义一组更灵活的路径，以便它们匹配。

```
users <- data.frame(
  uid=c(12,13),
  username=c("kim", "john")
)

#' Lookup a user
#' @get /users/<id>
function(id) {
  subset(users, uid==id)
}
```

此 API 使用动态路径/users/来匹配任何形式的请求，/users/后跟一些路径元素，如数字或字母。在这种情况下，如果找到具有相关 ID 的用户，则返回有关用户的信息，否则返回空对象。

此 API 使用动态路径/users/来匹配任何形式的请求，/users/后跟一些路径元素，如数字或字母。在这种情况下，如果找到具有相关 ID 的用户，则返回有关用户的信息，否则返回空对象。

您可以根据需要命名这些动态路径元素，但请注意，动态路径中使用的名称必须与函数的参数名称匹配（在本例中为两者 id）。

您甚至可以执行更复杂的动态路由，例如：

```
#' @get /user/<from>/connect/<to>
function(from, to) {
  # Do something with the 'from' and 'to' variables...
}
```

除非另有说明，否则从查询字符串或动态路径传递到管道工端点的所有参数都将是字符串。例如，请考虑以下 API。

```
#' @get /type/<id>
function(id) {
  list(
    id = id,
    type = typeof(id)
  )
}
```

访问 `http://localhost:8000/types/14` 将返回：

```
{
  "id": ["14"],
  "type": ["character"]
}
```

如果您只打算支持动态路由中特定参数的特定数据类型，则可以在路径本身中指定所需的类型。

```
#* @get /user/<id:int>
function(id) {
  next <- id + 1
  # ...
}

#* @post /user/activated/<active:bool>
function(active) {
  if (!active) {
    # ...
  }
}
```

下面详细介绍了可以在动态类型中使用的类型名称的映射以及它们如何映射到 R 数据类型。

R Type	Plumber Name
logical	bool , logical
numeric	double , numeric
integer	int

3.2.3.1 静态文件处理

plumber 包括一个静态文件服务器，可用于托管静态资源（如 JavaScript，CSS 或 HTML 文件）的目录。这些服务器配置相当简单，并且可以集成到 plumber 应用程序中。

```
#* @assets ./files/static  
list()
```

此示例将在服务器./files/static 的默认/public 路径中公开本地目录。因此，如果您有一个文件./files/static/branding.html，它将在您的 plumber 服务器上可用/public/branding.html。

您可以选择提供其他参数来配置用于服务器的公共路径。例如

```
#* @assets ./files/static /static  
list()
```

请求的目录不在/public，但在/static

在上面的示例中，服务器的“实现”只是一个空的 list()。还可以指定 function() 与其他 plumber 注释类似的操作。此时，实现不会改变静态服务器的行为。最终，此列表或函数可以通过更改缓存控制设置等内容来提供配置服务器的机会。

3.2.3.2 输入处理

根据传入 HTTP 请求的路径和方法路由请求，但请求可以包含比此更多的信息。它们可能包含其他 HTTP 标头，查询字符串或请求正文。所有这些字段都可以被视为您的 Plumber API 的“输入”。

这个可以详细的参考 plumber 的官方文档，在此不再赘述。

3.3 Rendering Output

3.3.1 序列化器

默认情况下，Plumber 通过 jsonliteR 包将对象序列化为 JSON。但是，包中内置了各种其他序列化程序。

注解	内容类型	说明/参考
@json	application/json	jsonlite::toJSON()
@html	text/html; charset=utf-8	无需任何其他序列化即可通过响应
@jpeg	image/jpeg	jpeg()
@png	image/png	png()
@serializer htmlwidget	text/html; charset=utf-8	htmlwidgets::saveWidget()
@serializer unboxedJSON	application/json	jsonlite::toJSON(unboxed=TRUE)

3.3.2 绕过序列化

在某些情况下，可能需要直接从 R 返回值而不进行序列化。您可以通过从端点返回响应对象来绕过序列化。例如，请考虑以下 API

```
#' Endpoint that bypasses serialization
#' @get /
function(res) {
  res$body <- "Literal text here!"

  res
}
```

从此端点返回的响应将包含 Literal text here! 没有 Content-Type 标头且没有任何其他序列化的主体。

同样，您可以利用 @serializer contentType 不对响应进行序列化但指定 contentType 标头的注释。如果希望更好地控制发送的响应，可以使用此批注。

```
#* @serializer contentType list(type="application/pdf")
#* @get /pdf
function() {
  tmp <- tempfile()
  pdf(tmp)
  plot(1:10, type="b")
  text(4, 8, "PDF from plumber!")
  text(6, 2, paste("The time is", Sys.time()))
  dev.off()
```

```
readBin(tmp, "raw", n=file.info(tmp)$size)
}
```

运行此 API 并访问 `http://localhost:8000/pdf` 将下载从 R 生成的 PDF（如果您的客户支持，则本机显示 PDF）

3.3.3 boxed and unboxed JSON

您可能已经注意到，从 Plumber 生成的 API 响应将奇异值（或“标量”）呈现为数组。例如：

```
jsonlite:: toJSON(list(a=5))
```

```
## {"a": [5]}
```

a 元素的值虽然是单数，但仍然呈现为数组。这可能会让您感到惊讶，但这样做是为了保持输出的一致性。虽然 JSON 将标量与矢量对象区分开来，但 R 却没有。这在将 R 对象序列化为 JSON 时会产生歧义，因为不清楚特定元素是应该呈现为原子值还是 JSON 数组。

考虑以下 API，它以字典方式返回比给定字母“更高”的所有字母。

```
#' Get letters after a given letter
#' @get /boxed
function(letter="A") {
  LETTERS[LETTERS > letter]
}

#' Get letters after a given letter
#' @serializer unboxedJSON
#' @get /unboxed
function(letter="A") {
  LETTERS[LETTERS > letter]
}
```

访问 `http://localhost:8000/boxed?letter=U` 或 `http://localhost:8000/unboxed?letter=U` 将返回相同的响应

```
["V", "W", "X", "Y", "Z"]
```

但是，`http://localhost:8000/boxed?letter=Y` 将产生：

```
["Z"]
```

而 `http://localhost:8000/unboxed?letter=Y` 将产生：

```
"Z"
```

Plumber 继承了 `jsonlite::toJSON` 设置的默认值，`auto_unbox=FALSE` 这将导致所有长度为 1 的向量仍然呈现为 JSON 数组。如果要更改特定端点的此行为，可以将端点配置为使用序列化程序（如上所示）

3.3.4 自定义图像序列化程序

`@jpeg` 和 `@png` 分别返回到客户端 `jpeg()` 或 `png()` 功能。这些功能接受各种各样的定制包括输出额外的选项 `width`, `height` 以及 `bg` 等等

```
#' Example of customizing graphical output
#' @png (width = 400, height = 500)
#' @get /
function() {
  plot(1:10)
}
```

