第十一章 用Shiny APP发布分析结果

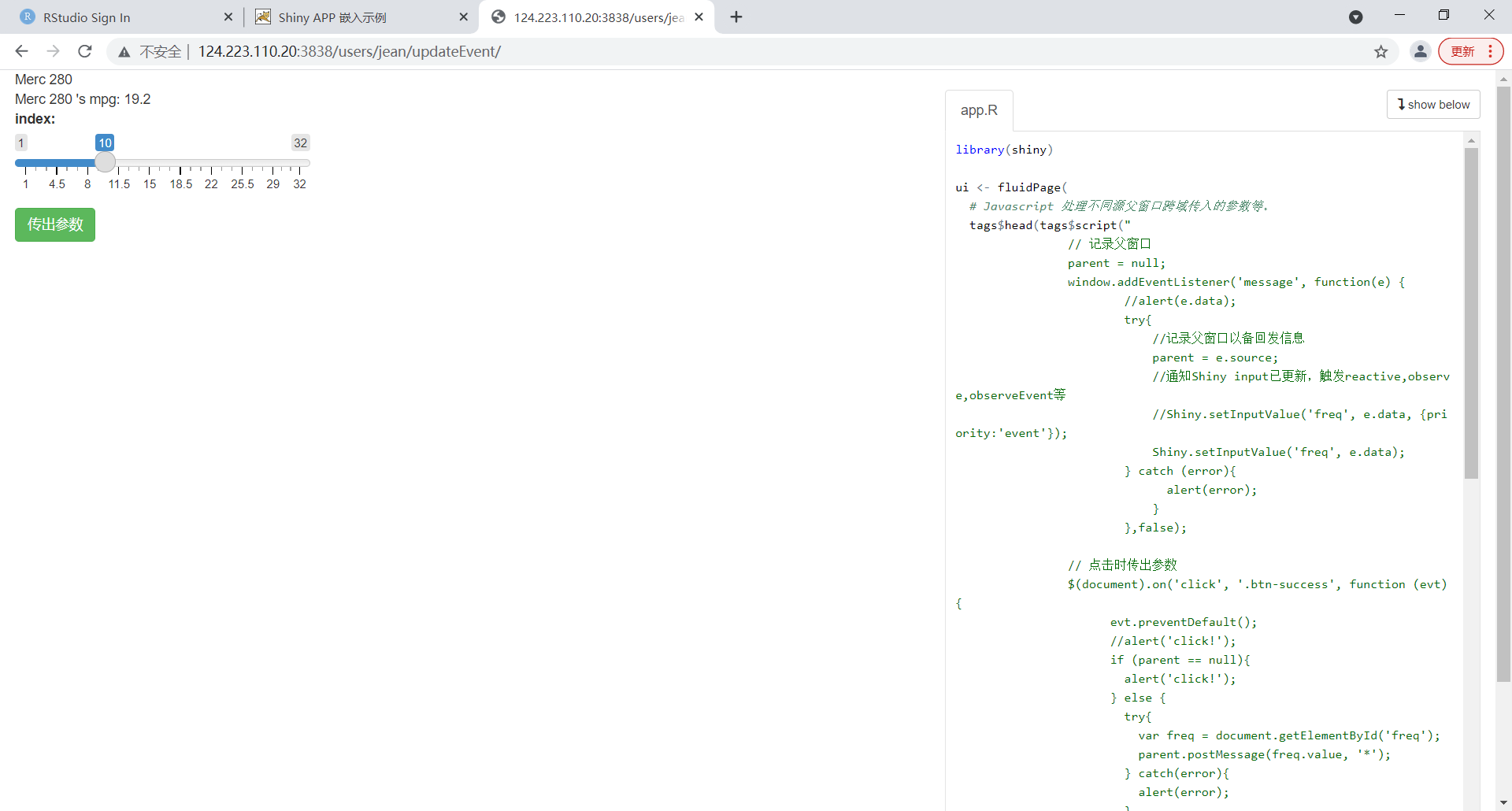
前面各章对各种税收数据作了深度分析的尝试，有R语言的，Python语言的，也有Cypher语言访问图数据库，包括图算法的Java扩展插件等。分析结果出来后，需要以适当的渠道与形式呈现给最终用户。一般来说深度分析是相当高技能的工作，只有少数人能做，但分析结果需要以非常易用易懂的形式呈现给最终用户，深度分析才能落地应用，实现它的价值。目前移动互联网普及的环境下，最方便的大概就是网页的形式了。Rstudio Shiny Server正是这样一个数据分析结果发布平台，通过其自定义的交互式UI组件，把分析结果封装成WEB APP，每个数据分析应用就是一个Shiny APP，交互式的呈现给最终用户。它独有的reactive编程模式则极大的简化了基于浏览器的B/S应用开发，UI及服务器端的逻辑（深度数据分析）都在服务器端由R语言完成，可以看作是对R语言的扩充，极大的简化了发布的过程。也可以通过reticulate包调用Python完成，从而扩展了平台的适用场景。该平台比较适合本书的需求，下面通过几个小例子感性的了解一下。

第一节 开发Shiny APP并嵌入宿主网页

本节先通过几个小的Shiny APP了解 Shiny APP的架构及特点。这些APP可以通过我的研究网页访问，网址：

<https://124.223.110.20:8443/shiny/index.html> 。

一、[查询车型资料示例](https://124.223.110.20:8443/shiny/testembedded.html)

这个简单的Shiniy APP通过一个sliderInput控件给出的下标，显示mtcars数据集中相应车型的资料。拉动控件改变下标则更新下标对应车型的资料。

一个Shiny APP由ui.R、server.R、global.R（可选）三部分R语言程序和一个描述运行模式的DESCRIPTION(可选)文件组成，DESCRIPTION中的运行模式为Showcase时，页面中就可以看到APP的源码，以及浏览器端输入更新时触发了服务器端的哪部分源码。UI部分与Server部分可以合并成单一个文件app.R，本例是单一程序文件运行在Showcase模式。

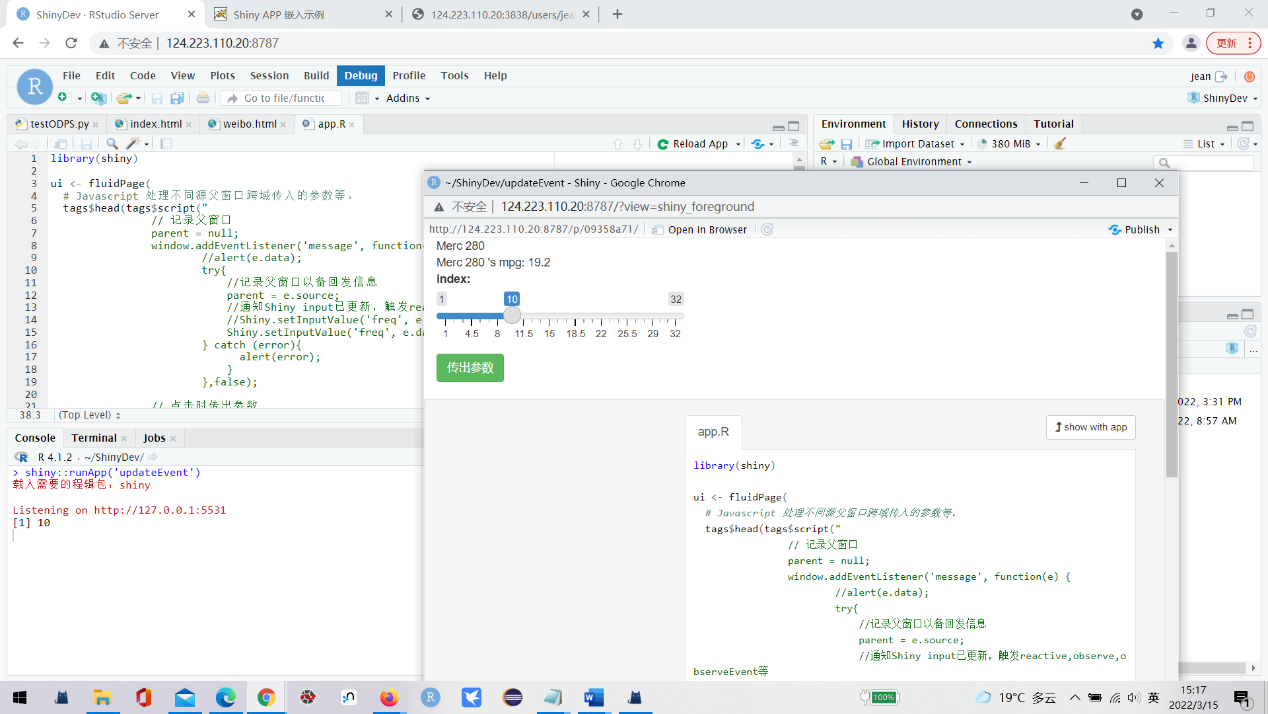
函数ui定义了该APP的UI部分，由服务器端生成并根据交互式UI控件，如本例sliderInput的变化，通过网页的文档对象模型DOM由服务器端动态的生成与更新，它最终会生成一个HTML页面。这里先不去管前面tags$head嵌入的javascript部分及后面的observeEvent()部分，可以先注释掉。函数server定义了服务器端响应输入变化的处理逻辑，包括定义一个响应变量监视浏览器端输入变量的变化，根据响应变量的变化更新浏览器端的文本输出，以及更新sliderInput的可视化状态。所有的输入变量都在input列表下定义，所有的响应输出都在output列表下定义，它们在UI函数与Server函数中以Shiny规定的语法互相引用，在UI中通过一系列的render函数组合引用输出变量最后动态生成UI控件的HTML代码。最后

shinyApp(ui, server)

则把两个函数组装成一个Shiny APP。可以看到浏览器与服务器两端运行的程序都集成到了服务器端的R程序中开发，由Shiny的reactive机制来解决浏览器与服务器两端之间的通讯与数据同步，因而极大的简化了B/S应用的开发，简单快捷的把深度分析的结果发布到WEB界面。

在Shiny Server的主目录下建一个文件夹，把APP相关的文件拷贝进去，比如本例拷贝到~/updateEvent目录下，即可在浏览器中访问。

在Rstudio IDE中，可以直接按程序页上的“Run APP”按钮调试运行程序。



library(shiny)

# UI函数，定义Shiny APP的UI

ui <- fluidPage(

# Javascript 处理不同源父窗口跨域传入传出的参数。

tags$head(tags$script("

// 记录父窗口

parent = null;

window.addEventListener('message', function(e) {

//alert(e.data);

try{

//记录父窗口以备回发信息

parent = e.source;

//通知Shiny input已更新，触发reactive,observe,observeEvent等

//Shiny.setInputValue('freq', e.data, {priority:'event'});

Shiny.setInputValue('freq', e.data);

} catch (error){

alert(error);

}

},false);

// 点击时传出参数

$(document).on('click', '.btn-success', function (evt) {

evt.preventDefault();

//alert('click!');

if (parent == null){

alert('click!');

} else {

try{

var freq = document.getElementById('freq');

parent.postMessage(freq.value, '\*');

} catch(error){

alert(error);

}

}

});

")),

# 输出output$name为文本，与server()函数renderText()配对

textOutput("name"),

# 输出output$MPG为文本，与server()函数renderText()配对

textOutput("MPG"),

# 定义sliderInput交互式变量 input$freq

sliderInput("freq","index:",min = 1, max = length(row.names(mtcars)),

value = 10, round = TRUE),

# 定义传出参数按钮

actionButton("sendout", "传出参数", class = "btn-success")

)

# 服务端函数，实现所有服务器端逻辑

server <- function(input, output, session) {

# 定义响应变量freq，监视输入变量input$freq的变化

freq<- reactive({

paste(input$freq)

})

# 根据响应变量freq的变化，调用render函数更新输出变量output$name，

# Shiny会通过配对的textOutput()函数更新浏览器中对应的文本。

output$name <- renderText({

name<- row.names(mtcars)[input$freq]

})

# 更新输出变量output$MPG对应的文本

output$MPG <- renderText({

name<- row.names(mtcars)[input$freq]

paste(name, "'s mpg:", mtcars[name,]$mpg)

})

# 根据响应变量freq监视浏览器端输入变量的变化事件

# 通知浏览器更新sliderInput的可视化状态。

# 该函数在处理嵌入宿主页面发来的参数时会激活。

observeEvent(freq(),{

print(paste("event triggered:",input$freq))

# 更新浏览器端JS Widget

updateSliderInput(session, "freq", value = freq())

})

}

# 组装成一个Shiny APP

shinyApp(ui, server)

DESCRIPTION文件则指示了Shiny运行该APP的方式，比如这里指示其以Showcase方式运行，可以在浏览器上看到APP的源码，当交互式输入变量变化时，服务器端被激活的源码会以黄色高亮显示，直观的演示reactive机制工作的过程。该模式要指定一个服务器端后台程序侦听的端口号。

Title: 查询车型资料示例

DisplayMode: Showcase

Port: 3839

二、Shiny APP嵌入宿主页面

Shiny提供的Shiny APP与传统WEB界面应用系统集成的方法，就是通过IFRAME把Shiny APP嵌入到传统系统的宿主网页中，如本例所示。

如果要实现与传统WEB界面系统的紧密集成，真正连通，宿主页面可以传入参数以设定APP运行的初始状态，APP可以传出参数报告运行的结果或状态，Shiny APP需要与宿主页面之间建立双向的数据交换通道。这里要解决网页跨JS主域通信的问题。

在开放的互联网公共环境中，为了保障用户的安全，浏览器有个[同源安全性的原则](https://en.wikipedia.org/wiki/Same-origin_policy)，宿主网页与APP网页这样在浏览器中打开的不同网页之间（虽然嵌入为IFRAME显示在同一个页面上，但在浏览器内部的表示中是两个网页，两个窗口），只有来源相同时才可以互相操作（DOM模型及数据）。所谓同源就是浏览器访问网址url的协议+域名+端口都一样。本例Shiny Server部署在http://124.223.110.20:3838 ，宿主Tomcat的网页部署在http://124.223.110.20:8080，所以APP网页与宿主网页是不同源的，这也是Shiny APP要嵌入存量系统网页的经典场景。本例通过HTML5引入的跨文档通信 API（Cross-document messaging）来解决，这个API为window对象新增了一个.postMessage方法，允许跨窗口通信，不论这两个窗口是否同源。具体的实现是，在宿主页面与APP页面中都增加收发消息的javascript脚本。前面已经在APP UI中嵌入了javascript，回头再讲，现在先看看宿主网页testembedded.html。



<!DOCTYPE html>

<html lang="zh-CN" xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Shiny APP 嵌入示例</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<script type="text/javascript">

window.addEventListener('message', receiveMessage);

function receiveMessage(event) {

//alert(event.data);

var passout = document.getElementById('passout');

passout.value = event.data;

}

function setFreq(){

try{

var freq = document.getElementById("freq");

//alert(freq.value);

// document.getElementById("shinyappframe") 失败。

var win = document.getElementsByTagName('iframe')[0].contentWindow;

// postMessage()，不受浏览器同源规则限制，可用于任何内嵌的iframe。

// 第二个参数是接受窗口的源，协议+域名+端口，可用\*表示发给所有窗口。

win.postMessage(freq.value,'\*');

//alert("Sent!");

} catch (error) {

alert(error);

}

}

</script>

</head>

<body>

<div id="titleBar">

<h1>Shiny APP 嵌入示例</h1>

</div>

<div id="shiny">

<div class="caption">

<table>

<tr>

本示例先按《传入》按钮传入数据给嵌入的Shiny APP，记录调用的父窗口，然后才可以按《传出》按钮传出数据给父窗口。<br/>

</tr>

<tr>

<td>mtcars示例</td>

<td><a href = index.html>返回</a></td>

<td>

<form>

传入下标参数：<input id="freq" type="text" value="15"></input>

<input id="passin" type="button" value="传入参数" onclick="setFreq()">

传出mtcars数据：<input id="passout" type="text" value=""></input>

</form>

</td>

</tr>

</table>

</div>

**<h1> Shiny APP在这里以IFRAME的形式嵌入宿主网页</h1>**

<iframe id="shinyappframe" src="http://124.223.110.20:3838/users/jean/updateEvent/" style="border: 1px solid #AAA; width: 100%; height: 600px"></iframe>

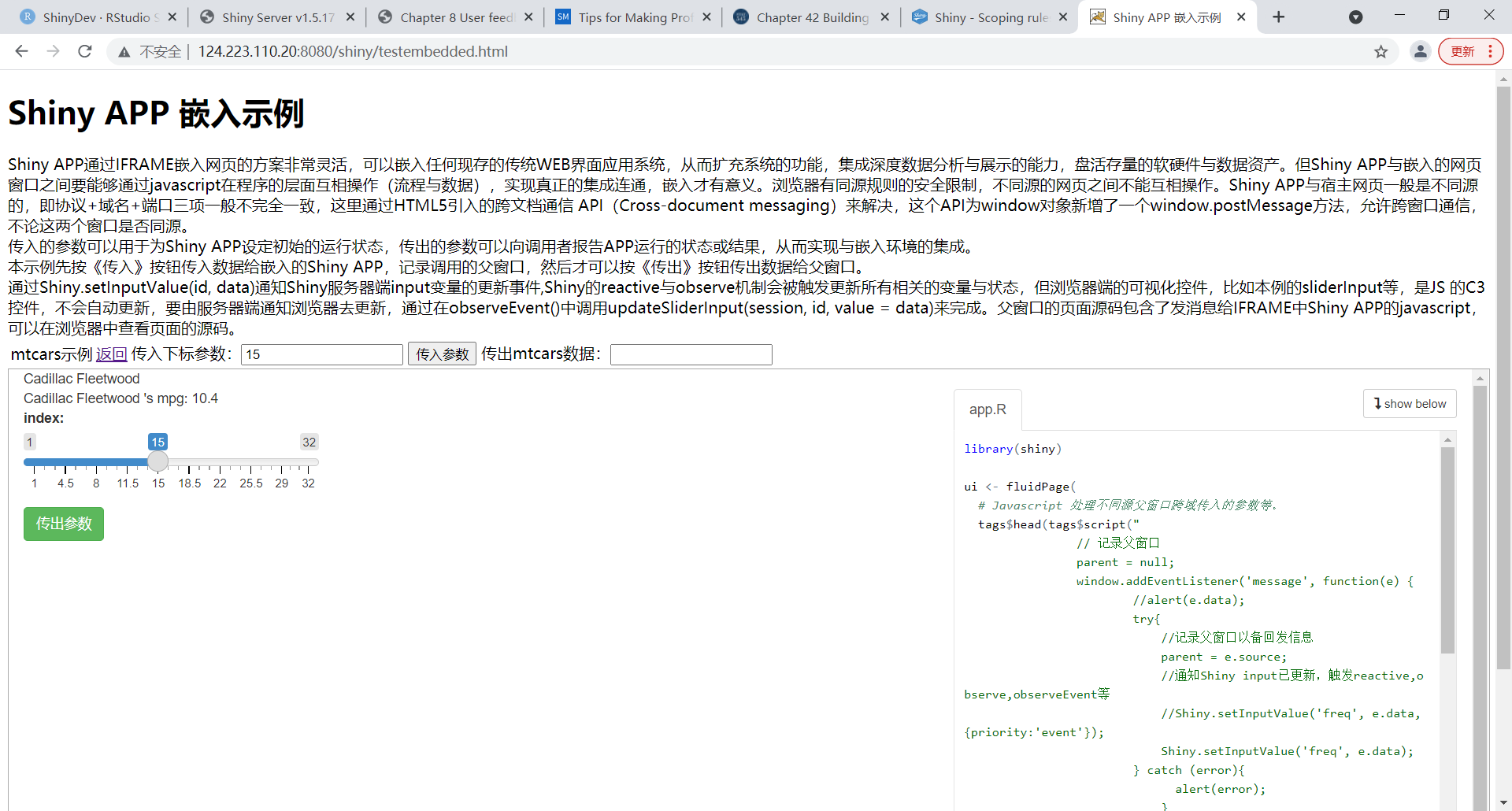
</div>

<div>

</div>

</body>

</html>



本例先按《传入》按钮传入数据给嵌入的Shiny APP，记录调用的父窗口，然后才可以按APP的《传出》按钮传出数据给父窗口。定义了2个javascript函数，setFreq()通过DOM模型取得Shiny APPIFRAME的窗口对象，然后用postMessage()函数把参数以字符串的形式发消息给它。receiveMessage()函数从收到的消息中取出Shiny APP传出的参数，更新到宿主网页的变量中。通过将receiveMessage()注册为message的EventListener，当宿主网页收到消息时，就会激活receiveMessage()。