3.3.5 错误处理

Plumber 包装每个端点调用，以便它可以正常捕获错误。

```
#' Example of throwing an error
#' @get /simple
function() {
  stop("I'm an error!") # 状态代码为 500
}

#' Generate a friendly error
#' @get /friendly
function(res) {
  msg <- "Your request did not include a required parameter."
  res$status <- 400 # Bad request
  list(error=jsonlite::unbox(msg))
}
```

3.3.6 设置 Cookies(*)

作为完成请求的一部分，Plumber API 可以选择在客户端上设置 HTTP cookie。HTTP API 不会隐式包含“会话”的概念。如果没有一些附加信息，Plumber 无法确定进来的两个 HTTP 请求是否与同一用户相关联。Cookie 提供了一种方法来委托客户端代表您存储某些状态，以便所选数据可以比单个 HTTP 请求更长时间；这里讨论使用 cookie 跟踪 API 中的状态的全部含义。

3.3.6.1 设置未加密的 Cookie

下面的 API 端点将返回一个随机字母，但它会记住您是否喜欢大写或小写字母的首选项。

```
#' @put /preferences
function(res, capital) {
  if (missing(capital)) {
    stop("You must specify a value for the 'capital' preference.")
  }
  res$setCookie("capitalize", capital)
}

#' @get /letter
function(req) {
  capitalize <- req$cookies$capitalize

  # Default to lower-case unless user preference is capitalized
  alphabet <- letters

  # The capitalize cookie will initially be empty (NULL)
  if (!is.null(capitalize) && capitalize == "1") {
    alphabet <- LETTERS
  }

  list(
    letter = sample(alphabet, 1)
  )
}
```

由于此 API 使用 PUT 请求来测试此 API，因此我们将 curl 在命令行上使用它来测试它。（没有任何关于需要 PUT 请求的 cookie；您可以轻松地修改此 API 以使用 GET 请求。）我们可以从访问/letter 端点开始，我们将看到 API 默认为小写字母。curl http://localhost:8000/letter

```
{
  "letter": ["y"]
}
```

如果我们发送 PUT 请求并指定 capital 参数，则将在客户端上设置 cookie，以允许服务器在将来的请求中容纳我们的首选项。在 curl，您需要使用该 -c 选项指定要在其中保存这些 cookie 的文件。这是一个很好的提醒，客户端处理 cookie 的方式不同 - 有些人根本不会支持它们 - 所以如果你想使用它们，请确保你打算用 API 支持的客户端很好地使用 cookie。

要发送 PUT 请求，将参数设置 capital 为 1，我们可以调用：curl -c cookies.txt -X PUT -data ‘capital=1’ “http://localhost:8000/preferences”。如果您打印出该 cookies.txt 文件，您现在应该看到它包含一个名为 capitalize 值的 cookie 1。

我们可以提出另一个 GET 请求，/letter 看它是否符合我们的偏好。但是我们需要告诉 curl 我们使用 -b 交换机发送此请求时使用我们刚刚创建的 cookie 文件：curl -b cookies.txt http://localhost:8000/letter██。

3.3.6.2 设置加密的 cookie

除了存储纯文本 cookie 之外，Plumber 还支持处理加密的 cookie，加密的 cookie 会阻止用户查看其中存储的内容，并对其内容进行签名，以便用户无法修改存储的内容。

要使用此功能，必须在构建路由器后将其明确添加到路由器中。例如，您可以运行以下命令序列来创建支持加密会话 cookie 的路由器。

```
pr <- plumbr("myfile.R")
pr$registerHooks(sessionCookie("mySecretHere", "cookieName"))
pr$run()
```

你会注意到上面的例子正在使用 sessionCookiePlumber 附带的钩子。通过在路由器上添加注册这些挂钩，您将确保该 req\$session 对象在传入请求中可用，并且 cookieName 在响应准备好发送给用户时持久保存到命名的 cookie。在这个例子中，用于加密数据的密钥“mySecretHere”显然是一个非常弱的密钥。

与之不同的是 res\$setHeader()，附加的值 req\$session 是通过序列化的 jsonlite；这样您就可以自由地在会话中使用更复杂的数据结构，例如列表。与此不同的是 res\$setHeaders()，req\$session 使用您提供的密钥作为 sessionCookie() 函数的第一个参数来加密数据。

例如，我们将存储一个加密的 cookie，用于计算此客户端访问特定端点的次数：

```
#* @get /sessionCounter
function(req) {
  count <- 0
  if (!is.null(req$session$counter)) {
    count <- as.numeric(req$session$counter)
  }
  req$session$counter <- count + 1
  return(paste0("This is visit #", count))
}
```

同样，sessionCookie() 在此代码工作之前，您需要在路由器上注册挂钩。

如果您检查浏览器中设置的 cookie，您会发现它的值在到达客户端时已加密。但是当它到达 Plumber 时，您的 cookie 可以作为常规 R 列表使用，并且可以被读取或修改。

3.4 部署

不要使用 run() 方法，部署自己的 API，这样是很危险的！对于 plumber 的生产环境的部署有如下方式：DigitalOcean，RStudio Connect，Docker，pm2。在这里我们着重介绍 plumber API 部署在 Docker 和 pm2，我们将举一个具体的时机例子并测试我们生产环境的部署效果。

3.4.1 Docker（基础）

在此不会深入研究 Docker 的细节或如何在系统上设置或安装所有内容。Docker 为那些希望入门的人提供了一些很好的资源。在这里，我们假设您已安装 Docker，并且您熟悉启动容器所需的基本命令。

安装镜像

```
docker pull trestletech/plumber

# 查看现在系统中存在的镜像
docker images

# 后边会常用的 docker 命令
docker pull **
docker ps # 查看正在运行的容器列表
docker stop IDs
docker rmi IDs
docker rm XXX
```

运行服务

```
docker run --rm -p 8000:8000 trestletech/plumber \
/usr/local/lib/R/site-library/plumber/examples/04-mean-sum/plumber.R
```

- docker run 告诉 Docker 运行一个新容器
- --rm 告诉 Docker 在容器完成后进行清理
- -p 8000:8000 说将端口 8000 从 plumber 容器（我们将运行服务器的位置）映射到本地计算机的端口 8000
- trestletech/plumber 是我们要运行的镜像的名称
- /usr/local/lib/R/site-library/plumber/examples/03-mean-sum/plumber.R 是 Docker 容器内部到您要托管的 Plumber 文件的路径。

```
docker run --rm -p 8000:8000 -v `pwd`/api.R:/plumber.R trestletech/plumber /plumber.R
```

- -v 把当前工作目录下的 api.R 挂载到镜像中的/plumber.R 的目录下
- 定义了该文件的位置应为/plumber.R，所以我们最后给出的参数告诉容器在哪里查找 plumber 定义

3.4.2 Docker（高级）

关于多 plumber 部署，同一端口的多应用部署，负载均衡，可以参考<https://www.rplumber.io/docs/hosting.html#custom-dockerfiles>

3.4.3 pm2

不熟悉在 Docker 中托管 API，那么您需要找到一种运行方式，直接管理服务器上的 Plumber API。

pm2 是一个最初以 Node.js 为目标的流程管理器。这里我们将展示在 Ubuntu 14.04 中执行此操作所需的命令，但您可以使用 **pm2** 支持的任何操作系统或分发。最后，您将拥有一台服务器，可在启动时自动启动 **plumber** 服务，如果它们崩溃则重新启动它们，甚至可以集中管理管道服务的日志。

3.4.3.1 安装 pm2

现在您已准备好安装 **pm2**。**pm2** 是 npm (Node.js 的包管理系统) 中维护的包；它还需要 Node.js 才能运行。所以要开始你要安装 Node.js。在 Ubuntu 14.04 上，必要的命令是：

```
sudo apt-get update
sudo apt-get install nodejs npm
# 由于 Ubuntu 下已经有一个名叫 node 的库，因此 Node.js 在 ubuntu 下默认叫 nodejs，需要额外处理一下
sudo ln -s /usr/bin/nodejs /usr/bin/node
```

一旦安装了 npm 和 Node.js，就可以安装 **pm2** 了。

```
sudo npm install -g pm2
```

这将 **-g** 在您的服务器上安装 **pm2 global ()**，这意味着您现在应该能够运行 **pm2 -version** 并获取已安装的 **pm2** 的版本号。

为了让 **pm2** 在启动时启动你的服务，你应该运行 **sudo pm2 startup** 它将为你的系统创建必要的文件，以便在你启动你的机器时运行 **pm2**。

3.4.3.2 Wrap Your Plumber File

将 **Plumber** 文件部署到服务器后，您仍需要告诉服务器如何运行服务器。你可能习惯于运行像这样的命令

```
pr <- plumb("myfile.R")
pr$run(port=4500)
```

不幸的是，**pm2** 本身并不理解 R 脚本；您可以使用该 **pm2 list** 命令查看 **pm2** 已在运行的服务。如果您现在运行此命令，您将看到 **pm2** 没有任何它负责的服务。将脚本和代码存储在所需目录中后，使用以下命令告知 **pm2** 您的服务。

```
pm2 start --interpreter="Rscript" /usr/local/plumber/myfile/run-myfile.R
```

您应该看到有关 **pm2** 的一些输出，它们启动了您的服务实例，然后是 **pm2** 的一些状态信息。如果一切正常，您将看到您的新服务已注册并正在运行。您可以通过 **pm2 list** 再次执行来查看相同的输出。

一旦您对已定义的 **pm2** 服务感到满意，您可以使用 **pm2 save** 告诉 **pm2** 保留下次启动计算机时运行的服务集。您定义的所有服务都将自动重新启动。

- 如果要查看更多信息，使用 **pm2 show run-myfile**
- 如果您需要检查服务的日志文件，则可以运行 **pm2 logs run-myfile**
- 如果您想要查看服务器和所有 **pm2** 服务的运行状况的大图，您可以运行 **pm2 monit** 它将显示所有服务的 RAM 和 CPU 使用情况的仪表板。

3.4.4 XGBoost 模型预测部署实例

我们将给予 XGBoost 模型，通过一个具体的实例演示从模型的训练到部署的整个过程，这里我们分别采用 docker 和 **pm2** 部署我们的模型

- 1. 模型的训练与预测

```
# 垃圾邮件分类，一共 57 个特征
library(xgboost)
library(glmnet)
```

```

library(ElemStatLearn)
x = as.matrix(spam[, -ncol(spam)])
y = as.numeric(spam$spam) - 1
xgbmodel = xgboost(data = x, label = y, nrounds = 5, objective = 'binary:logistic')
# 这里模型保存的路径可以自己设置
save(xgbmodel, file="xgb.rda")
# glmmmodel = cv.glmnet(x = x, y = y, family = 'binomial')
# save(glmmmodel, file="glm.rda")

# 模型加载
xgb_model <- load("xgb_model/xgb.rda")
xgbmodel

# 模型预测
data_spam <- as.matrix(spam[1, -ncol(spam)])
pred <- predict(xgbmodel,data_spam)
pred <- xgboost:::predict.xgb.Booster(object = xgbmodel, newdata = data_spam)

```

- 2. 构建 plumber API 接口

```

#* Logging
#* @filter logger
function(req) {
  model <- load("/home/R_code/xgb_model/xgb.rda")
  cat(as.character(Sys.time()), "-",
      req$REQUEST_METHOD, req$PATH_INFO, "-",
      req$HTTP_USER_AGENT, "@", req$REMOTE_ADDR, "\n")
  plumber::forward()
}

# xgb_api

#* XGBoost predict model
#* @param id The row id of Spam data.
#* @serializer unboxedJSON
#* @get /predict
function(id) {
  if(as.integer(id) <= 4601){
    data_spam <- as.matrix(spam[as.integer(id), -ncol(spam)])
    pred <- predict(xgbmodel,data_spam)
    # pred <- xgboost:::predict.xgb.Booster(object = xgbmodel, newdata = data_spam)
    list(spam_id = id, predict = pred)
  }
  else{
    stop('Your id error')
  }
}

#* static source
#* @assets /home/R_code/xgb_model/static /home
list()

```

- 3. 在 R 中启用服务

```
library(xgboost)
library(ElemStatLearn)
library(plumber)

setwd('/home/R_code/xgb_model')
pr <- plumber('xgb_api.R')
pr$run(host='0.0.0.0', port=8088)
```

- 4.Docker 中部署

```
docker run --rm -p 8000:8000 -v `pwd`/xgb_model_docker:/xgb_model_docker trestletech/plum
```

但是此时我们使用的像 xgboost 这样的包并没有加载到镜像中，解决这个问题的方法是我们自己自定义 dockerfiles。关于这方面的内容你可以参考 docker 的相关教程。

- 5.pm2 部署

```
pm2 start --interpreter="Rscript" /home/R_code/xgb_model/test_api.R
```

- 6. 效果展示

```
root@Data:~# pm2 start --interpreter="Rscript" /home/R_code/xgb_model/test_api.R
[PM2] Spawning PM2 daemon with pm2_home=/root/.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting /home/R_code/xgb_model/test_api.R in fork_mode (1 instance)
[PM2] Done.
```