现在看看APP UI中嵌入的javascript，也是定义了收发消息的函数。稍有不同的是，在接受消息的匿名函数中，把全局变量parent置为消息来源的窗口，以便在后面传出APP的数据。然后调用Shiny内置的函数Shiny.setInputValue()通知服务器交互式输入变量freq发生了改变:

//记录父窗口以备回发信息

parent = e.source;

//通知Shiny input已更新，触发reactive,observe,observeEvent等

Shiny.setInputValue('freq', e.data);

然后会象直接操作界面改变sliderInput一样，触发服务器端的响应变量freq及依赖该响应变量的reactive()代码与observe()代码、observeEvent()代码，在observeEvent()中调用Shiny内置的updateSliderInput()通知浏览器更新sliderInput JS Widget。

observeEvent(freq(),{

print(paste("event triggered:",input$freq))

# 更新浏览器端JS Widget

updateSliderInput(session, "freq", value = freq())

})

传出数据的实现更简单，如果父窗口不为空，则通过DOM模型提取要传出的数据，parent.postMessage()发送消息。

// 点击时传出参数

$(document).on('click', '.btn-success', function (evt) {

evt.preventDefault();

//alert('click!');

if (parent == null){

alert('click!');

} else {

try{

var freq = document.getElementById('freq');

parent.postMessage(freq.value, '\*');

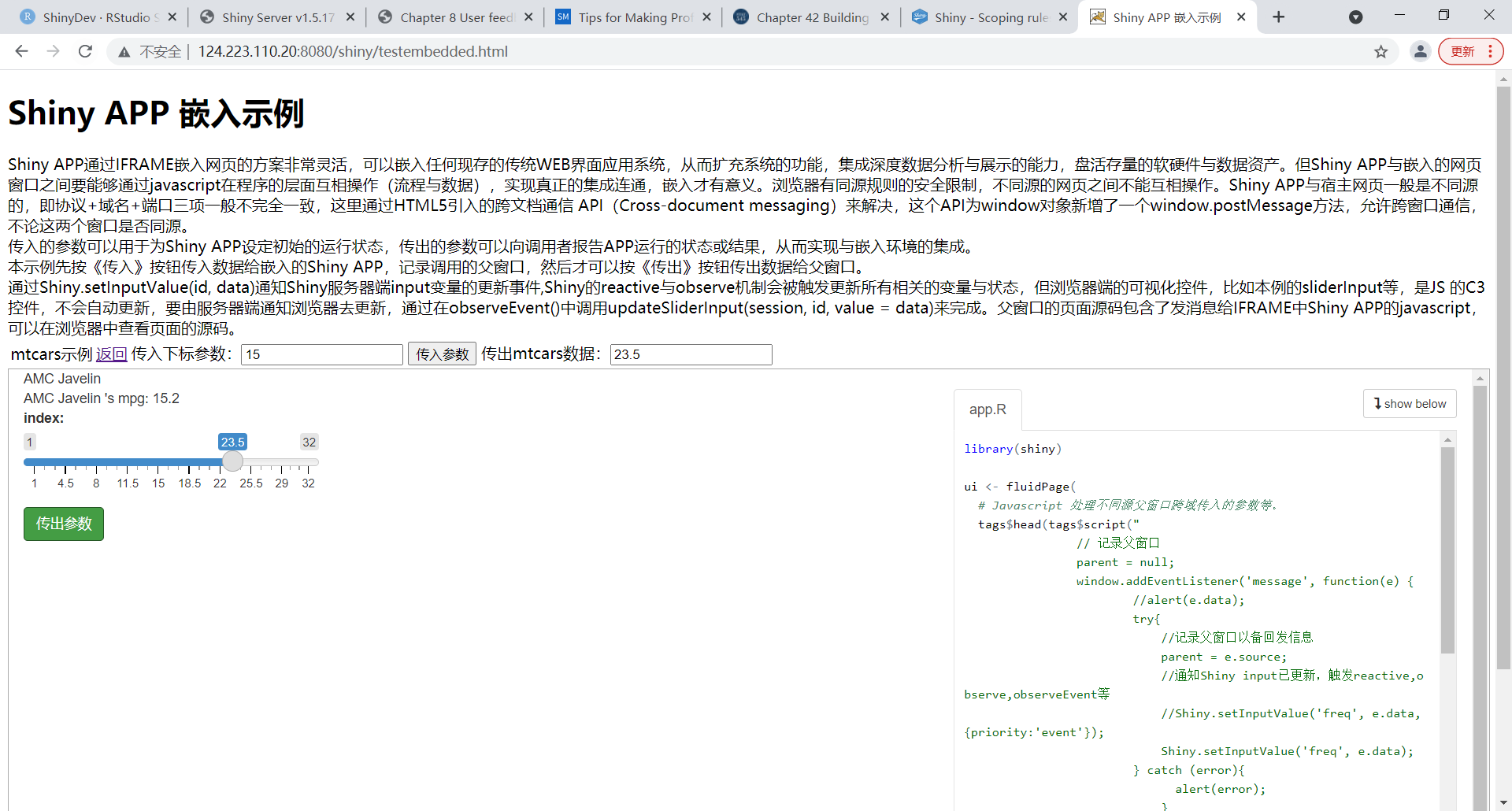
} catch(error){

alert(error);

}

}

});



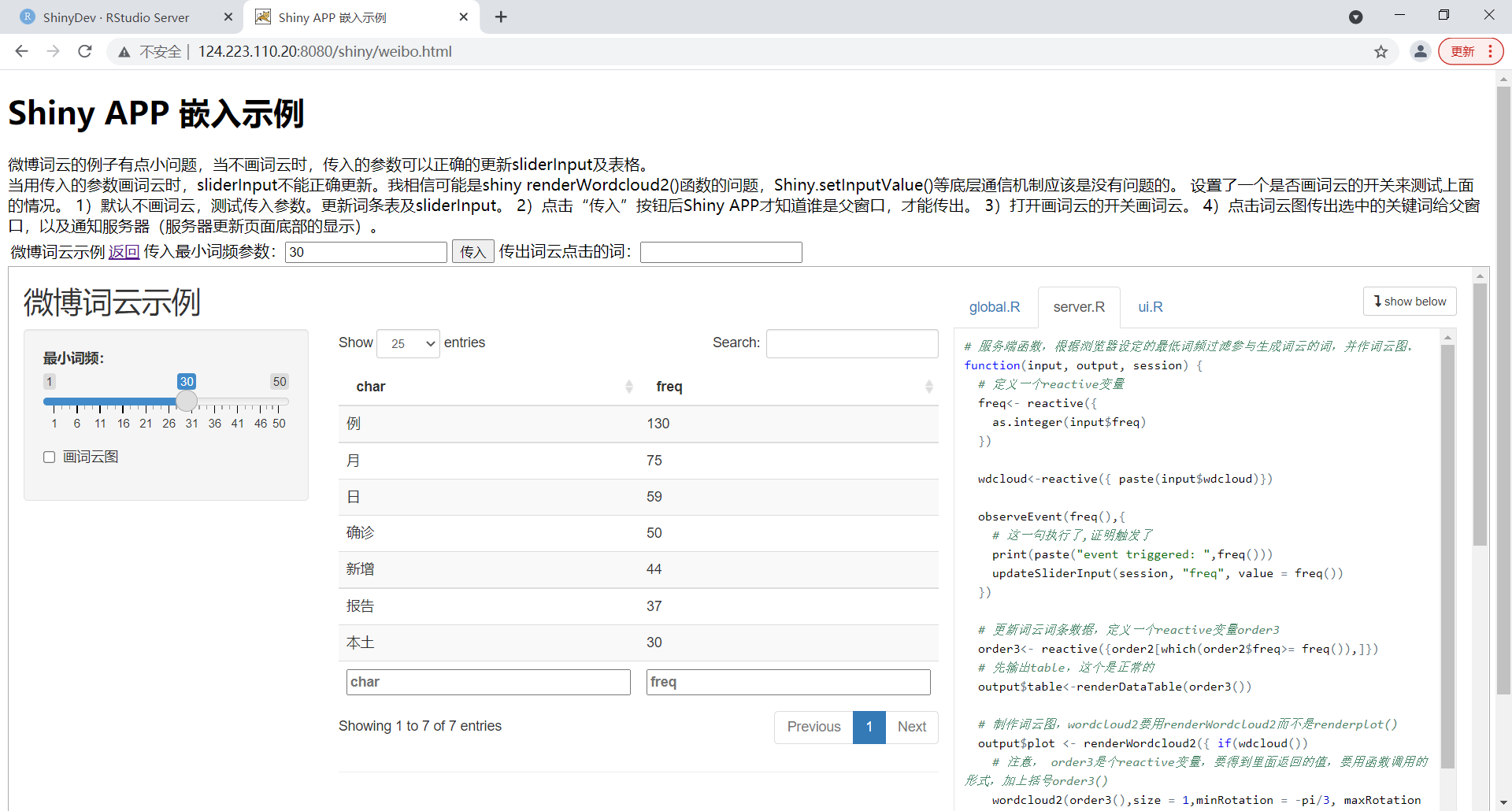
postMessage()发送的是文本消息，收发多个参数时可以封装为json字符串，具体可以参考[这篇文章](https://www.ruanyifeng.com/blog/2016/04/same-origin-policy.html)。

三、[微博词云示例](https://124.223.110.20:8443/shiny/weibo.html)

第十章《文本挖掘》第二节《可视化微博数据挖掘》中有微博词云的例子，那是用Python Orange做的，这里我们看个R语言Shiny APP做的交互式微博词云的简单分析例子，了解大家在微博上关注的热点。也是把Shiny APP嵌入到一个宿主网页中。

本例有4个程序文件,weibo.html是宿主页面，也是部署在Tomcat上，global.R用于存放UI函数与Server函数通用的全局数据，这里用本人的微博账号连接到[微博开放平台](https://open.weibo.com/wiki/%E9%A6%96%E9%A1%B5)上，通过API抓取一些公共的微博数据。ui.R定义了APP的界面，它用一个sliderInput来给出词频阀值，过滤掉阀值以下的词，以使主题更简单明了，然后输出词频表与词云图。定义了一个开关变量，用于决定是否输出词云图。server.R定义了服务器端的逻辑，定义响应变量监视输入变量的变化，输出词频表，画词云图，根据宿主页面传入的参数，通知浏览器更新sliderInput，以及更新词云中点击选中的词，这个作为APP分析的结果要传回宿主页面。先看看运行效果。

传入最低词频参数，过滤词频表。



打开画词云图开关画词云图，传出词云中点击选中的词给宿主页面。



下面来具体看看实现的程序。

1、weibo.html，注意词频阀值freq是number类型，在server.R中要转换为Integer类型。

<!DOCTYPE html>

<html lang="zh-CN" xmlns="http://www.w3.org/1999/xhtml">

<head>

<title>Shiny APP 嵌入示例</title>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<script type="text/javascript">

//接受消息

window.addEventListener('message', receiveMessage);

function receiveMessage(event) {

//alert(event.data);

var passout = document.getElementById('wordSelected');

passout.value = event.data;

}

//发送消息

function setFreq(){

try{

var freq = document.getElementById("freq");

//alert(freq.value);

// document.getElementById("shinyappframe") 失败。

var win = document.getElementsByTagName('iframe')[0].contentWindow;

// postMessage()，不受浏览器同源规则限制，可用于任何内嵌的iframe。

// 第二个参数是接受窗口的源，协议+域名+端口，可用\*表示发给所有窗口。

win.postMessage(freq.value,'\*');

//alert("Sent!");

} catch (error) {

alert(error);

}

}

</script>

</head>

<body>

<div id="titleBar">

<h1>Shiny APP 嵌入示例</h1>

</div>

<div id="shiny">

<div class="caption">

<table>

<tr>

微博词云的例子有点小问题，当不画词云时，传入的参数可以正确的更新sliderInput及表格。<br>

当用传入的参数画词云时，sliderInput不能正确更新。我相信可能是shiny renderWordcloud2()函数的问题，Shiny.setInputValue()等底层通信机制应该是没有问题的。

设置了一个是否画词云的开关来测试上面的情况。

1）默认不画词云，测试传入参数，更新词条表及sliderInput。

2）点击“传入”按钮后Shiny APP才知道谁是父窗口，才能传出。

3）打开画词云的开关画词云。

4）点击词云图传出选中的关键词给父窗口，以及通知服务器（服务器更新页面底部的显示）。

</tr>

<tr>

<td>微博词云示例</td>

<td><a href = index.html>返回</a></td>

<td>

<form>

传入最小词频参数：<input id="freq" type="number" value="15"></input>

<input id="passin" type="button" value="传入" onclick="setFreq()">

传出词云点击的词：<input id="wordSelected" type="text" value=""></input>

</form>

</td>

</tr>

</table>

</div>

<iframe id="shinyappframe" src="http://124.223.110.20:3838/users/jean/weibo/" style="border: 1px solid #AAA; width: 100%; height: 600px"></iframe>

</div>

</body>

</html>

2、global.R

global.R只在APP第一次加载时执行一次，其中的变量对ui.R与server.R都可见，一般用于初始化APP及存放全局共享的数据。所以本例只在APP初始化时抓取一次微博，重启前不会更新。要每个新建立的session都抓取一次，则需要把代码放在server函数中，reactive代码之外。不过Rweibo包在那里运行有问题，因为是演示性质，暂时只抓取一次。

#微博词云实例

# 参阅 https://zhuanlan.zhihu.com/p/82166549，shiny中使用wordcloud2要用一些特殊的函数。

# https://gist.github.com/AdamSpannbauer/135793c68b90b46f44dbe50364c0edf5

# 开启showcase 模式可以参阅 https://blog.csdn.net/douerw/article/details/115350908

# runApp("weibo",display.mode= "showcase", port=3839)

library(Rweibo)

library(jiebaR)

library(tm)

library(wordcloud2)

#注册应用，只需运行一次，写入本地配置文件

# registerApp(app\_name = "RJean", "appid", "access key")

#取得授权，有效期间只需运行一次，写入本地配置文件

# roauth <- createOAuth(app\_name = "RJean", access\_name = "rweibo",

# login = TRUE, username = "username", password = "password")

# 创建授权对象，用缓存的有效授权

roauth <- createOAuth("RJean", "rweibo")

#微博转换为数据框

weibo2dataframe<-function(res){

#i<-1

dt<- data.frame(uid= character(), name= character(),screen\_name= character(), id= character(),text= character(), created\_at= character())

for(i in 1:length(res)){

tmp<- data.frame(uid= res[[i]]$user$idstr, name= res[[i]]$user$name,screen\_name= res[[i]]$user$screen\_name,

id= res[[i]]$idstr,text= res[[i]]$text, created\_at= res[[i]]$created\_at)

dt<-rbind(dt,tmp)

}

return (dt)

}

#循环抓取一些微博，转成数据框，做文本挖掘测试

wdt<- data.frame(uid= character(), name= character(),screen\_name= character(),id= character(),text= character(), created\_at= character())

for(i in 1:10){

res<- statuses.friends\_timeline(roauth,page=i, count = 100)

if(length(res)>0){

tmp<-weibo2dataframe(res)

wdt<-rbind(wdt,tmp)

cat(length(res))

cat("\n")

}else{cat(i); cat("\n");break}

}

#只看中央媒体的主题,要先加关注

#wdt2<-wdt[which(wdt$name %in% c("央视新闻","人民日报","人民网","新浪新闻","新华网","头条新闻","中国新闻网","环球时报","新浪博客","广东税务","珠海税务")),]

wdt2<-wdt

#把微博文本向量转换为语料库

ovid <- Corpus(VectorSource(wdt2$text))

# 之后要对每一条微博进行处理，正则匹配去掉@，去掉标点，去掉里面出现的图片等

s1 <- gsub('[a-zA-Z0-9]','',ovid)

s1 <- gsub('[\\pP+~$`^=|<>～｀＄＾＋＝｜＜＞￥×]','',s1)

s1 <- gsub('①|②|③|④|⑤|⑥|⑦|⑧|⑨|℃|↓|→|丨','',s1)

#去掉各种副词

s1<-gsub("[的|和|了|来|与|到|由|等|从|以|一|为|在|上|各|去|对|侧|多|并|千|万|年|更|向|这是]","",s1)

#分词

seg<-worker()

seg<=s1

#建立词频

freq2<-freq(segment(s1,seg))

#按词频排序

index <- order(-freq2[,2])

order2<<-freq2[index, ]