App name	id	version	mode	pid	status	restart	uptime	cpu	mem	user	watching
test_api	0	N/A	fork	25682	online	0	0s	0%	20.6 MB	root	disabled

Use `pm2 show <id|name>` to get more details about an app

```
root@Data:~# pm2 list
```

App name	id	version	mode	pid	status	restart	uptime	cpu	mem	user	watching
test_api	0	N/A	fork	25682	online	0	36s	0%	114.8 MB	root	disabled

Use `pm2 show <id|name>` to get more details about an app

```
root@Data:~#
```





3.5 其他

3.5.1 环境

当你 plumb() 是一个文件时，Plumber 会调用 source() 该文件来评估你定义的任何顶级代码。

```
# Global code; gets executed at plumb() time.
counter <- 0

#' @get /
function() {
  # Only gets evaluated when this endpoint is requested.
  counter <- counter + 1
}
```

counter 则将创建变量，该变量将存在于为此 API 创建的环境中，每访问一次 counter 就加 1，此技术允许所有端点

和过滤器共享在 API 顶层定义的某些数据。

默认情况下，每个路由和子路由都会创建自己的环境，如果要共享环境，则在创建路由器时需要提供单个共享环境。关于环境的介绍可以参考我的 R 语言面向对象的编程。

3.5.2 文件系统

写入文件过程中的并发性是不可靠的，举个简单的例子，如果您已经将 API 水平扩展到五个 R 进程，那么两个转到 `write.csv()` 同时，你会看到一个进程的数据被另一个进程立即覆盖，或者 - 更糟糕的是 - 你可能最终得到一个无法读取的损坏的 CSV 文件。不要依赖文件系统来协调并发运行的单个 R 进程的共享状态。还有就是要注意，有些平台可能不具备磁盘持久化。

协调 API 状态时要考虑的最后一个选项是利用外部数据存储。这可以是关系数据库（如 MySQL），非关系数据库（如 MongoDB），也可以是 Redis 等事务数据存储。

任何这些选项的一个重要考虑因素是确保它们是“事务性的”，这意味着两个试图同时写入的 Plumber 进程不会相互覆盖。如果你对此感兴趣，可在<https://db.rstudio.com/>看一些的的介绍

如果在终止 Plumber 流程时有一个需要清理的数据库连接池。您可以使用 `exit` 钩子来定义这样的处理程序

```
pr <- plumb("plumber.R")
pr$registerHook("exit", function() {
  print("Bye bye!")
})
pr$run()
```

当您中断 API 时（例如，通过 Escape 按键或 Ctrl+C），您将看到 Bye bye! 打印到控制台。您甚至可以注册多个 `exit` 钩子，它们将按照注册顺序运行。

3.5.3 API 的安全性

3.5.3.1 网络和防火墙

防火墙是一种阻止不受欢迎的网络流量的方法。大多数台式计算机和许多服务器都带有开箱即用的防火墙。这意味着如果要公开在端口 8000 上运行的 API，则需要将防火墙配置为接受该端口上的传入连接。防火墙也可以在其他网络中介上配置，因此您可能需要配置多个防火墙以允许流量通过，以便向 API 客户端公开所需的端口。

3.5.3.2 拒绝服务（DoS）

使用拒绝服务（DoS）攻击是为了通过流量压缩服务器或服务来暂时关闭服务器或服务。DoS 场景可能是由单个无知用户无意中发出可能要求服务器执行某项不可能完成任务的请求引起的，或者可能是由恶意行为者故意引入，利用大量机器重复发出昂贵的请求。服务器响应。

在设计 Plumber API 时，应该采用一些做法，以便围绕 API 请求可能发起的工作提供安全保护。

```
#' This is an example of an UNSAFE endpoint which
#' is vulnerable to a DOS attack.
#' @get /
#' @png
function(pts=10) {
  # An example of an UNSAFE endpoint.
  plot(1:pts)
}

#' This is an example of an safe endpoint which
#' checks user input to avoid a DOS attack
#' @get /
```

```
#' @png
function(pts=10) {
  if (pts > 1000) {
    stop("pts must be < 1,000")
  }

  plot(1:pts)
}
```

在这里，可以看到我们只允许用户请求最多 1,000 个点的图表。超过该限制的任何请求将立即终止，无需进一步计算。

还有其他的，比如限制文件读取，跨站请求伪造等，可以详细参考一些前端的知识，如果你只是在企业内部内网环境中构建接口，可暂时不考虑这些内容。

3.5.4 创建和控制路由器

要以编程方式实例化新的 Plumber 路由器，您可以调用 `plumber$new()`。这将返回一个没有端点的空白 Plumber 路由器。您可以调用 `run()` 返回的对象来启动 API，但它不知道如何响应任何请求，因此任何传入流量都会得到 404 响应。

3.5.5 定义端点

可以使用该 `handle()` 方法在路由器上定义端点。例如，要定义响应 GET 请求/和 POST 请求的 Plumber API/`submit`，您可以使用以下代码：

```
pr <- plumber$new()
pr$handle("GET", "/", function(req, res) {
  # ...
})

pr$handle("POST", "/submit", function(req, res) {
  # ...
})
```

如果使用注释来定义 API，则在这些 `handle` 调用中定义的“处理程序”函数与您在 `plumber.R` 文件中定义的代码相同。

该 `handle()` 方法采用其他参数，允许您控制端点的细微差别行为，例如它可能抢占哪个过滤器或应该使用哪个序列化程序。例如，以下端点将使用 Plumber 的 HTML 序列化程序。

```
pr <- plumber$new()
pr$handle("GET", "/", function() {
  "<html><h1>Programmatic Plumber!</h1></html>"
}, serializer=plumber::serializer_html())
```

3.5.6 定义过滤器

使用 `filter()` Plumber 路由器的方法定义新的过滤器：

```
pr <- plumber$new()

pr$filter("myFilter", function(req) {
  req$filtered <- TRUE
```

```

forward()
}

pr$handle("GET", "/", function(req) {
  paste("Am I filtered?", req$filter)
})

```

3.5.7 在路由器上注册钩子 (*)

plumber 路由器支持“钩子”的概念，可以注册这些钩子以在请求的生命周期中的特定点执行某些代码。Plumber 路由器目前支持四个钩子：

- `preroute(data, req, res)`
- `postroute(data, req, res, value)`
- `preserialize(data, req, res, value)`
- `postserialize(data, req, res, value)`

在上述所有情况中，您可以访问 `data` 参数中的一次性环境，该参数作为每个请求的临时数据存储创建。钩子可以在这些钩子中存储临时数据，这些钩子可以被处理同一请求的其他钩子重用。

在 Plumber 路由器中定义钩子时的一个特性是能够修改返回的值。这种钩子的约定是：任何接受命名参数的函数 `value` 都应该返回新值。这可能是传入的值的未修改版本，也可能是变异值。但在任何一种情况下，如果你的钩子接受一个参数 `value`，那么你的钩子返回的任何东西都将被用作响应的新值。

您可以使用该 `registerHook` 方法添加钩子，也可以使用 `registerHooks` 带有名称列表的方法一次添加多个钩子，其中名称是钩子的名称，值是处理程序本身。

```

pr <- plumber$new()
pr$registerHook("preroute", function(req) {
  cat("Routing a request for", req$path_info, "...\\n")
})
pr$registerHooks(list(
  preserialize=function(req, value) {
    print("About to serialize this value:")
    print(value)

    # Must return the value since we took one in. Here we're not choosing
    # to mutate it, but we could.
    value
  },
  postserialize=function(res) {
    print("We serialized the value as:")
    print(res$body)
  }
))

pr$handle("GET", "/", function() { 123 })

```

发出 GET 请求/将从我们注册的三个事件中打印出各种信息。

3.5.8 Mounting & Static File Routers (*)

plumber 路由器可以通过使用 `mount()` 方法通过路径划分 API，这是将大型 API 分解为较小文件的绝佳技术。

```
root <- plumber$new()

users <- plumber$new("users.R")
root$mount("/users", users)

products <- plumber$new("products.R")
root$mount("/products", products)
```

这与用于定义提供静态文件目录的路由器的方法相同。静态文件路由器只是使用创建的 Plumber 路由器的一个特例 PlumberStatic\$new()。例如

```
pr <- plumber$new()

stat <- PlumberStatic$new("./myfiles")

pr$mount("/assets", stat)
```

这将使存储在./myfiles 目录中的文件和目录在 API 的/assets/路径下可用。

3.5.9 自定义路由器

还可以使用以下任何一种方法修改路由器的行为：

- setSerializer() - 设置路由器的默认序列化程序。
- setErrorHandler() - 设置在任何过滤器或端点生成错误时调用的错误处理程序。
- set404Handler() - 设置在任何过滤器，端点或子路由器无法提供传入请求时调用的处理程序。

3.5.10 调试

3.5.10.1 打印调试

大多数程序员首先通过在代码中添加 print 语句来进行调试，以便在某些时候检查状态。在 R 中，print() 或者 cat() 可以用来打印出一些状态。例如，cat("i is currently:", i) 可以在代码中插入，以帮助您确保变量 i 是代码中该位置的变量。

这种方法在 Plumber 同样可行。在交互式环境中开发 Plumber API 时，此调试输出将记录到您调用 run()API 的同一终端。在非交互式生产环境中，这些消息将包含在 API 服务器日志中以供以后检查。

3.5.10.2 交互式调试

在本地开发 API 时通过 browser() 在其中一个过滤器或端点中添加调用，然后在客户端中访问 API 来利用。当您想要检查多个不同的变量或与函数内部的当前状态进行交互时，这提供了一种强大的技术。

```
#' @get /
function(req, res) {
  browser()

  list(a=123)
}
```

The screenshot shows an RStudio interface with several tabs open at the top: '02_test.R', '02_test_server.R', 'Source on Save', and '03_test.R'. The '03_test.R' tab is active, displaying the following R code:

```
1 #' @get /
2 function(req, res){
3   browser()
4   list(a=123)
5 }
6 }
```

Below the code editor is a terminal window titled 'Console' with the path 'C:/Users/xujing/LAPTOP-LLR84L1D/Desktop/闭关修炼/闭关/R_online/my_codes/plumber/'. The terminal output shows the following:

```
Warning message:
In readlines(file) : incomplete final line found on '02_test.R'
> pr$run()
Starting server to listen on port 8246
Running the swagger UI at http://127.0.0.1:8246/_swagger_/
> pr <- plumber::plumb("03_test.R")
Warning message:
In readlines(file) : incomplete final line found on '03_test.R'
> pr$run()
Starting server to listen on port 8246
Running the swagger UI at http://127.0.0.1:8246/_swagger_/
Called from: (function (req, res)
{
  browser()
  list(a = 123)
})(res = <environment>, req = <environment>)
Browse[1]> |
```


Chapter 4

jug

4.1 What is jug?

jug 是一个微型的轻量级的框架，基于 httpuv 包，为的是部署你的 R 代码更简单。

jug 不会是一个高效的框架，它的作用是让你轻松的为你的 R 代码创建 API，jug 的简单灵活，理论上你可以用其构建一个更一般的 Web 应用。

4.2 Install and Hello World

要安装最新版本，请使用 devtools：

```
devtools::install_github("Bart6114/jug")  
  
# jug.parallel 允许 jug 并行处理请求  
devtools :: install_github ("Bart6114/jug.parallel")
```

或者安装 CRAN 版本：

```
install.packages("jug")
```

加载库：

```
library(jug)  
library(jug.parallel)  
  
# Example1  
jug()  
  
# Example2  
library(jug)  
  
jug() %>%  
  get("/", function(req, res, err){  
    "Hello World!"  
  }) %>%  
  simple_error_handler_json() %>%  
  serve_it()
```

```
# Example 3
library(jug)

jug() %>%
  get("/", function(req, res, err) {
    "Hello World!"
  }) %>%
  simple_error_handler_json() %>%
  serve_it_parallel(processes=8)

kill_servers()
```

jug 与 magrittr (%>%) 的管道功能密切配合。

4.3 Middleware (中间件)

在中间件方面, jug 有遵循中间件的规范 Express。在 jug 中, 中间件是一个可以访问 `request (req)`, `response (res)` 和 `error (err)` 对象的函数。

可以定义多个中间件。中间件的添加顺序很重要。请求将从添加的第一个中间件 (更具体地说是在其中指定的函数 - 请参见下一段) 开始。它将继续通过添加的中间件传递, 直到中间件不返回 NULL。

4.3.1 方法不敏感的中间件

该 `use` 函数是一个方法不敏感的中间件说明符。虽然它对方法不敏感, 但它可以绑定到特定路径。如果 `path` 参数 (接受带 `grep` 设置的正则表达式字符串 `perl=TRUE`) 如果设置为 NULL, 它也会变得路径不敏感, 并将处理每个请求。

路径不敏感的栗子:

```
jug() %>%
  use(path = NULL, function(req, res, err) {
    "test 1,2,3!"
  }) %>%
  serve_it()
```

```
$ curl 127.0.0.1:8080/xyz
test 1,2,3!
```

同样的栗子, 但是路径敏感:

```
jug() %>%
  use(path = "/", function(req, res, err) {
    "test 1,2,3!"
  }) %>%
  serve_it()
```

```
$ curl 127.0.0.1:8080/xyz
curl: (52) Empty reply from server

$ curl 127.0.0.1:8080
test 1,2,3!
```