3、ui.R

增加了一小段javascript,根据选中的词动态生成一个input参数传递到服务器端告知服务器，并发消息给父窗口，通知选中了哪个词，两头都通知。WordCloud2词云中点击选中关键词的DOM对象id是“wcSpan”。

# 浏览器端UI函数，选择最低词频，更新显示词云图。

fluidPage(

# Javascript 处理父窗口传入的参数等。

tags$head(tags$script("

// 记录父窗口，初始为空。

parent = null;

// 处理接收到的消息。

window.addEventListener('message', function(e) {

//alert(e.data);

try{

//记录父窗口以备回发信息

parent = e.source;

//向服务器发送input变量更新消息

Shiny.setInputValue('freq', e.data, {priority:'event'});

} catch (error){

alert(error);

}

},false);

//发送消息给父窗口

function sendMessage(msg){

if (parent == null){

//alert('No outer frame!');

} else {

try{

//这是服务器端返回的词云中点击选中的词，output$selected\_word

var selected = document.getElementById('selected\_word');

parent.postMessage(msg, '\*');

//alert('sent!');

} catch(error){

alert(error);

}

}

}

")),

# Application title

titlePanel("微博词云示例"),

sidebarLayout(

# Sidebar with a slider and selection inputs

sidebarPanel(

sliderInput("freq", "最小词频：", min = 1, max = 50, value = 10),

checkboxInput("wdcloud","画词云图", value = FALSE)

),

# Show Word Cloud

mainPanel(

# 输出词频表

dataTableOutput('table'),

hr(),

# wordcloud2要用wordcloud2Output而不是plotOutput()

wordcloud2Output("plot", width = "100%", height = "600px"),

hr(),

# 根据选中的词用javascript动态生成一个input参数传递到服务器端。

# 并发消息给父窗口，通知选中了哪个词。

tags$script(HTML(

"$(document).on('click', '#canvas', function() {",

'word = document.getElementById("wcSpan").innerHTML;',

"Shiny.onInputChange('selected\_word', word);",

"sendMessage(word);",

"});"

)) ,

# 服务器端返回这个选中的词，并显示出来

textOutput("selected\_word"),

hr()

)

)

)

4、server.R

# 服务端函数，根据浏览器设定的最低词频过滤参与生成词云的词，并作词云图。

function(input, output, session) {

# 定义一个reactive变量监视词频阀值的变化，注意是Integer类型。

freq<- reactive({

as.integer(input$freq)

})

# 定义一个reactive变量监视是否画词云图开关的变化

wdcloud<-reactive({ paste(input$wdcloud)})

# 根据传入参数更新sliderInput的状态

observeEvent(freq(),{

# 这一句执行了,证明触发了

print(paste("event triggered: ",freq()))

updateSliderInput(session, "freq", value = freq())

})

# 更新词云词条数据，定义一个reactive变量order3

order3<- reactive({order2[which(order2$freq>= freq()),]})

# 先输出词条表，这个是正常的

output$table<-renderDataTable(order3())

# 制作词云图，wordcloud2要用renderWordcloud2而不是renderplot()

output$plot <- renderWordcloud2({ if(wdcloud())

# 注意， order3是个reactive变量，要得到里面返回的值，要用函数调用的形式，加上括号order3()

wordcloud2(order3(),size = 1,minRotation = -pi/3, maxRotation = pi/3,rotateRatio = 0.8,fontFamily = "微软雅黑", color = "random-light", shape = 'star')

})

#接收浏览器端点击选中的词并返回浏览器，显示在网页底端。

observe({ if(wdcloud())

if (!is.null(input$selected\_word)) {

print(input$selected\_word)

#using default clicked word input id

output$selected\_word = renderText(input$selected\_word)

}

})

}

第二节 [Shiny APP连接云端](https://124.223.110.20:8443/shiny/radiant.html)

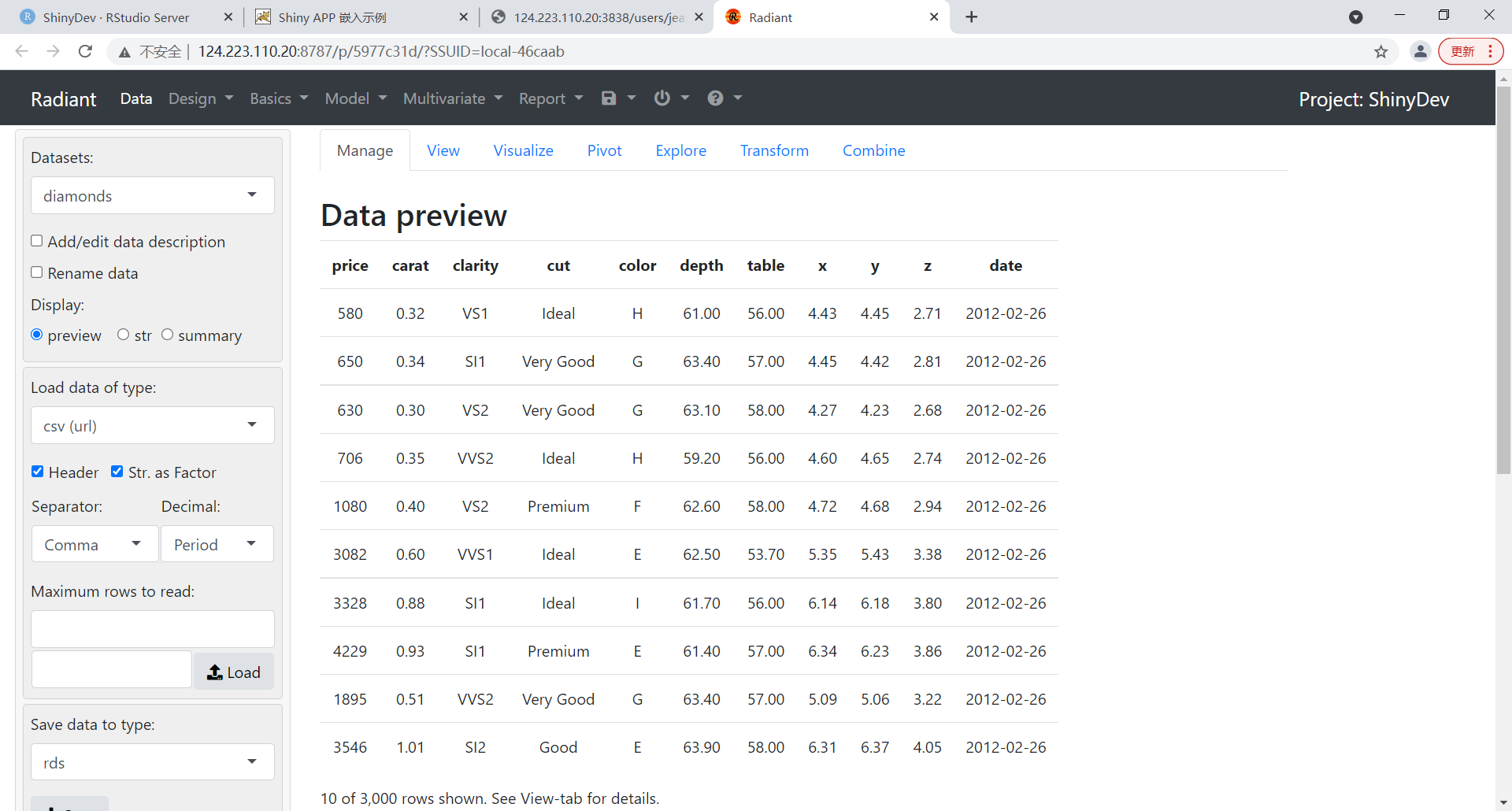
Shiny APP属于中间层的应用，上一节嵌入宿主网页解决了与应用层集成的问题，打通了任脉。本节看看Shiny APP连接云端，打通不同来源的数据源，连接数据层的示例。本例将使用Shiny APP Radiant, 功能类似于客户端的Rattle，是[加州大学圣迭戈分校的Rady管理学院](https://rady.ucsd.edu/)用于R语言机器学习教学的软件，可以从R命令行用install.packages(“radiant”)安装。该APP比较复杂，功能强大，还没有深入研究其源码，所以不用以演示传入传出参数集成，也不在其基础上开发程序扩增功能，而是用hack的方式，作最小程度的修改，连接阿里云与华为云加载数据，以使用云端的大数据与算力，这是本省税务系统都有使用的云端平台。Shiny平台在这里提供了一条超越内卷的途径，通过RODPS包直连阿里云，通过reticulate包调用Python脚本连接华为云，正如上一节R连接微博一样，R与Python共同提供了Shiny连接各种数据库（如Oracle）、图数据库（如Neo4j）、GIS、NLP等各种数据源的途径，也提供了充分利用R及Python强大的数据分析能力、作图能力、机器学习、深度学习能力的途径，让用户可以超越平台及语言的内卷，按自己的需要和习惯去驾驭大数据，从而把精力集中在业务和分析上，最有效率的做事。这是打通了督脉。

在前面的章节中已经可以看到，解决发票大数据分析的基础性问题，发票货劳名称识别用的是HanLP（使用深度学习框架tensorflow或pytorch），交易网络分析用的是Neo4j，人们要看到这样一个客观的现实，没有一个厂商或者一个平台是全能的，寸有所长，尺有所短。但现实的落地应用需要各种各样的能力，这就是说作为应用中台的中间层，需要有整合各厂商产品的能力，作为业主的甲方，尤其是政府部门，需要有选择的能力与空间。从这个视角来看，Shiny无疑是个可以胜任的平台。有志于此的厂商，需要对标国际水准，把压力转换为动力，做出更好的产品，建立开放的生态。万类霜天竞自由，只有开放的生态才是生机勃发，创新迭出的，也只有开放的生态才能激发各行各业万众创新的热情和能量。

R语言与Python深度数据分析提供的是裂变的能量，云端大数据平台提供的是聚变的能量，Shiny平台把二者连接在一起，打通任督二脉后，开发的APP，如Radiant等，就会爆发出氢弹一样的能量。

下面具体看看，先看看Radiant。在Rsutdio或R命令行中，用radiant::radiant()就可以启动它，默认会载入自带的diamonds数据集，它提供了各种对载入数据集探索、统计分析、作图、机器学习建模等深度分析的可视化功能。因为是WEB界面，在手机上也可以运用自如，非常方便。

其中的load data of type下拉列表中，可以从csv，csv(url)等多种不同的数据源载入数据。下来要做的是，稍为hack一下数据载入部分的源码，从csv(url)加载数据时，如果url以“.R”结尾，就从R语言脚本加载数据，这为我们提供了连接各种数据源加载数据，并作所有需要的预处理以生成特征矩阵的可能。当然也可以规定url以“.py”结尾时从Python脚本加载数据，不过既然Shiny是R语言的，需要通过reticulate包去加载Python脚本，就只处理“.R”后缀好了，当需要从Python脚本中加载时，可以在一个R脚本中封装它。



为了hack Radiant的源码，需要把radiant包安装目录下~/inst/app拷贝到Rstudio IDE的开发环境中home目录下（重命名为radiant），Radiant数据加载的部分在radiant.data包中，也要把radiant.data包安装目录下~/inst/app拷贝到Rstudio IDE的开发环境中（重命名为radiant.data），这样就可以在Rstudio IDE中修改和调试源码。

把Radiant部署到Shiny Server上运行的方法是，把radiant包安装目录下~/inst/app拷贝到Shiny Server的主目录下并改个名，比如radiant，具体可参阅[Radiant的文档](https://radiant-rstats.github.io/docs/)。本例中要修改radiant.data包的~/tools/data/manage\_ui.R，调试改好后，需要拷贝到该包的安装目录下替换原来的文件，以便在所有的运行环境中都生效。

一、为Radiant增加从R脚本载入数据的入口

在对radiant.data作了初步的分析后，确定了从csv(url)加载数据的操作在该包~/tools/data/manage\_ui.R中的事件响应函数

observeEvent(input$url\_csv\_load, {})

中处理，可以看到input$url\_csv\_load是菜单Data->Manage所属的UI函数下定义的一个交互式输入变量，对应着UI上的load按钮。这个函数很好改，为了便于比较，用注释的形式列出原来的代码，绿色显示，增加的代码红色显示，其余不改的代码黑色显示，数据集加载后就以脚本的名字命名。主要的处理是，如果url以“.R”结尾，就调用load\_fromRscript()函数从R脚本载入数据，该函数从url装入R脚本并调用其中的load\_data()函数，load\_data()要返回一个R语言的data frame，这是我定义的约定。

# Added by Jean 2022/02/25

# Load dataset from external R script

load\_fromRscript<- function(url){

source(url)

# load\_data() should return a data frame

return(load\_data())

}

observeEvent(input$url\_csv\_load, {

## loading csv file from url, example https://radiant-rstats.github.io/docs/examples/houseprices.csv

if (radiant.data::is\_empty(input$url\_csv)) return()

url\_csv <- gsub("^\\s+|\\s+$", "", input$url\_csv)

# Modified by Jean 2022/02/25

# objname <- basename(url\_csv) %>% sub("\\.csv", "", .) %>% sub("\\?.\*$", "", .)

# if (!objname == radiant.data::fix\_names(objname)) {

# objname <- "csv\_url"

# }

#

# dataset <- try(load\_csv(

# url(url\_csv),

# delim = input$man\_sep,

# col\_names = input$man\_header,

# n\_max = input$man\_n\_max,

# dec = input$man\_dec,

# saf = input$man\_str\_as\_factor

# ), silent = TRUE)

# Added by Jean 2022/02/25 for debugging

type<-substr(input$url\_csv,nchar(input$url\_csv)-1,nchar(input$url\_csv))

print(paste("Type",type, sep=":"))

if (type == ".R"){

print(paste("Load data from external R script ",sep=":"))

objname <- basename(url\_csv) %>% sub("\\.R", "", .) %>% sub("\\?.\*$", "", .)

if (!objname == radiant.data::fix\_names(objname)) {

objname <- "R\_script\_url"

}

dataset <- try(load\_fromRscript(input$url\_csv

), silent = TRUE)

} else {

objname <- basename(url\_csv) %>% sub("\\.csv", "", .) %>% sub("\\?.\*$", "", .)

if (!objname == radiant.data::fix\_names(objname)) {

objname <- "csv\_url"

}

dataset <- try(load\_csv(

url(url\_csv),

delim = input$man\_sep,

col\_names = input$man\_header,

n\_max = input$man\_n\_max,

dec = input$man\_dec,

saf = input$man\_str\_as\_factor

), silent = TRUE)

}

cmd <- ""

if (inherits(dataset, "try-error") || is.character(dataset)) {

# Modified by Jean 2022/02/25 for better debugging.

# upload\_error\_handler(objname, "#### There was an error loading the csv file from the provided url")

upload\_error\_handler(objname, paste("Error", dataset[1],getwd(), sep=":"))

} else {

r\_data[[objname]] <- dataset

## generate command

delim <- input$man\_sep

col\_names <- input$man\_header

dec <- input$man\_dec

saf <- input$man\_str\_as\_factor

n\_max <- input$man\_n\_max

n\_max <- if (is\_not(n\_max) || n\_max < 0) Inf else n\_max

if (delim == "," && dec == "." && col\_names == FALSE) {

cmd <- glue('

{objname} <- readr::read\_csv(

"{url\_csv}",

n\_max = {n\_max}

)')

} else {

cmd <- glue('

{objname} <- readr::read\_delim(

"{url\_csv}",

delim = "{delim}", col\_names = {col\_names}, n\_max = {n\_max},

locale = readr::locale(decimal\_mark = "{dec}", grouping\_mark = "{delim}")

)')

}

cmd <- paste0(cmd, " %>%\n fix\_names()")

if (saf) cmd <- paste0(cmd, " %>%\n to\_fct()")

cmd <- glue('{cmd}\nregister("{objname}")')

}

if (exists(objname, envir = r\_data) && !bindingIsActive(as.symbol(objname), env = r\_data)) {

shiny::makeReactiveBinding(objname, env = r\_data)

}

r\_info[["datasetlist"]] <- c(objname, r\_info[["datasetlist"]]) %>% unique()

r\_info[[paste0(objname, "\_descr")]] <- fix\_smart(attr(r\_data[[objname]], "description"))

r\_info[[paste0(objname, "\_lcmd")]] <- cmd

updateSelectInput(

session, "dataset",

label = "Datasets:",

choices = r\_info[["datasetlist"]],

selected = r\_info[["datasetlist"]][1]

)

})

二、从阿里云载入数据

阿里云有R语言接口包RODPS，载入数据是很直接的。本例从阿里云载入航班乘客数据集airpassengers,脚本是load\_from\_ODPS\_Example.R。

# R语言脚本连接阿里云MaxCompute载入特征矩阵数据，

# 返回一个data frame，函数是指定的名称load\_data()

# 访问MaxCompute需要的 access key、secret key等另外存储在配置文件中。

library(RODPS)

rodps.set("odps.sql.type.system.odps2","TRUE")

rodps.set("odps.sql.decimal.odps2","TRUE")

rodps.set("odps.sql.hive.compatible","FALSE")

rodps.set("odps.sql.allow.fullscan","true")

load\_data<- function(){

df<- rodps.sql("select \* from airpassengers;")

# 对字符串型日期稍作处理

df$yy<- as.integer(substr(df$month,1,4))

df$mm<- as.integer(substr(df$month,6,7))

# 下标可以看作是时间序列顺序，用于作图

df$index<-as.integer(row.names(df))

# 返回一个 data frame

return(df)

}

这里要讲一下url中脚本路径的问题。Shiny Server的主目录是/srv/shiny-server，Radiant部署在它子目录radiant下，R与Python脚本放在它的子目录scripts下。Radiant APP启动后，它的工作目录就是自己，所以输入的R脚本url要翻到父目录，写成

file://../scripts/load\_from\_ODPS\_Example.R

因为还没有通读Shiny的文档，不了解Shiny APP是否可以访问主目录之外的文件，所以把R与Python脚本放在scripts子目录下，这会引起安全的问题，有些脚本大概是不想让公众随便看的，可以把scripts子目录的访问权限设置为770来暂时解决。这里是演示性质大家随便看。

数据载入的效果后面一起再看。

三、从华为云载入数据

华为云Data Lake Insight只有Python SDK，可以从Python脚本载入数据,然后用一个R脚本通过reticulate包调用它。先看看Python脚本load\_from\_DLI.py，程序比较直观，最后生成一个pandas data frame。

# -\*- coding: utf-8 -\*-

# Python连接华为云DLI载入特征矩阵数据，返回给Shiny APP Radiant

from dli.dli\_client import DliClient

# 这是自己添加的自定义python模块，

# 存放访问DLI的access key与secret key，因为源码是网上可见的。

from dli import dlikeys

# print(dlikeys.dli\_project\_id)

# print(dlikeys.dliAK)

# print(dlikeys.dliSK)

import time

import pandas as pd

def init\_aksk\_dli\_client():

auth\_mode = 'aksk'

region = 'cn-south-1'

project\_id = dlikeys.dli\_project\_id

ak = dlikeys.dliAK

sk = dlikeys.dliSK

dli\_client = DliClient(auth\_mode=auth\_mode, region=region, project\_id=project\_id,ak=ak, sk=sk)

return dli\_client

def run\_sql(dli\_client, db\_name, queue\_name, sql):

# execute SQL

try:

sql\_job = dli\_client.execute\_sql(sql, db\_name)

result\_set = sql\_job.get\_result()

except DliException as e:

print(e)

return

if result\_set.row\_count == 0:

return None

return result\_set

dli\_client = init\_aksk\_dli\_client()

db\_name = 'geshui'

queue\_name = 'default'

t1 = time.time()

month = [] ; passengers=[]

sql = "select month,passengers from airpassengers;"

result\_set = run\_sql(dli\_client, db\_name, queue\_name, sql)

print("loading data......")

for row in result\_set:

cols = row.columns

month.append(cols[0])

passengers.append(cols[1])

t2 = time.time()

print("time: ",t2-t1)

# 从 list建立 pandas data frame

data = pd.DataFrame({"month":month,"passengers":passengers})

再看看调用它的R脚本load\_from\_DLI.R：

# R语言脚本连接华为云 DLI载入特征矩阵数据，

# 返回一个data frame，函数是指定的名称load\_data()

# R语言连接Python用reticulate包。

library(reticulate)

load\_data<- function(){

# 根据当前工作目录判断是Rstudio IDE开发环境还是Shiny Server部署环境

location<- substr(getwd(),1,4)

# 调用Pyhton脚本连接华为云 DLI，它没有R接口包。

# Rstudio 开发环境是 /usr/lib64/R-4.1.2/lib64/R/library/radiant

if (location == '/usr'){ # Rstudio 开发环境 在 ~/scripts下

part<- '~'

# Shiny Server部署环境是 /srv/shiny-server/radiant

}else{ # 服务器部署环境，在 ../scripts下

part<- '..'