请注意, 在上面的示例中, 缺少错误/缺少路由处理 (服务器可能崩溃/不响应), 稍后将详细介绍。

4.3.2 方法敏感的中间件

与请求方法不敏感的中间件相同的样式，有可用的请求方法敏感中间件。更具体地讲，您可以使用 `get`, `post`, `put` 和 `delete` 功能。

此类中间件使用 `path` 参数绑定到路径。如果 `path` 设置为 `NULL`，它将绑定到路径的每个请求，对应相应的请求方法。

```
jug() %>%
  get(path = "/", function(req, res, err) {
    "get test 1,2,3!"
  }) %>%
  serve_it()

$ curl 127.0.0.1:8080
get test 1,2,3!
```

中间件意味着被链接，因此要将不同的功能绑定到不同的路径：

```
jug() %>%
  get(path = "/", function(req, res, err) {
    "get test 1,2,3 on path /"
  }) %>%
  get(path = "/my_path", function(req, res, err) {
    "get test 1,2,3 on path /my_path"
  }) %>%
  serve_it()

$ curl 127.0.0.1:8080
get test 1,2,3 on path /

$ curl 127.0.0.1:8080/my_path
get test 1,2,3 on path /my_path
```

4.3.3 Websocket 协议

默认情况下，所有中间件便利功能都绑定到 `http` 协议。但是，您可以使用 `websocket` 中间件功能通过 `websocket` 访问 `jug` 服务器 `ws`。下面是回传传入消息的示例。

```
jug() %>%
  ws("/echo_message", function(binary, message, res, err) {
    message
  }) %>%
  serve_it()
```

打开连接并向 `ws://127.0.0.1:8080/echo_message` 其发送例如消息 `test` 将返回该值 `test`。

请注意，`websocket` 支持在此阶段是实验性的，尽量不使用 `jug` 操作 `websocket`

4.3.4 `include` 定义其他位置的中间件

为了使代码更加模块化，您可以将其他定义的中间件链包含到您的 `jug` 实例中。为此，您可以使用 `collector()` 和 `include()` 功能的组合。

下面是一个 `collector` 本地定义（在相同的 R 脚本中）和 `include` 票子：

```

collected_mw<-
  collector() %>%
  get("/", function(req, res, err) {
    return("test")
  })

res<-jug() %>%
  include(collected_mw) %>%
  serve_it()

```

然而，也有可能 include 一个 collector 是在另一个.R 文件中定义。

让我们说下面是文件 my_middlewares.R:

```

library(jug)

collected_mw<-
  collector() %>%
  get("/", function(req, res, err) {
    return("test2")
  })

```

我们可以 include 如下：

```

res<-jug() %>%
  include(collected_mw, "my_middlewares.R") %>%
  serve_it()

```

4.4 预定义的中间件

4.4.1 错误处理

一个简单的错误处理中间件 (simple_error_handler/ simple_error_handler_json)，它捕获未绑定的路径和 func 评估错误。如果您没有实现自定义错误处理程序，我建议您将其中任何一个添加到您的 jug 实例中。simple_error_handler 返回一个 HTML 错误页面而 simple_error_handler_json 返回一个 JSON 消息。

```

jug() %>%
  simple_error_handler() %>%
  serve_it()

```

```

$ curl 127.0.0.1:8080
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Not found</title>
  </head>
  <body>
    <p>No handler bound to path</p>
  </body>
</html>

```

如果要实现自己的自定义错误处理，只需查看这些简单错误处理中间件的代码即可。

请注意，通常您希望在指定所有其他中间件后将错误处理程序中间件附加到 jug 实例。

4.4.2 轻松使用自己的函数

创建 jug 的主要原因是可以轻松访问您自己的自定义 R 函数。功能 `decorate` 专门为此目的而构建。如果 `decorate` 您自己的函数，它会将请求的查询字符串中传递的所有参数转换为函数的参数。它还将所有头文件作为参数传递给函数。如果您的函数不接受...参数，则会删除函数未明确请求的所有查询/标头参数。如果您的功能请求 `req`, `res` 或 `err` 参数（或...）相应的对象将被传递。

```
say_hello<-function(name) {paste("hello", name, "!")}

jug() %>%
  get("/", decorate(say_hello)) %>%
  serve_it()
```

如果在上面，您通过 `name` 查询字符串或 GET 请求中的标头传递参数，它将返回如下例所示。

```
$ curl 127.0.0.1:8080/?name=Bart
hello Bart !
```

4.4.3 静态文件服务器

`serve_static_file` 中间件可以提供静态文件。

```
jug() %>%
  serve_static_files() %>%
  serve_it()
```

默认根目录是返回的目录，`setwd()` 可以通过向中间件提供 `root_path` 参数来指定 `serve_static_files`.

除了开发之外，我不建议使用 jug 来提供静态文件。

4.4.4 CORS 功能 (*)

CORS 功能(跨源资源共享)由 `cors()` 中间件功能引入。

请考虑以下示例。

```
jug() %>%
  cors() %>%
  get("/", function(req, res, err){
    "Hello World!"
}) %>%
  serve_it()

$ curl -v 127.0.0.1:8080/
* Trying 127.0.0.1...
* Connected to 127.0.0.1 (127.0.0.1) port 8080 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1:8080
> User-Agent: curl/7.43.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Type: text/html
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Methods: POST,GET,PUT,OPTIONS,DELETE,PATCH
< Content-Length: 12
```

```
<
* Connection #0 to host 127.0.0.1 left intact
```

如您所见，这会添加一些默认的 CORS 标头。查看?cors 配置选项，请注意您还可以通过指定 path 参数将 CORS 标头添加到特定路径。

4.4.5 认证

目前，只有通过中间件功能内置支持基本身份验证https://www.httpwatch.com/httpgallery/authentication/auth_basic，中间件将检查有效用户名/密码组合的请求。如果传递了无效组合，它将返回 401 状态，WWW-Authenticate 标题和文本正文，指出存在身份验证错误。

首先，您需要定义一个接受 username 和 password 参数的函数。TRUE 如果组合有效且 FALSE 组合无效，则应返回功能。一个虚拟的例子如下所示。注意，此功能还可以检查例如数据库以验证组合。

```
# dummy account checker
account_checker <- function(username, password) {
  # do something to verify the username and password and return TRUE if combination OK
  all(username == "test_user",
      password == "test_password")
}
```