}

# 定位Python脚本目录

path<- paste(part,"/scripts/","load\_from\_DLI.py",sep="")

# 运行Python脚本

py\_run\_file(path)

# 访问Python脚本中的pandas data frame

df<- py$data

# 对字符串日期稍作处理

df$yy<- as.integer(substr(df$month,1,4))

df$mm<- as.integer(substr(df$month,6,7))

df$index<-as.integer(row.names(df))

# 返回一个R语言data frame

return(df)

}

稍为解释一下，R脚本与Python脚本都放在scripts子目录下，开发环境用户home目录与Shiny Server部署环境主目录下各有一个。R脚本通过不同的前缀在不同的环境去定位并运行匹配的Python脚本。

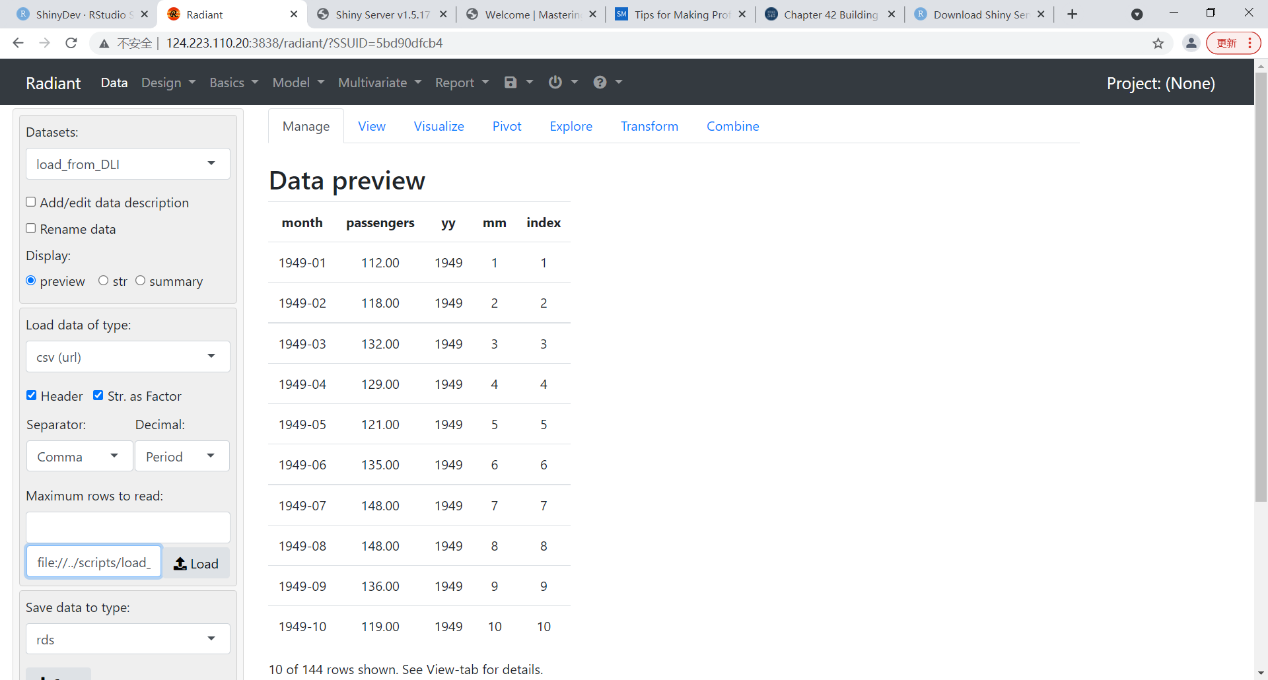
注意R脚本中访问Python脚本中变量的方式，是通过加前缀直接引用。reticulate包调用Python脚本时，会启动一个Python进程，直接连通它，通过reticulate包两个脚本中的变量是可以互相直接引用的，具体可以参阅[参考资料1](https://zhuanlan.zhihu.com/p/245985092)，[参考资料2](https://blog.csdn.net/qq_31342997/article/details/89433255)，以及reticulate包的[文档1](https://rstudio.github.io/reticulate/)与[文档2](https://rstudio.github.io/reticulate/articles/calling_python.html)。

从华为云载入数据示例的url是：

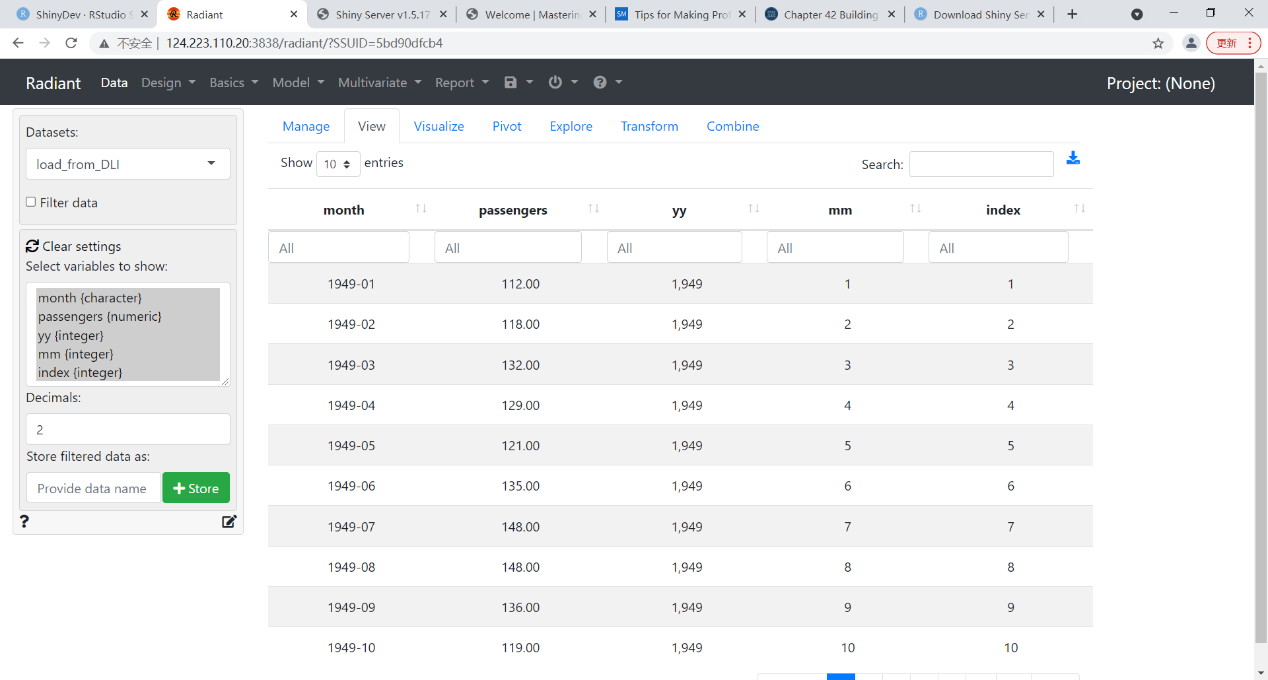
file://../scripts/load\_from\_DLI.R

1. 从R脚本载入数据并探索

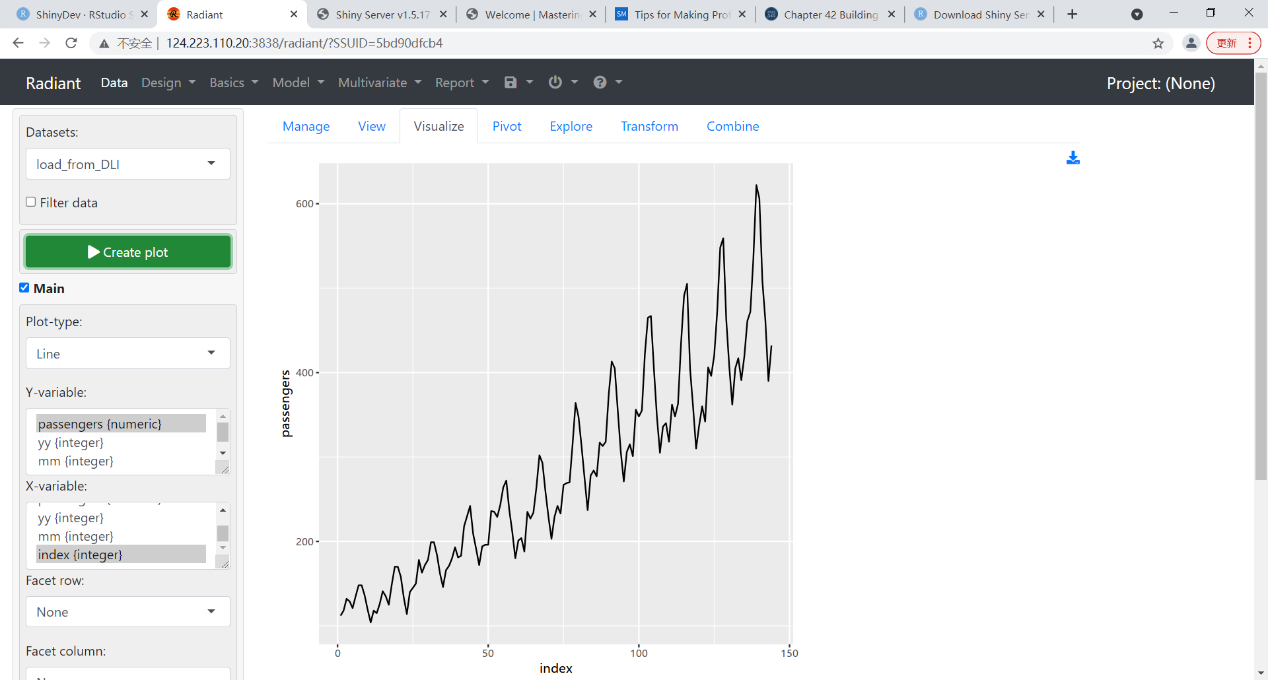
1、载入效果，从阿里云与华为云载入的都是airpassengers数据集。



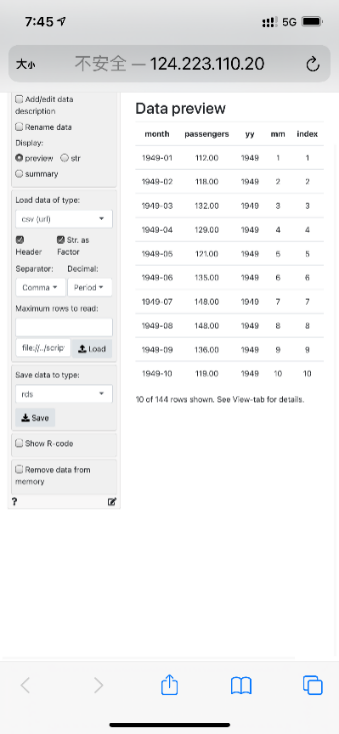
2、切换到View Tab浏览一下数据。



3、切换到Visualize Tab画个时序图，前面已经把行号变成顺序索引。



4、手机上运行的效果。

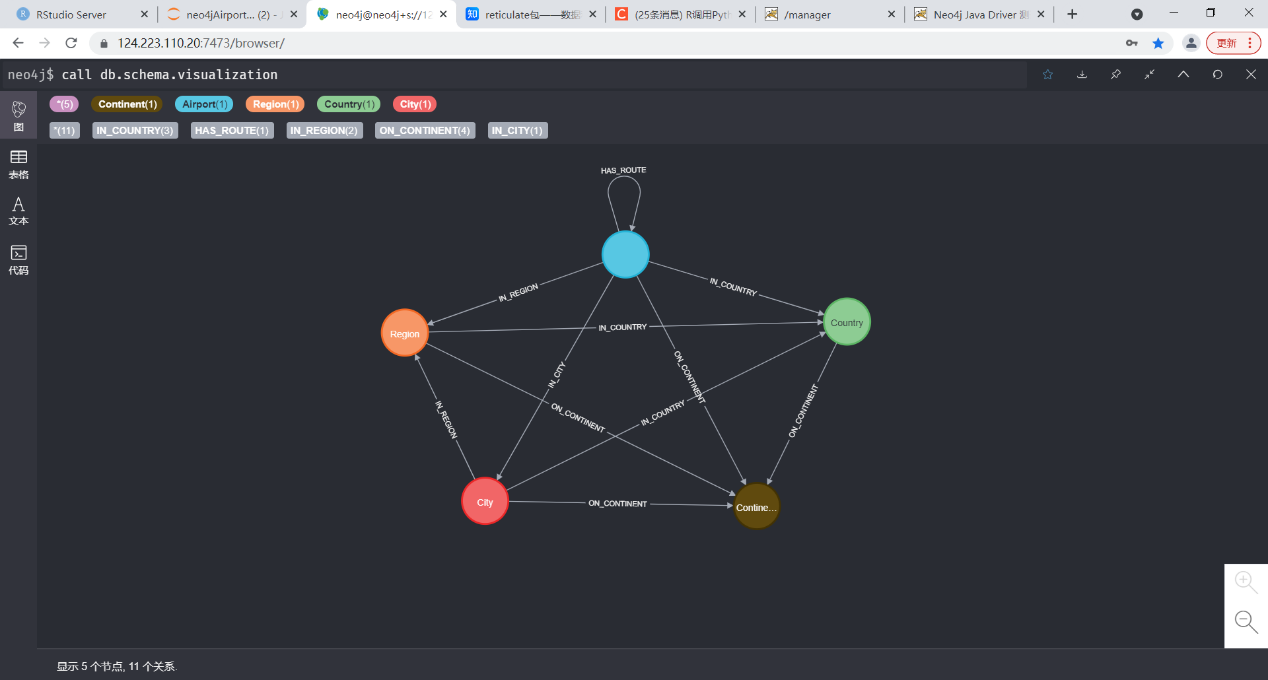


第三节 Shiny 网络分析APP

第二章《发票交易网络分析》中介绍了用Neo4j对发票交易网络做各种分析，或者是后台写脚本做，或者是在Neo4j Browser中做，都需要懂技术写脚本。可以把一些开发好的网络分析模型，用Shiny APP封装，发布给相关的业务人员使用，这样发票交易网络分析从端到端的路径就打通了。下面以机场航线网络为例，看看实现的过程。这个例子APP根据给定的一个国内机场，返回指定跳数内可以通达的国内机场网络，然后在浏览器中可视化的分析探索，选择感兴趣的机场，返回给调用该APP的宿主网页。

一、数据模型

本例的数据集来自 [Neo4j GDS的教学案例](https://github.com/neo4j-graph-examples/graph-data-science2)，它记录了世界各城市机场之间的航线网络，下面是其中的节点（实体）类型和关系模型图，关系HAS\_ROUTE记录了机场之间的航线，所以它和发票交易网络一样，是个加权有向图。