接下来，您需要 auth_basic 在中间件链中实例化中间件。该 auth_basic 函数接受用户名/密码验证功能作为第一个参数。下面给出两个例子。第一个显示如何对特定路径 (/test) 进行身份验证。

```
jug() %>%
  get("/", function(req, res, err) {
    "/ req"
  }) %>%
  get("/test", auth_basic(account_checker), function(req, res, err) {
    "/test req"
  }) %>%
  serve_it()
```

下面的第二个示例显示了如何为 jug 实例中的所有路径激活基本身份验证。

```
jug() %>%
  use(NULL, auth_basic(account_checker)) %>%
  get("/", function(req, res, err) {
    "/ req"
  }) %>%
  serve_it()
```

4.5 事件监听

从版本 0.1.7.902 开始，事件监听的概念已经可用。由于中间件不足以实现强大的 Logger，因此引入了事件和事件监听的概念。目前，侦听器可以绑定到事件，下面给出一个示例：

```
jug() %>%
  get("/", function(req, res, err) {"foo"}) %>%
  on("finish", function(req, res, err) {
    print("the finish event was received; request processing finished!")
  }) %>%
  serve_it()
```

目前有三项活动：

- **start**: 一旦收到新请求，就会触发此事件
- **finish**: 一旦请求完全处理，就会触发此事件
- **error**: 一旦在中间件内引发错误，就会触发此事件

`start` 和 `finish` 事件将传递的当前状态 `req`, `res` 以及 `err` 对象。`error` 事件将传递第四个参数，即错误消息的字符串表示。

4.6 预定义的事件侦听器

4.6.1 Logger

```
futile.logger
jug() %>%
  get("/", function(req,res,err){"foo"}) %>%
  get("/err", function(req,res,err){stop("bar")}) %>%
  logger(threshold = futile.logger::DEBUG, log_file='logfile.log', console=TRUE) %>%
  simple_error_handler_json() %>%
  serve_it()
```

在上面的示例中，`Logger` 阈值设置 `futile.logger::DEBUG` 为我们在执行期间接收详细信息，在这个例子中，`Logger` 将写入 `logfile.log` 和将输出到控制台。有关 `Logger` 阈值的更多信息，请查看该 `futile.logger` 包的文档。

4.7 请求，响应和错误对象

4.7.1 Request (`req`) 对象

该 `req` 对象包含请求规范。它有不同的属性：

- `req$params` 由查询字符串，JSON 正文，URL 参数或多部分表单传递的参数的命名列表
- `req$path` 请求路径
- `req$method` 请求方法
- `req$raw` 传递的原始请求对象 `httpuv`
- `req$body` 完整的请求正文作为字符串
- `req$protocol` 无论是 `http` 或 `websocket`
- `req$headers` 请求中的标头的命名列表（作为小写并从 `HTTP_` 底层 `httpuv` 框架提供的前缀中剥离）

它附带以下功能：

- `req$get_header(key)` 返回与请求中指定键关联的值（无需担心 `HTTP_` 前缀）
- `req$set_header(key, value)` 允许在处理请求时设置/更改标头（对于将数据传递到下一个中间件可能很有用）
- `req$attach(key, value)` 将变量附加到 `req$params`

4.7.2 Response (`res`) 对象

该 `res` 对象包含响应规范。它有不同的属性：

- `res$headers` 一个命名的标题列表
- `res$status` 响应的状态（默认为 `200`）
- `res$body` 响应的主体（自动设置为不 `NULL` 返回的中间件的内容或通过诸如此类的方法 `res$json()`）

它还有一组功能：

- `res$set_header(key, value)` 设置自定义标头
- `res$content_type(type)` 设置自己的内容类型（MIME）
- `res$status(status)` 设置响应的状态
- `res$text(body)` 明确地设定反应的主体
- `res$json(obj, auto_unbox=TRUE)` 将对象转换为 JSON，将其设置为正文并设置正确的内容类型
- `res$plot(plot_obj, base64=TRUE)` 方便函数将绘图对象作为响应体返回，返回的绘图可以是图像的 `base64` 表示（默认）或实际的二进制数据

4.7.3 Error (`err`) 对象

该 `err` 对象包含可通的错误列表 `err$errors`。您可以通过调用将错误添加到此列表中 `err$set(error)`。错误将转换为字符。有关更多详细信息，请参阅“错误处理”。

4.8 URL 调度

在路径参数 `get`, `post`, ... 功能被处理为正则表达式模式。

如果路径定义中有命名的捕获组，则它们将附加到该 `req$params` 对象。例如，模式 `/test/(?<id>.*)/(?<id2>.*)` 将导致变量 `id` 和 `id2`（及其各自的值）绑定到 `req$params` 对象。

如果路径模式未以字符串 ^ 正则表达式标记的开头启动或以字符串标记的结尾结束 \$，则将分别在路径模式规范的开头和结尾处明确地插入这些模式。例如，路径模式 / 将转换为 ^/\$

4.9 启动 jug 实例

只需 `serve_it()` 在管道链的末端调用（参见 Install and Hello World！示例）

4.10 线性回归模型的 API 举栗

训练 `mtcars` 数据集上的线性回归模型，并假设我们的目标是 `mpg` 根据输入 `gear` 和预测每加仑英里或变量 `hp`。

```
head(mtcars)

##          mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4     21.0   6 160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag 21.0   6 160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710    22.8   4 108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive 21.4   6 258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360 175 3.15 3.440 17.02  0  0    3    2
## Valiant       18.1   6 225 105 2.76 3.460 20.22  1  0    3    1

mpg_model <- lm(mpg ~ gear + hp, data = mtcars)

summary(mpg_model)

##
## Call:
## lm(formula = mpg ~ gear + hp, data = mtcars)
##
## Residuals:
##      Min        1Q    Median        3Q       Max
## -4.7977 -2.4288 -0.7685  2.2405  7.5943
```

```

## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 17.755144   3.241809   5.477 6.74e-06 ***
## gear         3.176520   0.762584   4.165 0.000255 ***
## hp          -0.063931   0.008206  -7.791 1.36e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## Residual standard error: 3.108 on 29 degrees of freedom
## Multiple R-squared:  0.7513, Adjusted R-squared:  0.7341 
## F-statistic: 43.79 on 2 and 29 DF,  p-value: 1.731e-09

```

建立一个最小预测函数。

```

predict_mpg <- function(gear, hp) {
  predict(mpg_model,
    newdata = data.frame(gear=as.numeric(gear),
                          hp=as.numeric(hp)))[[1]]
}

```

我们可以通过提供 gear 和 hp 参数来测试函数。

```
predict_mpg(gear = 4, hp = 80)
```

```
## [1] 25.34671
```

现在，要将此函数公开为 Web API，我们需要构建一个 jug 实例。我们可以使用内置的 decorate 中间件来简化 predict_mpg 功能的集成。下面是一个最小的例子。

```

jug() %>%
  post("/predict-mpg", decorate(predict_mpg)) %>%
  simple_error_handler_json() %>%
  serve_it()