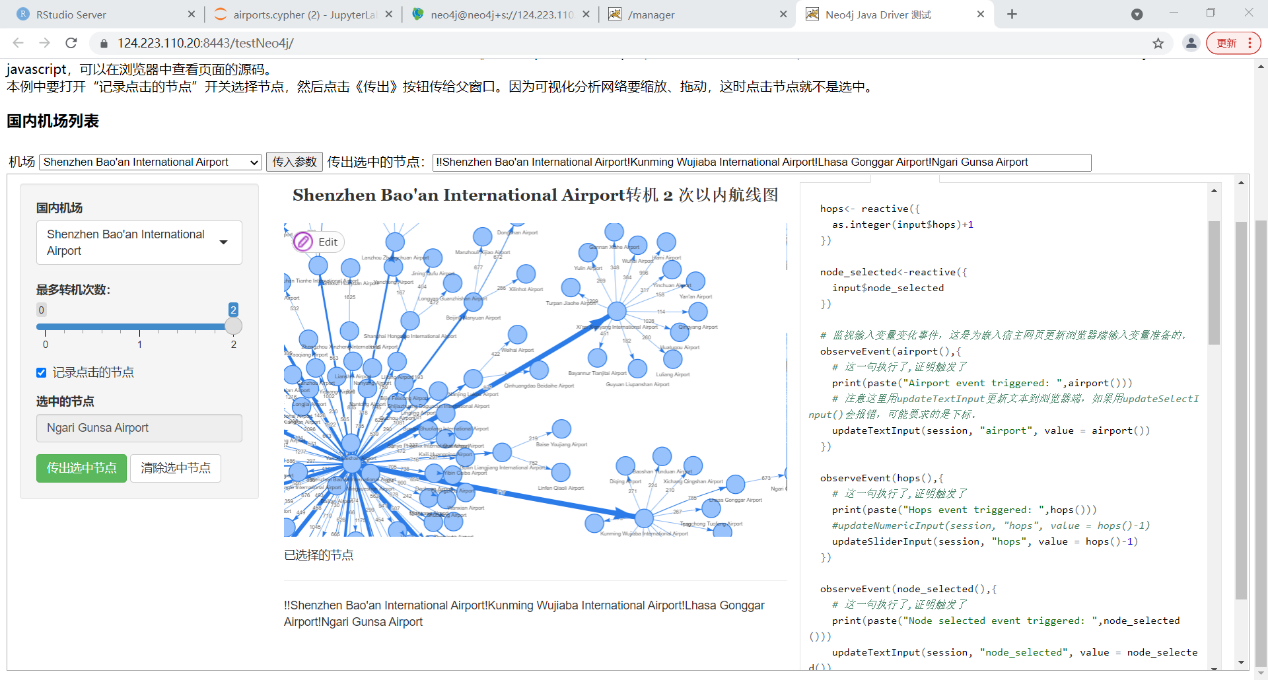


二、表示层

B/S多层应用中浏览器端的UI属于表示层，它运行html，css，javascript，websocket等，完成客户端的业务逻辑。它们由应用层的J2EE服务器等生成。本例的WEB APP跑在Tomcat上，只有一个页面index.jsp，它通过IFRAME嵌入了运行在Shiny Server上的网络分析示例APP，然后通过HTML5的postMessage()机制双向交换数据。

1、运行效果

1）从下拉列表中选择一个机场作为参数传入Shiny APP。  
2）点击“传入参数”按钮传入。  
3）可视化探索从该机场出发的航线网络，可以滚动鼠标滚轮缩放，按住节点和边拖放，以便分析节点和边。  
4）点击“记录点击的节点”开关，再点击记录感兴趣的节点。  
5）点击“传出”按钮返回选中的节点列表作为分析结果给调用它的父窗口。



2、index.jsp

1）先直接用java连接Neo4j执行一个Cypher查询抽取网络中国内的机场列表，在jsp中生成一个下拉列表，供选择传入Shiny APP作为初始参数。

2）通过HTML5的postMessage()机制与嵌入的Shiny APP交换数据，收发的是字符串，多个参数也可以封装为JSON格式。

3）通过IFRAME嵌入Shiny APP。

表示层中涉及到Shiny APP UI及与父窗口通讯的html与javascript会嵌入Shiny APP的R语言程序中，后面再介绍。

<%@ page language=*"java"* contentType=*"text/html; charset=GBK"*

pageEncoding=*"GBK"*%>

<%@ page import=*"java.util.List,java.util.ArrayList,org.neo4j.driver.\*,static org.neo4j.driver.Values.parameters"*%>

<%

Driver driver = GraphDatabase.driver("bolt+ssc://124.223.110.20:7687", AuthTokens.basic("neo4j", "Jean!2022@tencent"));

Session neo4j\_session = driver.session();

Transaction tx = neo4j\_session.beginTransaction();

String cypherQuery = "match (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:$country}), (n:Airport)-[:HAS\_ROUTE]->(target) return distinct n.descr as name; ";

List<Record> result = **new** ArrayList<Record>();

Result res = tx.run(cypherQuery, parameters("country", "CN"));

**while** (res.hasNext()) {

Record row = res.next();

result.add(row);

}

tx.commit();

tx.close();

**for** (Record record : result) {

System.out.println(record.get("name").asString());

}

driver.close();

%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"GBK"*>

<title>Neo4j Java Driver 测试</title>

<script type=*"text/javascript"*>

//接受消息

window.addEventListener('message', receiveMessage);

**function** receiveMessage(event) {

//alert(event.data);

**var** nodesSelected = document.getElementById('nodesSelected');

**var** message = event.data.trim();

nodesSelected.value = message;

**if** (message.length >30){

nodesSelected.size = message.length;

}**else**{

nodesSelected.size = 30;

}

}

//发送消息

**function** setAirport(){

**try**{

**var** airport = document.getElementById("airport");

//alert(airport.value);

// document.getElementById("shinyappframe") 失败。

**var** win = document.getElementsByTagName('iframe')[0].contentWindow;

// postMessage()，窗口间通信，不受浏览器同源规则限制，可用于任何内嵌的iframe。

// 第二个参数是接受窗口的源，协议+域名+端口，可用\*表示发给所有窗口。

win.postMessage(airport.value,'\*');

//alert("Sent!");

} **catch** (error) {

alert(error);

}

}

</script>

</head>

<body>

<table>

<tr><h1>国内机场航线网络分析示例</h1></tr>

<tr><h3>国内机场列表</h3></tr>

<tr>

<td>

<form action=*"#"*>

<label for=*"airport"*>机场</label>

<select name=*"airport"* id=*"airport"*>

<% **for** (Record record : result) { %>

<option value=*"*<%=record.get("name").asString()%>*"*><%=record.get("name").asString()%></option>

<% }%>

</select>

<input type=*"button"* value=*"传入参数"* onclick="setAirport()" />

传出选中的节点：<input id=*"nodesSelected"* type=*"text"* value=*""* size=*30*></input>

</form>

</td>

</tr>

</table>

<iframe id=*"shinyappframe"* src=*"https://124.223.110.20/shiny/users/jean/airports2/"* style="border: *1px solid #AAA*; width: *100%*; height: *600px*"></iframe>

</body>

</html>

三、应用层

应用层实现服务器端的业务逻辑。因为是演示性质，本示例的应用层只有一个页面index.jsp，并且非常简单，只是抽取了国内的机场列表。可以扩展一下获得传入参数之前在宿主业务系统中的前置业务过程（逻辑），以及获得传出结果之后的后续业务过程，比如规划航线上的航班票务，那就是完整的应用层业务逻辑。

四、中间层

中间层的主要任务是系统集成，提供大数据的分析展示能力。Shiny APP的主要长处是可以通过R语言与Python语言连接各种不同的后端系统，比如数据仓库、图数据库、GIS、NLP等，作各种深度的数据分析如机器学习、深度学习、回归分析、时间序列分析、文本挖掘等，从而拓展传统J2EE应用系统的功能，我的书中对此都作了探讨和尝试。本例中是连接数据层的Neo4j图数据库。

Shiny APP的结构，因为是WEB界面的APP，所以把表示层（UI）与实现服务器端逻辑的中间层分离了（本来是应用层，在本解决方案中，因为嵌入上一级J2EE的应用层中，被下推一级成为了中间层），表示层的UI在ui.R中实现，中间层在server.R中实现，Shiny的Reactive编程模式通过websocket等在二者之间建立了自动同步机制，监测UI端所有输入变量的变化，自动同步到服务器端，更新所有依赖于这些输入变量的输出变量，比如各种统计图、表格等，再动态生成并更新UI中的可视化内容。浏览器工作在文档对象模型DOM上，所有的网页内容都可以动态的生成和更新。这是Shiny APP的基本架构。下面具体看看本例的程序，通常还有一个可选的global.R来放APP启动时只执行一次的代码，存放ui.R与server.R中所有用户共享的全局数据。

1、global.R

装入reticulate、igraph等引用的包，通过reticulate连接Python，然后装入定义了业务功能函数的Python脚本，最后调用Python函数getAirports()连接Neo4j得到国内的机场列表。

library(reticulate)

library(sqldf)

library(igraph)

# visNetwork提供了浏览器端交互式动态展示网络的Javascript Widget，并且提供了Shiny支持。

# 它的网络图是平面的，比较适合于发票交易网络的可视化展示。

library(visNetwork)

# 确定连接Neo4j的Python脚本的目录位置

print(getwd())

location<- substr(getwd(),1,4)

# 调用Pyhton脚本连接 Neo4j, Neo4j的官方Python Driver比开源R Driver好用。

if (location != '/srv'){ # Rstudio 开发环境

part<- '~'

}else{ # 服务器部署环境

part<- '..'

}

path<- paste(part,"/scripts/","neo4jAirports.py",sep="")

print(path)

# 装入脚本

source\_python(path)

# 调用Python函数载入国内有出发航线数据的机场列表

airports<- getAirports("CN")

2、ui.R

用Shiny的各种UI组件定义了Shiny APP的UI，就是上面运行效果见到的样子，它们实际上是HTML Widget，绑定到浏览器上运行的各种javascript库，由R的htmlwidgets包支持。具体可参阅 [Shiny的API文档](https://shiny.rstudio.com/reference/shiny/1.6.0/)。

1）嵌入javascript处理与父窗口通过postMessage()机制交换数据。收到父窗口传入的机场名字参数时，调用Shiny的Shiny.setInputValue()函数通知服务器端输入变量的更新，再由reactive机制更新浏览器端的UI。

2）UI本身。Shiny APP的所有输入变量都在input列表下，所有输出变量都在output列表下，与server.R交叉引用这些输入输出变量，通过reactive()和observe()、observeEvent()机制监测输入的变化，然后由各种render()函数动态生成UI中的可视化输出（估计是该类输出相关的数据），再由ui.R中的对应的output()函数生成浏览器端的可视化UI代码（估计是DOM模型下的javascript与HTML、CSS）。sever.R中的render()函数与ui.R中的output()函数一一对应，后面再具体讲讲。注意最后面又嵌入了一段javascript，禁止用户自己修改输入变量“选中的节点”node\_selected，它在记录点击的节点时自动更新。

# 浏览器端UI函数，画机场航线网络图。

fluidPage(

# Javascript 处理父窗口传入的参数等。

tags$head(tags$script("

// 记录父窗口，初始为空。

parent = null;

// 处理接收到的消息。

window.addEventListener('message', function(e) {

//alert(e.data);

try{

//记录父窗口以备回发信息

parent = e.source;

//向服务器发送input变量更新消息

//Shiny.setInputValue('airport', e.data, {priority:'event'});

Shiny.setInputValue('airport', e.data);

} catch (error){

alert(error);

}

},false);

// 点击时传出参数

$(document).on('click', '.btn-success', function (evt) {

evt.preventDefault();

var selected = document.getElementById('nodes\_selected');

if (selected ==null){

alert('No node is selected!');

}else{

if (parent == null){

alert(selected.innerHTML);

} else {

try{

parent.postMessage(selected.innerHTML, '\*');

} catch(error){

alert(error);

}

}

}

});

")),

# Application title

titlePanel("机场航线示例"),

sidebarLayout(

# Sidebar with a selectioninput and a numericInput

sidebarPanel(

selectInput(

'airport',

'国内机场',

airports,

selected = 'Zhuhai Airport'

),

#numericInput("hops", "最多转机次数", value = 0, min = 0, max = 2),

sliderInput("hops",

"最多转机次数：",

min = 0, max = 2, value = 0),

checkboxInput("ifRecord","记录点击的节点", value = FALSE),

textInput("node\_selected","选中的节点",""),

actionButton("sendout", "传出选中节点", class = "btn-success"),

actionButton("clearNodes", "清除选中节点", class = "btn-cleear")

),

# Show the network

mainPanel(

# 画航线网络图

visNetworkOutput("plot"),

hr(),

tags$div(

"已选择的节点"

),

hr(),

textOutput("nodes\_selected"),

# 禁止自己修改 node\_selected textInput

tags$script(HTML(

'var node\_selected = document.getElementById("node\_selected");',

"node\_selected.disabled = true;"

))

)

)

)

3、server.R

1）监测输入变量的变化。

例如ui.R中定义了selectInput类型Widget的输入变量airport，在server.R中定义响应函数airport = reactive({...}）来监测它的变化。后面引用它的observe()、observeEvent()、renderVisNetwork()都会被它的变化触发。前面ui.R中嵌入的javascript在接收到父窗口传入的国内机场参数时，调用Shiny的Shiny.setInputValue('airport', e.data)函数，通知服务器端输入变量airport发生了变化，服务器端的 observeEvent(airport(),{...})事件响应函数被激活，调用updateText(session, "airport", value = airport())去更新UI中airport变量的值及可视化状态。每种类型的输入变量都有它对应的update()函数。

2）根据输入的变化更新输出

输入变量的变化也会触发引用它的render()函数更新输出。这里用output$plot = renderVisNetwork({...})来生成网络的可视化输出，存储在输出变量列表output的变量plot中。然后在ui.R中，调用与renderVisNetwork()配对的visNetworkOutput("plot")函数把网络可视化的呈现在UI中，注意它引用的是前面server.R中定义的输出变量plot。每种类型的输出变量都有它专用的render()函数和对应的output()函数。

具体在renderVisNetwork()中，graph = getAirLines(airport(),hops())引用输入变量的监测函数得到输入变量的最新值，调用Python函数getAirLines()得到从给定机场出发指定跳数内的航线图，它是一个[source,target,distance]列表的路径边集dataframe，然后用SQL语言操作dataframe graph，合并重复的边赋给graph2，再由igraph用graph2在内存中重建图，最后输出节点集与边集，用visNetwork()函数生成图的可视化输出赋给变量vg。visEdges()函数处理边的可视化效果，visEvents(selectNode ="...")嵌入了一个javascript函数处理UI界面上节点被点击的事件selectNode，检查“记录点击的节点”开关ifRecord是否打开，如果打开了就调用Shiny.setInputValue('node\_selected',...)函数通知服务器（注意：javascript运行在UI端的浏览器中），输入变量node\_selected的值发生了变化，触发observeEvent(node\_selected(),{...})函数更新UI，以及renderText({...}）函数更新session中的已选择节点列表，更新输出变量output$nodes\_selected及其UI的可视化状态。所以前面ui.R中嵌入javascript禁止用户自己改变输入变量node\_selected。

对于小的dataframe，用sqldf包以SQL语法去操作它，对于大的dataframe，比如几千万条记录，要用dplyr包以非SQL的语法去操作它，该包实现了SQL语言对应的功能。

# 服务端函数，根据浏览器选定的机场选出其给定步数内航线可达的机场。

function(input, output, session) {

# 定义reactive变量，监视输入的变化

airport<- reactive({

input$airport

})

hops<- reactive({

as.integer(input$hops)+1

})

node\_selected<-reactive({

input$node\_selected

})

# 监视输入变量变化事件，这是为嵌入宿主网页更新浏览器端输入变量准备的。

observeEvent(airport(),{

# 这一句执行了,证明触发了

print(paste("Airport event triggered: ",airport()))

# 注意这里用updateTextInput更新文本到浏览器端，如果用updateSelectInput()会报错，可能要求的是下标。

updateTextInput(session, "airport", value = airport())

})

observeEvent(hops(),{

# 这一句执行了,证明触发了

print(paste("Hops event triggered: ",hops()))

#updateNumericInput(session, "hops", value = hops()-1)

updateSliderInput(session, "hops", value = hops()-1)

})

observeEvent(node\_selected(),{

# 这一句执行了,证明触发了

print(paste("Node selected event triggered: ",node\_selected()))

updateTextInput(session, "node\_selected", value = node\_selected())

})

# 用Chrome devtools->elements展开当前的DOM对象，可以看到正确的 DOM id是 nodeSelectplot

# observeEvent(input$nodeSelectplot,{

# # 这一句执行了,证明触发了

# print(paste("Node selected event triggered: ",input$nodeSelectplot))

# })

# 清除选中节点

observeEvent(input$clearNodes,{

session$userData$nodes\_selected <- ""

updateTextInput(session, "node\_selected", value = "")

})

# 更新网络图

output$plot <- renderVisNetwork({

# 调Python函数得到指定机场指定跳数以内的通达航线图

graph<- getAirLines(airport(),hops())

# 合并多重边，存在多条路径都经过同一条边的情况，统计多重边的数量作为合并后边的权重。

graph2<-sqldf("select source,target,distance, count(\*) as paths

from graph group by source,target,distance

order by source,target")

# 建立 igraph图

g <- graph.data.frame(graph2, directed = TRUE)

# 节点标签

V(g)$label<-V(g)$name

# 边宽度

E(g)$width<-E(g)$paths

# 边标签

E(g)$label<-as.character(E(g)$distance)

# 输出转换成vsNetwork包要求的vertices & edges dataframe

verts<- as\_data\_frame(g, what="vertices")

# 必须有id列

names(verts)<-c("id", "label")

edges<- as\_data\_frame(g, what="edges")

# 生成网络可视化图

title <- paste(airport(),"转机 ",hops()-1," 次以内航线图",sep="")

vg<-visNetwork(nodes=verts, edges=edges, main=title) %>%

visEdges(arrows = 'to') %>%

visEvents(selectNode =

"function(properties) {

var ifRecord = document.getElementById('ifRecord');

//alert(ifRecord.checked);

if(ifRecord.checked){

var data = this.body.data.nodes.get(properties.nodes[0]).id;

//alert('selected nodes ' + data);

Shiny.setInputValue('node\_selected', data, {priority:'event'});

}

}"

)

# nodesIdSelection选项还没有找到正确的 input DOM id,文档所说的 $plot\_selected不对。

# 参阅： https://datastorm-open.github.io/visNetwork/shiny.html

# 用上面的visEvents(selectNode = function())代替。

# 参阅： https://datastorm-open.github.io/visNetwork/more.html

# 用Chrome devtools->elements展开当前的DOM对象，可以看到正确的 DOM id是 noeSelectplot，

# 其中plot是上面输出的本网络图的id output$plot，

# 这里还是用visEvents()来处理，先检查记录点击节点的开关是否打开再发送点击事件通知服务器端。

# visOptions(vg, nodesIdSelection = TRUE, manipulation = TRUE)

visOptions(vg, manipulation = TRUE)

})

# 显示选中的节点

output$nodes\_selected <- renderText({

session$userData$nodes\_selected<-paste(session$userData$nodes\_selected,node\_selected(),sep="!")

})

}

这就是Shiny的Reactive编程模式工作的过程，当Shiny APP工作在DESCRIPTION文件（可选）定义的showcase模式时，右边会打开代码窗口，输入变量变化所激活的代码会用黄色高亮显示，可以直观的看到它工作的过程。具体开发可以参阅 [Shiny的教程](https://shiny.rstudio.com/tutorial/)， [Shiny API文档](https://shiny.rstudio.com/reference/shiny/1.6.0/)，以及Rstudio首席科学家写的教材 [《Mastering Shiny》](https://mastering-shiny.org/)。 [visNetwork的使用](https://www.statworx.com/en/content-hub/blog/interactive-network-visualization-with-r/)具体可以参阅 [它的文档](https://datastorm-open.github.io/visNetwork/)。[igraph](https://igraph.org/r/#docs)的API可以看[它的文档](https://igraph.org/r/html/latest/)，这个是[R igraph的入门教程](https://igraph.org/r/html/latest/aaa-igraph-package.html)，这个是[Python版](https://igraph.org/python/tutorial/latest/)的。

4、Python函数。在global.R装入的脚本neo4jAirports.py中定义。

1）Neo4j Python Driver。连接协议bolt+ssc是自签证书SSL连接专用，不会验证证书的有效性，但会建立加密连接。用户名和密码另加一个模块neo4jkeys.py定义。

2）函数

函数getAirLines(source, length)调用path\_to\_airports(tx, source,length)，执行了一个40行的Cypher大查询，返回从节点source出发，length跳以内，所有通达节点的最短路径。getRings(source, length, amplitude, threshold)调用airports\_ring(tx, source, length, amplitude, threshold)返回给定机场的稳定强环路，它将用于后面的3D可视化演示。

函数getAirports(country)调用countryAirports(tx, country)执行一个Cypher查询，返回指定国家内有出发航线的所有机场。该数据集不完整，有些机场没有出发航线，会引起Shiny APP报错，要排除。

Cypher语句中可以通过$varname占位符定义参数，然后在调用时用tx.run(CypherSQL, varname=value)替换成具体的值，比较方便。

import pandas as pd

from neo4j import GraphDatabase, basic\_auth

from neo4j import neo4jkeys

# bolt： 无SSL

# bolt+s :SSL并检查证书链

# bolt+ssc: SSL但不检查证书链，自签证书专用

# 同样适用于neo4j协议, neo4j+s，neo4j+ssc

# https://neo4j.com/docs/python-manual/current/client-applications/

# 只返回最短路径

def getAirLines(source, length):

driver = GraphDatabase.driver("bolt+ssc://localhost:7687", auth=basic\_auth(neo4jkeys.user, neo4jkeys.password))

with driver.session(database="neo4j") as session:

graph = session.read\_transaction(path\_to\_airports, source,length)

driver.close()

return graph

def path\_to\_airports(tx, source,length):

cypher = '''

CALL {

MATCH (start:Airport{descr:$name})

CALL gds.alpha.shortestPath.deltaStepping.stream(

{nodeQuery: 'MATCH (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

RETURN id(n) AS id',

relationshipQuery: 'MATCH (startNode:Airport)-[r:HAS\_ROUTE]->(targetNode:Airport),

(startNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}),

(targetNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}) RETURN id(startNode) AS source, id(targetNode) AS target, r.distance AS distance',

startNode:start,

relationshipWeightProperty:'distance',

delta: 3.0 })

YIELD nodeId, distance

RETURN nodeId, distance

ORDER BY distance

//跳过第一条，是自己

SKIP 1

//LIMIT 10;

}

WITH nodeId

MATCH (start:Airport{descr:$name}),(target:Airport) WHERE id(target)=nodeId

CALL {

WITH start, target

CALL gds.shortestPath.dijkstra.stream(

{nodeQuery: 'MATCH (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

RETURN id(n) AS id',

relationshipQuery: 'MATCH (startNode:Airport)-[r:HAS\_ROUTE]->(targetNode:Airport),

(startNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}),

(targetNode:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"}) RETURN id(startNode) AS source, id(targetNode) AS target, r.distance AS distance',

sourceNode:id(start),

targetNode:id(target),

relationshipWeightProperty:'distance' })

YIELD path

WHERE length(path)<=$len

RETURN nodes(path) AS path

}

//RETURN path;

WITH path

UNWIND range(0, size(path)-1) AS index

WITH path[index] AS current, path[index+1] AS next

MATCH (current)-[r:HAS\_ROUTE]->(next)

RETURN current.descr AS source, next.descr as target, r.distance AS distance;

'''

start = []; target=[]; distance=[]

for record in tx.run(cypher, name=source, len=length):

start.append(record["source"])

target.append(record["target"])

distance.append(record["distance"])

# 从 list建立 pandas data frame

data = pd.DataFrame({"source":start,"target":target,"distance":distance})

return data

def getRings(source, length, amplitude, threshold):

driver = GraphDatabase.driver("bolt+ssc://localhost:7687", auth=basic\_auth(neo4jkeys.user, neo4jkeys.password))

with driver.session(database="neo4j") as session:

graph = session.read\_transaction(airports\_ring, source, length, amplitude, threshold)

driver.close()

return graph

def airports\_ring(tx, source, length, amplitude, threshold):

cypher = (\

'''

MATCH (china:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

WITH COLLECT(china.id) AS targets

MATCH path = (source:Airport{descr:$name})-[r:HAS\_ROUTE\*2..'''+str(int(length))+"]->(target)"+\

'''

WHERE target.descr = source.descr

AND ALL( node IN nodes(path) where node.id IN targets)

WITH path, [r in relationships(path)|r.distance] AS distances, $amplitude AS amplitude, $threshold AS threshold

WITH path, apoc.coll.avg(distances) AS avgDist, apoc.coll.min(distances) as minDist, apoc.coll.max(distances) as maxDist

WHERE minDist/avgDist >= (1-amplitude) AND maxDist/avgDist <= (1+amplitude) AND avgDist >= threshold

WITH path, relationships(path) as flights

UNWIND flights AS flight

WITH DISTINCT flight

MATCH (n:Airport)-[flight]->(m:Airport)

RETURN n.descr AS source, m.descr as target, flight.distance AS distance;

'''

)

# print(cypher)

start = []; target=[]; distance=[]

for record in tx.run(cypher, name=source, amplitude=amplitude, threshold=threshold ):

start.append(record["source"])

target.append(record["target"])

distance.append(record["distance"])

# 从 list建立 pandas data frame

data = pd.DataFrame({"source":start,"target":target,"distance":distance})

return data

# 返回国内机场列表

def getAirports(country):

driver = GraphDatabase.driver("bolt+ssc://localhost:7687", auth=basic\_auth(neo4jkeys.user, neo4jkeys.password))

with driver.session(database="neo4j") as session:

graph = session.read\_transaction(countryAirports, country)

driver.close()

return graph

def countryAirports(tx, country):

cypher="match (n:Airport)-[:IN\_COUNTRY]->(c:Country{code:$code}), \

(n:Airport)-[:HAS\_ROUTE]->(target) return distinct n.descr as name"

airports = []

for record in tx.run(cypher, code=country):

airports.append(record["name"])

# 从 list建立 pandas data frame

data = pd.DataFrame({"airport":airports})

return data

五、数据层

由Neo4j图数据库通过Cypher查询语言返回图数据，为了方便R语言处理，这里返回边的[source,target, distance]列表组成的dataframe，igraph可以根据dataframe在内存中重建图的实例，另一个查询返回国内机场列表，都是文本列表数据。

1、Cypher

1）查询给定机场出发航线通达机场的Cypher查询比较大，有40行，具体见上面的Python程序，它调用了GDS库中的两个图算法。单源最短路径

gds.alpha.shortestPath.deltaStepping.stream算法找到从source节点出发能通达的所有节点的最短距离，因为有向图中从一个节点到另一个节点可能有多条路径。最短路径算法

gds.shortestPath.dijkstra.stream找到从source节点到上面通达的每个target节点的最短路径。注意GDS不同版本中图算法的名称和参数可能会略有不同，具体要看文档。

然后舍去超过长度length的路径，展开每条路径，并把每条边展开成[source,target, distance]构成的列表，所以会有多条路径都经过的重复边。

2）查询国内有出发航班的Cypher比较简单，要求有一条发出的HAS\_ROUTE边即可，同样需要distinct排除重复，见上面的Python程序。

3）加载本例图数据。具体可看 [GDS教学实例Github主页](https://github.com/neo4j-graph-examples/graph-data-science2)上的 [文档](https://github.com/neo4j-graph-examples/graph-data-science2/blob/main/documentation/gds_browser_guide2.adoc)。

2、GDS库

1） [GDS库的Github主页](https://github.com/neo4j/graph-data-science)， [各种算法的文档](https://neo4j.com/docs/graph-data-science/1.8/alpha-algorithms/single-source-shortest-path/)及各版本GDS库插件的下载，插件的版本要适配服务器的版本，版本适配列表见上面的GDS库Github主页，现在最新发布的是2.0版。

2）开发有向图最小树形图朱刘算法java版。交易网络及产业链、供应链分析等有向图的分析比较需要这个算法，我根据网上找到的朱刘算法Python版在GDS1.6的基础上开发了它的java版Neo4j插件，包括常用的K最小树形图等，具体见第二章《发票交易网络分析》的第七节《Neo4j自定义过程之有向树形图》。GDS是个非常强大的图算法库，这样做的好处是无缝嵌入GDS库中，可以与GDS库的其它算法一起组合使用，就像前面查询机场通达最短路径的Cypher语句一样。

六、访问授权

有经验的读者读到这里大概会问这个问题了，做好了Shiny APP，怎样管理它的访问授权？开源的Shiny社区版没有访问授权管理的功能，收费的商业版Shiny Pro才有。可以这样来解决问题：

1）Shiny Server限制来访的IP，这个可以配置，只有应用层的J2EE服务器及开发团队的IP可以访问。

2）通过前面IFRAME一页嵌入的方式，可以把访问权限管理交给宿主业务系统去实现，只要管理宿主网页URL的访问授权即可，J2EE里有很多成熟的架构可用。

七、3D可视化网络分析

前面的可视化展示是2D的，3D可视化可以旋转与透视，提供了更好的探索效果。另外还可以增加第4维时间维，做成动画，这就是4D了。还可以通过物体的颜色、大小、形状、纹理等增加更多的数据表现维度（2D也可以）。

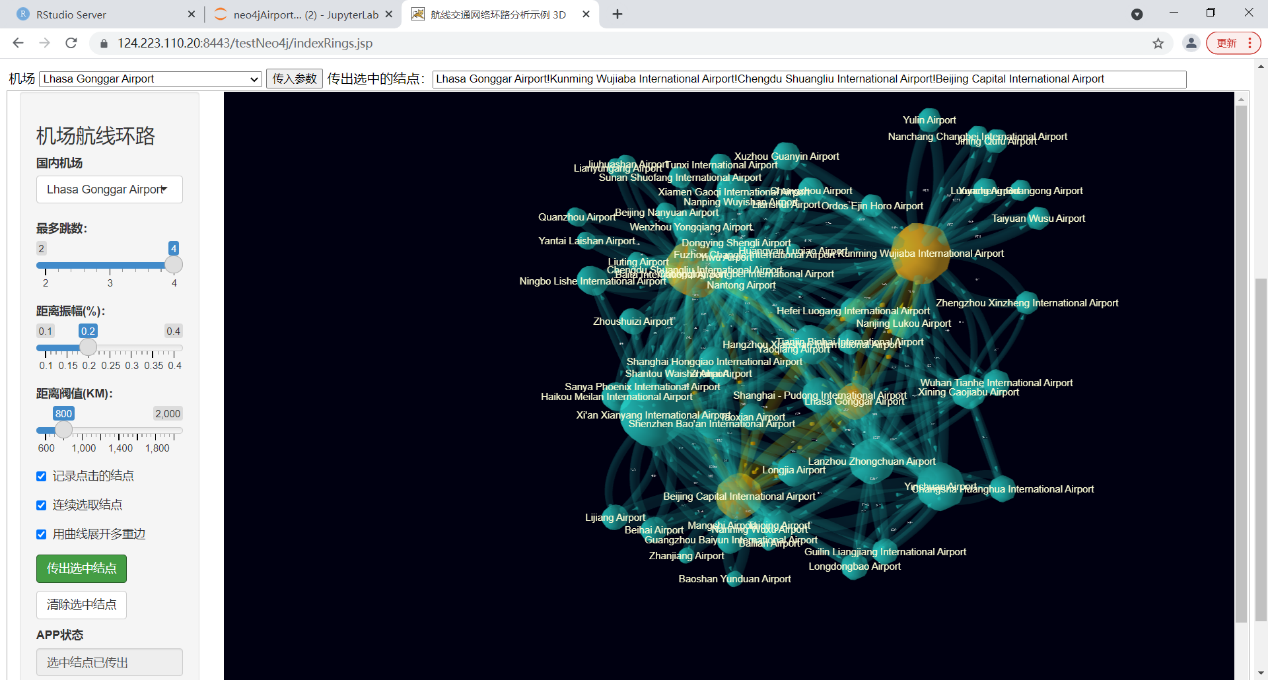
这里使用浏览器上比较流行的[ThreeJS](https://threejs.org/) 开源javascript 3D渲染（render）引擎，以及在它上面开发的有向图3D可视化开源项目 [3D Force-Directed Graph](https://github.com/vasturiano/3d-force-graph)，它的开发者是葡萄牙人[Vasco Asturiano](https://github.com/vasturiano)，Neo4j也在使用，更多的浏览器端开源可视化项目可以参考[这篇文章](https://awesomeopensource.com/projects/graph-visualization)。

[3D Force-Directed Graph](https://github.com/vasturiano/3d-force-graph)是纯粹的浏览器端JS库，没有直接支持Shiny Reactive编程的render()函数和配对的output()函数，我通过Shiny的消息通信机制，由服务器在reactive input的响应函数中，直接发送数据消息回浏览器，然后由浏览器中注册的消息处理函数载入数据，用[3D Force-Directed Graph](https://github.com/vasturiano/3d-force-graph)的API生成浏览器端图的实例，解决了问题。Shiny的javascript通信机制可以参考[这篇文章](https://shiny.rstudio.com/articles/communicating-with-js.html)。具体的API可以看[3D Force-Directed Graph的Github主页](https://github.com/vasturiano/3d-force-graph)。

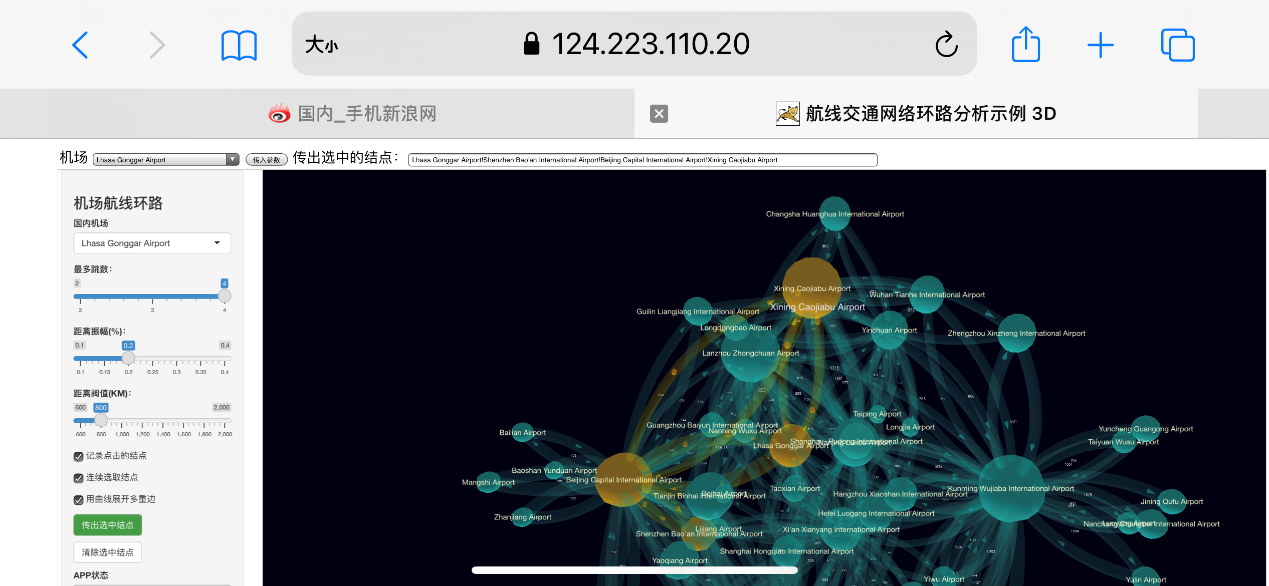
所以与前面的2D可视化例子相比，主要的改动是Shiny APP的3个程序文件。下面具体看看，先看看运行的效果，手机上运行也比较流畅，效果非常好。搭载的网络分析，换成更有落地应用意义的机场航线稳定强环路分析。稳定强环路的定义是经过源结点的环路，其中边权重的均值不小于阀值threshold，权重上下浮动的振幅小于amplitude。虚开发票其中一种典型的网络模型就是资金形成环路，转一圈后回到虚开的纳税人那里。设定阀值参数是要查看一定规模以上的资金流，设定振幅参数是排除无关交易连接形成的环路,比如在大额的主要业务交易中，有一笔正常的其它小额交易，它所形成的环路就应该排除。

1、运行效果

1)PC上运行效果



2）手机上运行效果



2、global.R

主要的变化是使用jsonlite库，后面将Cypher查询的结果（图）以json格式的文本返回给浏览器，由javascript处理。

library(reticulate)

library(sqldf)

library(igraph)

library(jsonlite)

# 确定连接Neo4j的Python脚本的目录位置

print(getwd())

location<- substr(getwd(),1,4)

# 调用Pyhton脚本连接 Neo4j, Neo4j的官方Python Driver比开源R Driver好用。

if (location != '/srv'){ # Rstudio 开发环境

part<- '~'

}else{ # 服务器部署环境

part<- '..'