```

我们现在可以向 `http://127.0.0.1:8080/predict-mpg` 发送 http POST 请求，它将返回预测值！它开箱即用，带有 JSON 主体中的参数，multipart/form-data 或者作为一个 x-www-form-urlencoded。

JSON 正文

```

curl -X POST \
  http://127.0.0.1:8080/predict-mpg \
  -H 'content-type: application/json' \
  -d '{"hp": 80, "gear": 4}'

```

多部分形式

```

curl -X POST \
  http://127.0.0.1:8080/predict-mpg \
  -H 'content-type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW'
  -F hp=80 \
  -F gear=4

```

urlencode 表单

```

curl -X POST \
  http://127.0.0.1:8080/predict-mpg \
  -H 'content-type: application/x-www-form-urlencoded' \
  -d 'gear=4&hp=80'

```

4.11 官方栗子

<https://github.com/Bart6114/jug-crud-example>

Chapter 5

fiery

Chapter 6

Rserve

Chapter 7

RestRserve

Chapter 8

mailR

mailR 是一个比较小的包，主要解决的问题是 R 与邮件发送的问题，该包就一个方法：send.mail() 方法调用方式为：

```
send.mail(from, to, subject = "", body = "", encoding = "iso-8859-1",
html = FALSE, inline = FALSE, smtp = list(), authenticate = FALSE,
send = TRUE, attach.files = NULL, debug = FALSE, ...)
```

参数列表：

- from 有效的发送者的邮箱
- to 目标接收的邮箱
- subject 邮箱主题
- body 邮件体
- encoding 邮件内容字符编码支持包括 iso-8859-1 (default), utf-8, us-ascii, and koi8-r
- html bool 值，是否把邮箱体解析成 html
- inline 布尔值，HTML 文件中的图像是否应该嵌入内联。
- smtp lsit 类型，链接邮箱的 smtp
- authenticate 一个布尔变量，用于指示是否需要授权连接到 SMTP 服务器。如果设置为 true，请参阅 SMTP 参数所需参数的详细信息。发送一个布尔值，指示电子邮件是否应该在函数的末尾发送。（默认行为）。如果设置为 false，函数将电子邮件对象返回给父环境。
- attach.files 链接到文件的文件系统中路径的字符串向量或有效 URL 到附加到电子邮件（详见更多信息附加 URL）
- debug bool 值，是否查看 debug 的真实细节
- ... Optional arguments to be passed related to file attachments. See details for more

Example1:

```
mailR::send.mail(
  from = 'sender@tuandai.com', # 发送人
  to = 'sendee@tuandai.com', # 接收人
  cc = 'carboncopy@tuandai.com', # 抄送人
  subject = ' 邮件标题',
  body = as.character(
    '<div style = "color:red"> 邮件正文，可以为 HTML 格式</div>',
  ),
  attach.files = NULL, # 附件的路径
```

```

encoding = "utf-8",
smtp = list(
  host.name = 'smtp.exmail.qq.com', # 邮件服务器 IP 地址
  port = 465, # 邮件服务器端口
  user.name = 'senderName', # 发送人名称
  passwd = 'yourpassword', # 密码
  ssl = T),
html = T, inline = T, authenticate = T, send = T, debug = F
)

```

Example2:

```

send.mail(from = "sender@gmail.com",
          to = c("Recipient 1 <recipient1@gmail.com>", "recipient2@gmail.com"),
          cc = c("CC Recipient <cc.recipient@gmail.com>"),
          bcc = c("BCC Recipient <bcc.recipient@gmail.com>"),
          subject="Subject of the email",
          body = "Body of the email",
          smtp = list(host.name = "aspmx.l.google.com", port = 25),
          authenticate = FALSE,
          send = TRUE)

```

Example3:

```

send.mail(from = "sender@gmail.com",
          to = c("recipient1@gmail.com", "recipient2@gmail.com"),
          subject = "Subject of the email",
          body = "Body of the email",
          smtp = list(host.name = "smtp.gmail.com", port = 465, user.name = "gmail_username"),
          authenticate = TRUE,
          send = TRUE)

```

Example4:

```

email <- send.mail(from = "Sender Name <sender@gmail.com>",
                    to = "recipient@gmail.com",
                    subject = "A quote from Gandhi",
                    body = "In Hindi : ॐ एक अपराह्न विद्युत अवधारणा एक अपराह्न विद्युत  

                    English translation: An ounce of practice is worth more than tons of p
                    encoding = "utf-8",
                    smtp = list(host.name = "smtp.gmail.com", port = 465, user.name = "gmail_username"),
                    authenticate = TRUE,
                    send = TRUE)

```

Example5:

```

send.mail(from = "sender@gmail.com",
          to = c("recipient1@gmail.com", "recipient2@gmail.com"),
          subject = "Subject of the email",
          body = "Body of the email",
          smtp = list(host.name = "smtp.gmail.com", port = 465, user.name = "gmail_username"),
          authenticate = TRUE,
          send = TRUE,
          attach.files = c("./download.log", "upload.log"),
          file.names = c("Download log", "Upload log"), # optional parameter
          file.descriptions = c("Description for download log", "Description for upload log"))

```

Example6:

```
send.mail(from = "sender@gmail.com",
          to = c("recipient1@gmail.com", "recipient2@gmail.com"),
          subject = "Subject of the email",
          body = "<html>The apache logo - <img src=\"http://www.apache.org/images/asf_logo.png\"/></html>",
          html = TRUE,
          smtp = list(host.name = "smtp.gmail.com", port = 465, user.name = "gmail_username",
          authenticate = TRUE,
          send = TRUE)
```

Example7:

```
send.mail(from = "sender@gmail.com",
          to = c("recipient1@gmail.com", "recipient2@gmail.com"),
          subject = "Subject of the email",
          body = "path.to.local.html.file",
          html = TRUE,
          inline = TRUE,
          smtp = list(host.name = "smtp.gmail.com", port = 465, user.name = "gmail_username",
          authenticate = TRUE,
          send = TRUE)
```


Chapter 9

Rweixin(*)

关于 RWeixin，这里不做介绍，可以参考：

- <https://github.com/Lchiffon/Rweixin>
- https://dataxujing.github.io/assets/bowen20/slides_langdawei.pdf

Chapter 10

参考文献

10.1 opencpu

- opencpu API 文档
- The opencpu System
- 利用 R 和 opencpu 搭建高可用的 HTTP 服务 — 刘思喆
- R 语言实战之模型部署

10.2 plumber

- plumber 官方教程
- R 工程化 — Rest API 之 plumber 包

10.3 jug

- jug 官方教程