}

path<- paste(part,"/scripts/","neo4jAirports.py",sep="")

print(path)

# 装入脚本

source\_python(path)

# 调用Python函数载入国内有出发航线数据的机场列表

airports<- getAirports("CN")

3、server.R

server.R中通过observ()函数监测airport、hops、amplitude、threshold四个输入变量的变化，调用Python函数查询给定机场指定跳数内的稳定强环路，这里是Shiny的Reactive机制，然后在该函数的最后，把查询返回的图数据，转换为 3D Force-Directed Graph API要求的json数据格式，调用session$sendCustomMessage("updateGraphData", jsonGraph)发送给浏览器，这里updateGraphData是自定义的消息类型，jsonGraph就是json格式的图数据，调用session$sendCustomMessage("setSourceNode", airport())通知浏览器设置相机的初始视野。后面清除选中结点也是通过这个机制，发送一个消息给浏览器执行：session$sendCustomMessage("clearSelectedNodes", airport()) 。

# 服务端函数，根据浏览器选定的机场选出其给定步数内航线可达的机场。

function(input, output, session) {

# 定义reactive变量，监视输入的变化

# 选定的机场

airport<- reactive({

input$airport

})

# 指定的跳数

hops<- reactive({

as.integer(input$hops)

})

# 指定的振幅内

amplitude<- reactive({

as.numeric(input$amplitude)

})

# 指定的阀值内

threshold<- reactive({

as.integer(input$threshold)

})

# 选中的结点

nodes\_selected<-reactive({

input$nodes\_selected

})

# 监视输入变量airport与hops的变化事件。

observe({

# 这一句执行了,证明触发了

print(paste("Observe triggered: ",paste(airport(),hops())))

t1<-proc.time()

# 调Python函数得到指定机场指定跳数以内的航线环路图

graph<- getRings(airport(),hops(), amplitude(), threshold())

t2<-proc.time()

timeUsed <- t2-t1

# 合并多重边，存在多条路径都经过同一条边的情况，统计多重边的数量作为合并后边的权重。

graph2<-sqldf("select source,target,distance, count(\*) as paths

from graph group by source,target,distance

order by source,target")

# 建立 igraph图

g <- graph.data.frame(graph2, directed = TRUE)

# 节点标签

V(g)$label<-V(g)$name

# 节点的航班数，最短路径子网统计出度和入度，因为末端的结点只有入度。这是为了演示设置结点的大小。

# 对于完全子网，航线比较多，只统计入度

V(g)$size <- degree(g, mode="out")

# 最短路径子网边的宽度

E(g)$width<-E(g)$distance/200

# 边标签

E(g)$label<-as.character(E(g)$distance)

# 测试通过颜色属性设置结点的颜色，这样可以通过igraph先设定结点与边的颜色，再传给3D Forced Directed Graph

# 这4句只是测试用途，可以不用管

V(g)$color<-"#00BFFF"

V(g)$color[1]<-"red"

E(g)$color<-"green"

E(g)$color[1]<-"red"

# 转换为 3D Force-Directed Graph 要求的json网络格式。

# 结点列表

verts<- as\_data\_frame(g, what="vertices")

# 必须有id列

names(verts)<-c("id", "label", "size","color")

# 边列表

edges<- as\_data\_frame(g, what="edges")

names(edges)<-c("source", "target","distance","paths","width","label","color")

jsonVerts<- toJSON(verts)

jsonEdges<- toJSON(edges)

# 合并结点与边的json数据集

jsonGraph<- paste('{

"nodes": ',jsonVerts,',

"links": ',jsonEdges,'}')

# 发送给浏览器，更新网络图

session$sendCustomMessage("updateGraphData", jsonGraph)

# 更新浏览器端selectInput的状态

# 注意这里用updateTextInput更新文本到浏览器端，如果用updateSelectInput()会报错，可能要求的是下标。

updateTextInput(session, "airport", value = airport())

# 设置初始视野，通知浏览器端聚焦源结点并着色，以便分析。

session$sendCustomMessage("setSourceNode", airport())

# 这个过程需要点时间，告知客户端更新APP状态。

updateTextInput(session, "appstatus", value = paste("网络数据已加载: ",round(timeUsed[3],1),"秒",sep=""))

})

# 浏览器端结点的javascript中.onNodeClick((node, event)事件处理，

# 发送消息函数Shiny.setInputValue('nodes\_selected', nodeList)会触发这个事件处理函数。

observeEvent(nodes\_selected(),{

# 这一句执行了,证明触发了

print(paste("Nodes selected event triggered: ",nodes\_selected()))

# 在session中记录选中的结点，记录APP的状态

session$userData$nodes\_selected<- nodes\_selected()

# 更新浏览器端显示的输入变量nodes\_selected

updateTextAreaInput(session, "nodes\_selected", value = nodes\_selected())

})

# 清除选中节点，浏览器端点击'清除选中结点'按钮会触发这个事件处理函数。

observeEvent(input$clearNodes,{

print("Clear nodes selected event triggered.")

# 更新session中记录的用户数据

session$userData$nodes\_selected <- ""

# 更新浏览器端显示的输入变量nodes\_selected

updateTextAreaInput(session, "nodes\_selected", value = "")

# 发送一个清理指令给浏览器端注册的javascript函数，重置图中选中结点的颜色并更新APP的状态。

session$sendCustomMessage("clearSelectedNodes", airport())

})

}

4、ui.R

ui.R的改动就比较大了，因为浏览器端要用3D Force-Directed Graph 的API操作各种可视化效果，具体API请看项目的主页，上面还有各种例子学习，可以直接看到它的实现代码和效果。

程序已经注释的很清楚了，直接看程序吧。要用Shiny.addCustomMessageHandler('updateGraphData', function(data) {}注册一个javascript函数处理服务器发来的json格式图数据，Shiny.addCustomMessageHandler('setSourceNode', function(data) {}对源结点着色并设置相机的初始视野，以便开始可视化探索，以及Shiny.addCustomMessageHandler('clearSelectedNodes', function(data) {}注册一个函数处理清除选中结点的指令，消息类型是自定义的。其它都是javascript在浏览器DOM模型下的常规编程。与前面一样，javascript都是嵌入到ui.R的R语言程序中，不过请记住ui.R运行在服务器端而嵌入的javascript随着由ui.R动态生成的UI HTML发往浏览器端，在浏览器上运行。因为3D Force-Directed Graph操作图可视化是纯粹的javascript客户端，所以嵌入的javascript比较多，成为了ui.R的主体。

这里要讲讲装入本例3D Force-Directed Graph及其依赖的ThreeJS库的方式，直接用Shiny的tags$script(src = "3d-force-graph.js")从www子目录装入会报错：“ Cannot use import statement outside a module”，用includeScript(path = "./www/3d-force-graph.js")从本地的www子目录装入是可以的，不过会和ui.R生成的html页面，包括里面嵌入的javascript合并在一起，非常庞大，用Chrome的devtools调试javascript脚本时就比较不方便了，3d-force-graph.js、three.js、CSS2DRenderer.js、three-spritetext.js这4个JS文件加起来有67000多行源码。所以本例中还是用URI的方式从网上装入：tags$script(src="//unpkg.com/3d-force-graph")。如果Shiny Server部署在内网，就只能从本地www子目录include了。

另外，服务器查询图要执行比较复杂的路径算法，需要点时间，响应函数更新浏览器端应用状态的消息会被阻塞至所有操作完成，所以在浏览器端监测Shiny的shiny:inputchanged事件，先在浏览器端更新APP的状态，以免用户不知道APP是否在正确的运行，这通过注册一个shiny:inputchanged事件处理函数来完成，Shiny 在浏览器端会触发的事件列表，可以参考[这篇文章](https://shiny.rstudio.com/articles/js-events.html)：

$(document).on('shiny:inputchanged', function(event) {…} 。

为了在浏览器上用javascript对图进行操作，为3D Force-Directed Graph的javascript图数据结构补充了邻接表与链接表的结构，在高亮显示选中结点的子网时就用它们来查找要高亮显示的边，也可以用它们来执行javascript端的其它图算法，比如遍历算法等。

参考3D Force-Directed Graph主页上的例子，对结点与边的着色不是通过设置它们的颜色属性进行，而是通过设置selectedNodes与highlightLinks这两个集合，然后由nodeColor()与linkColor()重新渲染，在渲染中根据是否在集合中分别赋予不同的颜色。3D效果是渲染图，每次改变都要重新渲染。也可以在函数中指定颜色属性的名称来指定结点与边的颜色，比如nodeColor(‘color’)与linkColor(‘color’)，结点与边都可在服务器端由igraph预先设置好颜色属性’color’的值，值改变后用这两个函数重新渲染即可，具体可参阅[这个链接](https://github.com/vasturiano/3d-force-graph/issues/520)及[项目主页的文档](https://github.com/vasturiano/3d-force-graph)。

边上质点流动的动画效果由linkDirectionalParticles()与emitParticle（）直接实现，运动质点的数量、速度、大小、颜色等都可以根据图中边的属性来确定，这样它就可以反映图上承载对象在时间维度上的流动状况，可视化探索就扩展为4D。就一般需求而言，结点的大小、形状、颜色与标签，边的宽度、方向、颜色与标签解决了，可视化就比较满意了，4D动画是锦上添花。

展开/折叠多重边通过调用mygraph.linkCurvature(0.25).linkVisibility(false).linkVisibility(true)完成，设置曲率为0是折叠，不为0是展开，通过设置边的可见性强制重新渲染视图。

# 浏览器端UI函数，画机场航线网络图。

fluidPage(

# Javascript 处理父窗口传入的参数等。

tags$head(

tags$script(HTML("

// 记录父窗口，初始为空。

parent = null;

// 处理接收到的消息。

window.addEventListener('message', function(e) {

//alert(e.data);

try{

//记录父窗口以备回发信息

parent = e.source;

//向服务器发送input变量更新消息

//Shiny.setInputValue('airport', e.data, {priority:'event'});

var appstatus = document.getElementById('appstatus');

// 更新APP状态，因为耗时较长

appstatus.value = '正在查询网络数据...';

Shiny.setInputValue('airport', e.data);

} catch (error){

alert(error);

}

},false);

// 点击时传出参数

$(document).on('click', '.btn-success', function (evt) {

evt.preventDefault();

var selected = document.getElementById('nodes\_selected');

if (selected ==null){

alert('No node is selected!');

}else{

if (parent == null){

alert(selected.value);

} else {

try{

parent.postMessage(selected.value, '\*');

} catch(error){

alert(error);

}

}

}

});

// 触发了shiny:inputchanged事件，从浏览器端更新状态提示，因为服务器端要执行完了才返回。

// 在触发服务器reactive()与observe之前先更新Browser端状态，因为耗时较长。

// 清除选择结点更新状态在Shiny.addCustomMessageHandler()执行完了再设置，不在这里更新。

$(document).on('shiny:inputchanged', function(event) {

var appstatus = document.getElementById('appstatus');

if (event.name === 'airport') {

appstatus.value = '正在查询网络数据...';

}

if (event.name === 'hops') {

appstatus.value = '正在查询网络数据...';

}

if (event.name === 'amplitude') {

appstatus.value = '正在查询网络数据...';

}

if (event.name === 'threshold') {

appstatus.value = '正在查询网络数据...';

}

if (event.name === 'sendout') {

appstatus.value = '选中结点已传出';

}

if (event.name === 'ifMultiEdge') {

// 不向服务器端传送信息，边曲线是纯粹的可视化效果

event.preventDefault();

try{

var ifMultiEdge = document.getElementById('ifMultiEdge');

// .linkVisibility(false).linkVisibility(true) 确保刷新视图

if (ifMultiEdge.checked){

mygraph.linkCurvature(0.25)

.linkVisibility(false)

.linkVisibility(true);

appstatus.value = '已展开重叠边的直线';

}else{

mygraph.linkCurvature(0)

.linkVisibility(false)

.linkVisibility(true);

appstatus.value = '已叠加多重边为直线';

}

}catch(error){

alert(error);

}

}

});

"))

),

# Scripts for Vasturiano 3D Force-Directed Graph

# load 3d-force-graph.js, three.js, CSS2DRenderer.js, three-spritetext.js

# tags$script(src = "3d-force-graph.js"),

# Will cause an error: Cannot use import statement outside a module

# It's O.K. this way, but need to connect to unpkg.com

tags$script(src="//unpkg.com/3d-force-graph"),

tags$script(src="//unpkg.com/three"),

tags$script(src="//unpkg.com/three/examples/js/renderers/CSS2DRenderer.js"),

tags$script(src="//unpkg.com/three-spritetext"),

# Or include the scripts in the local www directory, there're over 67K lines of the javascript sources

# Will bring problem to debugging with Chrome's devtools, too long for the source.

#includeScript(path = "./www/3d-force-graph.js"),

#includeScript(path = "./www/three.js"),

#includeScript(path = "./www/CSS2DRenderer.js"),

#includeScript(path = "./www/three-spritetext.js"),

sidebarLayout(

# Sidebar with a selectioninput and a numericInput

sidebarPanel(

# Application title

tags$h3("机场航线环路"),

selectInput(

'airport',

'国内机场',

airports,

selected = 'Zhuhai Airport'

),

sliderInput("hops",

"最多跳数：",

min = 2, max = 5, value = 3, step = 1),

sliderInput("amplitude",

"距离振幅(%)：",

min = 0.1, max = 0.4, value = 0.2, step = 0.05),

sliderInput("threshold",

"距离阀值(KM)：",

min = 600, max = 2000, value = 800, step = 100),

checkboxInput("ifRecord","记录点击的结点", value = FALSE),

# 手机没有键盘，这个开关是为了方便手机使用

checkboxInput("ifMultiSelect","连续选取结点", value = TRUE),

checkboxInput("ifMultiEdge","用曲线展开多重边", value = FALSE),

actionButton("sendout", "传出选中结点", class = "btn-success"),

tags$h6(" "),

actionButton("clearNodes", "清除选中结点", class = "btn-cleear"),

tags$h6(" "),

textInput("appstatus","APP状态", value ="正在查询网络数据..."),

textAreaInput("nodes\_selected","选中的结点", rows =6, resize="vertical", value =""),

# 插入javascript，禁止自己修改 nodes\_selected textAreaInput, appstatus textInput

tags$script(HTML("

var nodes\_selected = document.getElementById('nodes\_selected');

nodes\_selected.disabled = true;

var appstatus = document.getElementById('appstatus');

appstatus.disabled = true;

")),

# 1/6

width = 2

),

# Show the network

mainPanel(

# 画航线网络图

# 定义图显示的区域, 在该区域中加载图

tags$div(id="3d-graph"),

# Style sheet of node label

tags$head(

tags$style("

.node-label {

font-size: 12px;

padding: 1px 4px;

border-radius: 4px;

color: LightGoldenRodYellow;

background-color: none; // rgba(0,0,0,0.5)

user-select: none;

}

")),

# 注册一个javascript消息处理函数，接收服务器发来的json图数据，更新图

# 用HTML(), 否则javascript lambda表达式的=>箭头会被替换成&转义符，导致javascript执行出错。

# 消息类型是updateGraphData，格式是json。

tags$script(HTML("

//定义图的全局变量

//保存对Graph对象的全局引用，取消结点选择时要处理图的结点与边

let mygraph = null;

//选中的结点集

let selectedNodes = new Set();

//选中的结点集之间的边

let highlightLinks = new Set();

//选中结点与未选中结点的颜色

// 参见下面链接中的颜色表。

// https://www.w3schools.com/tags/ref\_colornames.asp

// 最好写成#十六进制，有些按名称引用的可能不认识。边的颜色会被稀释淡化，所以不少颜色之间可能看不出区别。

let colorNodeSelected = 'GoldenRod';

let colorNode = 'LightSeaGreen';

// 选中边与未选中边的颜色

let colorEdgeSelected = 'gold';

let colorEdge = '#20B2AA'; // #20B2AA, LightSeaGreen， rgb(32, 178, 170)。

// 更新加载状态的引用

let APPStatus = document.getElementById('appstatus');

//接收服务器传来的json图数据，转换为图对象

Shiny.addCustomMessageHandler('updateGraphData', function(data) {

// 初始化为一个空图

gData = {

nodes: [],

links: []

};

//装入图数据json对象

try {

gData = JSON.parse(data);

} catch(error){

alert(error);

return;

}

// 为Vasturiano 3D Force-Directed Graph的javascript表示补充邻接表结构，以便javascript中执行图算法。

// 初始化邻接表，为结点增加neighbors列表与links列表

gData.nodes.forEach(node => {

node.neighbors=[];

node.links=[];

});

// 邻接是双向的，这里初始化为无向图邻接，也可以按需要初始化为有向图邻接。

gData.links.forEach(link => {

// 按属性值查找出源与目标结点

var sourceNode = gData.nodes.find(obj => {return obj.id === link.source});

var targetNode = gData.nodes.find(obj => {return obj.id === link.target});

sourceNode.neighbors.push(link.target);

targetNode.neighbors.push(link.source);

sourceNode.links.push(link);

targetNode.links.push(link);

});

APPStatus.value = '正在加载网络数据...';

//加载图，可视化显示

const Graph = ForceGraph3D({

// 用于为结点加上标签

extraRenderers: [new THREE.CSS2DRenderer()]

})

// 获得显示的区域，上面tags$div(id='3d-graph')定义的对象

(document.getElementById('3d-graph'))

// 加载数据

.graphData(gData )

// 加载测试数据

//.jsonUrl('https://raw.githubusercontent.com/vasturiano/3d-force-graph/master/example/datasets/miserables.json')

// 定义结点标签，鼠标移动到上面时会显示

.nodeLabel('id')

// 设定结点的大小

.nodeVal(node => node.size)

// 按组显示颜色

//.nodeAutoColorBy('group')

// 选中的结点以不同的颜色标记

.nodeColor(node => selectedNodes.has(node) ? colorNodeSelected : colorNode)

// 也可以通过指定结点的颜色属性名称来指定结点的颜色，这样可以先在igraph中处理

//.nodeColor('color')

// 为结点加上标签，使用结点扩展

.nodeThreeObject(node => {

const nodeEl = document.createElement('div');

nodeEl.textContent = node.id;

//nodeEl.style.color = node.color;

nodeEl.className = 'node-label';

return new THREE.CSS2DObject(nodeEl);

})

.nodeThreeObjectExtend(true)

// 为边加上标签，使用边扩展

.linkThreeObjectExtend(true)

.linkThreeObject(link => {

// extend link with text sprite

const sprite = new SpriteText(`${link.distance}`);

sprite.color = 'lightgrey';

sprite.textHeight = 1.5;

return sprite;

})

.linkPositionUpdate((sprite, { start, end }) => {

const middlePos = Object.assign(...['x', 'y', 'z'].map(c => ({

[c]: start[c] + (end[c] - start[c]) / 2 // calc middle point

})));

// Position sprite

Object.assign(sprite.position, middlePos);

})

// 为边加上箭头

.linkDirectionalArrowLength(3.5)

.linkDirectionalArrowRelPos(1)

// 设定边的宽度

.linkWidth(link => link.width)

// 设置边的曲率，不设置就是直线，设置曲率对于多重边的效果比较好一点，不设置的话多重边会合并成双向直线。

//.linkCurvature(0.25)

// 处理结点的点击事件，选中的结点以不同的颜色标记，并高亮显示选中结点之间的边。

.onNodeClick((node, event) => {

// 查看'记录点击的结点'开关及'连续选取结点开关'

var ifRecord = document.getElementById('ifRecord');

var ifMultiSelect = document.getElementById('ifMultiSelect');

//alert(ifRecord.checked);

if(ifRecord.checked){

// multi-selection

if (event.ctrlKey || event.shiftKey || event.altKey || ifMultiSelect.checked) {

if (selectedNodes.has(node)){

// 先删除结点再处理边

selectedNodes.delete(node);

if(node.links.length>0){

node.links.forEach(link => {

if (selectedNodes.has(link.source) || selectedNodes.has(link.target)){

//alert('delete highlight edge!');

highlightLinks.delete(link);

}

});

}

}

else{

// 在添加结点前先处理边

if(node.links.length>0){

node.links.forEach(link => {

if (selectedNodes.has(link.source) || selectedNodes.has(link.target)){

//alert('add highlight edge!');

highlightLinks.add(link);

}

});

}

// 添加结点

selectedNodes.add(node);

}

} else { // single-selection

const untoggle = selectedNodes.has(node) && selectedNodes.size === 1;

// 清空选中的结点集及高亮显示的边集

highlightLinks.clear();

selectedNodes.clear();

// 如果原来没有结点就添加该结点

if ( !untoggle){

selectedNodes.add(node);

}

}

// update color of selected nodes and Edges

Graph.nodeColor(node => selectedNodes.has(node) ? colorNodeSelected : colorNode)

.linkColor(link => highlightLinks.has(link)? colorEdgeSelected : colorEdge);

// 通知服务器选中的结点有变化

var nodeList = '';

selectedNodes.forEach(node => {

if (nodeList.length>0)

nodeList = nodeList.concat('!',node.label);

else

nodeList = node.label;

});

Shiny.setInputValue('nodes\_selected', nodeList);

}

})

// 设置边的颜色，开始时没有高亮的边。

.linkColor((link) => {return colorEdge;})

// 也可以通过指定边的颜色属性名称来指定颜色，这样可以先在igraph中处理

//.linkColor('color')

// 边动画演示，加亮显示的边有1个流动的点，从源到目标结点，其它边没有

.linkDirectionalParticles(link => highlightLinks.has(link) ? 2 : 0)

// 通过速度来表示流量，各条边的流量不同。

.linkDirectionalParticleSpeed(link => link.paths \* 0.01)

// 边动画演示中流动小点的尺寸

.linkDirectionalParticleWidth(3)

// 边动画演示中流动小点的颜色，不定义就是与边的颜色一样

.linkDirectionalParticleColor(() => 'orange')

// 点击边时显示动画，从源到目标结点。

.onLinkClick((link, event) =>{Graph.emitParticle(link)});

;

// Spread nodes a little wider

Graph.d3Force('charge').strength(-120);

// 把Graph对象传出事件处理函数之外，以便全局引用，比如清除选择的结点与边

mygraph = Graph;

// 更新加载状态

APPStatus.value = '网络数据已加载';

});

//alert('Registered!');

")),

tags$script(HTML("

//接收服务器传来清除选中结点的指令。

Shiny.addCustomMessageHandler('clearSelectedNodes', function(data) {

try{

// 清空 selectedNodes, highlightLinks，并设置选中源结点

selectedNodes.clear();

highlightLinks.clear();

var sourceNode = gData.nodes.find(obj => {return obj.id === data});

selectedNodes.add(sourceNode);

// 应用前面保存的全局图对象，重新设置选中与没有选中结点的颜色

mygraph.nodeColor(node => selectedNodes.has(node) ? colorNodeSelected : colorNode)

.linkColor((link) => {return colorEdge;});

// 更新结点选择状态

APPStatus.value = '已清除选择的结点';

}catch(error){

alert(error);

}

});

//alert('Registered!');

")),

tags$script(HTML("

//接收服务器传来设置源结点初始视野的指令。

Shiny.addCustomMessageHandler('setSourceNode', function(data) {

try{

selectedNodes.clear();

//设置起点机场为默认选中

var sourceNode = gData.nodes.find(obj => {return obj.id === data});

selectedNodes.add(sourceNode);

// 应用前面保存的全局图对象，重新设置选中与没有选中结点的颜色

mygraph.nodeColor(node => selectedNodes.has(node) ? colorNodeSelected : colorNode);

// 测试通过属性指定颜色

//mygraph.nodeColor('color')

var hops = document.getElementById('hops');

if (hops.value < '5'){

// 如果是5跳，就不设置变焦

// 设置初始焦点

// 参阅： https://github.com/vasturiano/3d-force-graph/issues/523

mygraph.onEngineTick(() => {

//focusNode();

// 设置焦距

var distance = 320;

var distRatio = 1 + distance/Math.hypot(sourceNode.x, sourceNode.y, sourceNode.z);

mygraph.cameraPosition(

// new position

{ x: sourceNode.x \* distRatio, y: sourceNode.y \* distRatio, z: sourceNode.z \* distRatio },

sourceNode, // lookAt ({ x, y, z })

3000 // ms transition duration

);

// From now on, we don't want to invoke this function anymore.

mygraph.onEngineTick(() => {});

});

}

// 等Force Engin执行完成后，图稳定下来，自动调整焦距到看到全图。

// https://github.com/vasturiano/3d-force-graph/issues/340

mygraph.onEngineStop(() => {

mygraph.zoomToFit(0,1,node =>node.id===sourceNode.id);

// From now on, we don't want to invoke this function anymore.

mygraph.onEngineStop(() => {});

});

// 更新结点选择状态

APPStatus.value = '已设自动调整焦距';

}catch(error){

alert(error);

}

});

//alert('Registered!');

")),

# 5/6

width = 10

)

)

)

八、在网络分析中使用Neo4j用户自定义函数与过程

在各行各业实际落地应用的网络分析中，作为载荷的目标子网是千变万化的，从图数据库中选择目标子网就要使用一些过滤结点或边或路径的方法或算法，Neo4j中有很多这样的机制，不过不一定能满足各种应用场景的需求，所以它提供了用户 [自定义函数与过程](https://neo4j.com/docs/java-reference/current/extending-neo4j/customized-code/)的机制。比如在机场航线稳定强环路查找的例子中，对于查询经过源机场的环路，用了 [APOC插件](https://neo4j.com/labs/apoc/4.1/overview/)基于集合的三个函数apoc.coll.avg()、apoc.coll.min()、apoc.coll.max()来过滤出稳定强环路。假如要用中位数而不是平均值（更符合实际），就没有现成的函数可用，需要自己写了。这里介绍一下Neo4j用户自定义函数，用户自定义过程在第二章[《发票交易网络分析》](#第二章发票交易网络分析)第七节[《Neo4j自定义过程之有向树形图》](#第七节Neo4j自定义过程之树形图)已经介绍过了，不再赘述。

1、 [Neo4j的用户自定义函数与过程](https://neo4j.com/docs/java-reference/current/extending-neo4j/customized-code/)

用户自定义过程与函数的区别是，自定义过程返回一个Stream流序列，在Cypher中引用时要用YIELD逐组提取。比如 [Neo4j自定义过程样板项目](https://github.com/neo4j-examples/neo4j-procedure-template)中提取结点连接类型的用户自定义过程 [GetRelationshipTypes.java](https://github.com/neo4j-examples/neo4j-procedure-template/blob/4.4/src/main/java/example/GetRelationshipTypes.java)， [它的Junit测试](https://github.com/neo4j-examples/neo4j-procedure-template/blob/4.4/src/test/java/example/GetRelationshipTypesTests.java)中是这样引用的：

/\*\*

\* We should be getting the correct values when there is only one type in each direction

\*/

@Test

public void shouldReturnTheTypesWhenThereIsOneEachWay() {

final String expectedIncoming = "INCOMING";

final String expectedOutgoing = "OUTGOING";

// In a try-block, to make sure we close the session after the test

try(Session session = driver.session()) {

//Create our data in the database.

session.run(String.format("CREATE (:Person)-[:%s]->(:Movie {id:1})-[:%s]->(:Person)", expectedIncoming, expectedOutgoing));

//Execute our procedure against it.

Record record = session.run(**"MATCH (u:Movie {id:1}) CALL example.getRelationshipTypes(u) YIELD outgoing, incoming RETURN outgoing, incoming**").single();

//Get the incoming / outgoing relationships from the result

assertThat(record.get("incoming").asList(Value::asString)).containsOnly(expectedIncoming);

assertThat(record.get("outgoing").asList(Value::asString)).containsOnly(expectedOutgoing);

}

}

看看 [它的源码](https://github.com/neo4j-examples/neo4j-procedure-template/blob/4.4/src/main/java/example/GetRelationshipTypes.java)，通过Java Annotation @Procedure定义为用户自定义过程及规定它的调用名字example.getRelationshipTypes，然后返回Stream类型。

package example;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;

import java.util.stream.Stream;

import org.neo4j.graphdb.Direction;

import org.neo4j.graphdb.Node;

import org.neo4j.graphdb.Relationship;

import org.neo4j.logging.Log;

import org.neo4j.procedure.Context;

import org.neo4j.procedure.Description;

import org.neo4j.procedure.Name;

import org.neo4j.procedure.Procedure;

/\*\*

\* This is an example showing how you could expose Neo4j's full text indexes as

\* two procedures - one for updating indexes, and one for querying by label and

\* the lucene query language.

\*/

public class GetRelationshipTypes {

// This gives us a log instance that outputs messages to the

// standard log, normally found under `data/log/console.log`

@Context

public Log log;

/\*\*

\* This procedure takes a Node and gets the relationships going in and out of it

\*

\* @param node The node to get the relationships for

\* @return A RelationshipTypes instance with the relations (incoming and outgoing) for a given node.

\*/

**@Procedure(value = "example.getRelationshipTypes")**

**@Description("Get the different relationships going in and out of a node.")**

public **Stream<RelationshipTypes>** getRelationshipTypes(@Name("node") Node node) {

List<String> outgoing = new ArrayList<>();

node.getRelationships(Direction.OUTGOING).iterator()

.forEachRemaining(rel -> AddDistinct(outgoing, rel));

List<String> incoming = new ArrayList<>();

node.getRelationships(Direction.INCOMING).iterator()

.forEachRemaining(rel -> AddDistinct(incoming, rel));

**return Stream.of(new RelationshipTypes(incoming, outgoing));**

}

/\*\*

\* Adds the distinct type of a relationship to the given List<String>

\*

\* @param list the list to add the distinct relationship type to

\* @param relationship the relationship to get the name() from

\*/

private void AddDistinct(List<String> list, Relationship relationship){

AddDistinct(list, relationship.getType().name());

}

/\*\*

\* Adds an item to a List only if the item is not already in the List

\*

\* @param list the list to add the distinct item to

\* @param item the item to add to the list

\*/

private <T> void AddDistinct(List<T> list, T item){

if(!list.contains(item))

list.add(item);

}

/\*\*

\* This is the output record for our search procedure. All procedures

\* that return results return them as a Stream of Records, where the

\* records are defined like this one - customized to fit what the procedure

\* is returning.

\* <p>

\* These classes can only have public non-final fields, and the fields must

\* be one of the following types:

\*

\* <ul>

\* <li>{@link String}</li>

\* <li>{@link Long} or {@code long}</li>

\* <li>{@link Double} or {@code double}</li>

\* <li>{@link Number}</li>

\* <li>{@link Boolean} or {@code boolean}</li>

\* <li>{@link Node}</li>

\* <li>{@link org.neo4j.graphdb.Relationship}</li>

\* <li>{@link org.neo4j.graphdb.Path}</li>

\* <li>{@link Map} with key {@link String} and value {@link Object}</li>

\* <li>{@link List} of elements of any valid field type, including {@link List}</li>

\* <li>{@link Object}, meaning any of the valid field types</li>

\* </ul>

\*/

public static class RelationshipTypes {

// These records contain two lists of distinct relationship types going in and out of a Node.

public List<String> outgoing;

public List<String> incoming;

public RelationshipTypes(List<String> incoming, List<String> outgoing) {

this.outgoing = outgoing;

this.incoming = incoming;

}

}

}

用户自定义函数则返回一个单一的值，又分为标量函数与聚合函数两种，它们的Java Annotation与API都不同。

这个是APOC中 [apoc.agg.median()函数的源码](https://github.com/neo4j-contrib/neo4j-apoc-procedures/blob/4.4/core/src/main/java/apoc/agg/Median.java)，它是一个聚合函数，通过@UserAggregationFunction定义，通过@Description定义函数调用的方式，要实现由Java Annotation指定的aggregate()与result()接口：

package apoc.agg;

import org.neo4j.procedure.\*;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

/\*\*

\* @author mh

\* @since 18.12.17

\*/

public class Median {

**@UserAggregationFunction("apoc.agg.median")**

**@Description("apoc.agg.median(number) - returns median for non-null numeric values")**

public MedianFunction median() {

return new MedianFunction();

}

public static class MedianFunction {

private List<Double> values = new ArrayList<>();

**@UserAggregationUpdate**

**public void aggregate(@Name("value") Object value) {**

if (value instanceof Number) {

values.add(((Number)value).doubleValue());

}

}

**@UserAggregationResult**

**public Object result() {**

if (values.isEmpty()) return null;

Collections.sort(values);

int size = values.size();

if (size % 2 == 1) {

**return values.get(size /2);**

} else {

**return (values.get(size /2-1) + values.get(size /2)) / 2D;**

}

}

}

}

这个函数为什么在稳定强环路查询中不能使用呢？因为它是对查询中的所有结点或边求中位数，而上面的查询需要对特定路径中边的权重（集合）求中位数，即按path分组，所以要自己写一个。

这个是APOC中 [apoc.coll.avg()函数的源码](https://github.com/neo4j-contrib/neo4j-apoc-procedures/blob/4.4/core/src/main/java/apoc/coll/Coll.java)，通过@UserFunction定义，也通过@Description定义函数调用的方式，是一个标量函数，它不需要实现特定的接口。

**@UserFunction**

**@Description("apoc.coll.avg([0.5,1,2.3])")**

public Double avg(@Name("numbers") List<Number> list) {

if (list == null || list.isEmpty()) return null;

double avg = 0;

for (Number number : list) {

avg += number.doubleValue();

}

**return (avg/(double)list.size());**

}

2、建立用户自定义函数Maven项目

可以参考 [Neo4j的文档来自行建立Maven项目](https://neo4j.com/docs/java-reference/current/extending-neo4j/project-setup/)的pom.xml项目描述文件，我直接拷贝 [Neo4j的样板项目](https://github.com/neo4j-examples/neo4j-procedure-template)稍作修改，因为它已经配好了编译与Junit测试的依赖，修改依赖的Neo4j服务器版本号等即可，这里改为4.2.1，因为目前中文社区版的版本是4.2。另外还要给个合适的groupId与artifactId及版本号，生成的jar包以artifactId+version命名。

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>**cn.jean.neo4j**</groupId>

<artifactId>**neo4j-functions**</artifactId>

<version>1.0.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>Jean's Neo4j Functions</name>

<description>Neo4j functions building from a Neo4j Procedure template project</description>

<properties>

<java.version>11</java.version>

<maven.compiler.release>${java.version}</maven.compiler.release>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<neo4j.version>4.2.1</neo4j.version>

<neo4j-java-driver.version>4.2.1</neo4j-java-driver.version>

<junit-jupiter.version>5.7.0</junit-jupiter.version>

<maven-shade-plugin.version>3.2.1</maven-shade-plugin.version>

<maven-compiler-plugin.version>3.8.1</maven-compiler-plugin.version>

<assertj.version>3.13.2</assertj.version>

<maven-surefire-plugin.version>2.22.2</maven-surefire-plugin.version>

</properties>

<dependencies>

<dependency>

<!-- This gives us the Procedure API our runtime code uses.

We have a `provided` scope on it, because when this is

deployed in a Neo4j Instance, the API will be provided

by Neo4j. If you add non-Neo4j dependencies to this

project, their scope should normally be `compile` -->

<groupId>org.neo4j</groupId>

<artifactId>neo4j</artifactId>

<version>${neo4j.version}</version>

<scope>provided</scope>

</dependency>

<!-- Test Dependencies -->

<dependency>

<!-- This is used for a utility that lets us start Neo4j with

a specific Procedure, which is nice for writing tests. -->

<groupId>org.neo4j.test</groupId>

<artifactId>neo4j-harness</artifactId>

<version>${neo4j.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<!-- Used to send cypher statements to our procedure. -->

<groupId>org.neo4j.driver</groupId>

<artifactId>neo4j-java-driver</artifactId>

<version>${neo4j-java-driver.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>${junit-jupiter.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.assertj</groupId>

<artifactId>assertj-core</artifactId>

<version>${assertj.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<artifactId>maven-enforcer-plugin</artifactId>

<version>3.0.0</version>

<executions>

<execution>

<id>enforce</id>

<goals>

<goal>enforce</goal>

</goals>

<phase>validate</phase>

<configuration>

<rules>

<requireJavaVersion>

<version>${java.version}</version>

</requireJavaVersion>

</rules>

</configuration>

</execution>

</executions>

</plugin>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>${maven-compiler-plugin.version}</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>${maven-surefire-plugin.version}</version>

</plugin>

<plugin>

<!-- This generates a jar-file with our procedure code,

plus any dependencies marked as `compile` scope.

This should then be deployed in the `plugins` directory

of each Neo4j instance in your deployment.

After a restart, the procedure is available for calling. -->

<artifactId>maven-shade-plugin</artifactId>

<version>${maven-shade-plugin.version}</version>

<configuration>

<createDependencyReducedPom>false</createDependencyReducedPom>

</configuration>

<executions>

<execution>

<phase>package</phase>

<goals>

<goal>shade</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

3、开发apoc.coll.median()函数

参考上面apoc.agg.median()的源码，实现很简单。

package apoc.coll;

import org.neo4j.procedure.Description;

import org.neo4j.procedure.Name;

import org.neo4j.procedure.UserFunction;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class MyFunctions {

@UserFunction

@Description("apoc.coll.median([0.5,1,2.3])")

public Double median(@Name("numbers") List<Number> values) {

if (values == null || values.isEmpty()) return null;

Collections.sort(values,new Comparator<Number>() {

@Override

public int compare(Number o1, Number o2) {

Double d1 = (o1 == null) ? Double.POSITIVE\_INFINITY : o1.doubleValue();

Double d2 = (o2 == null) ? Double.POSITIVE\_INFINITY : o2.doubleValue();

return d1.compareTo(d2);

}

});

int size = values.size();

if (size % 2 == 1) {

return values.get(size /2).doubleValue();

} else {

double first = values.get(size /2-1).doubleValue();

double second = values.get(size /2).doubleValue();

return (first + second)/2;

}

}

}

4、Junit单元测试

需要编写一个单元测试程序，测试时需要加载一个嵌入式的Neo4j Server并连接到它执行Cypher语句，Neo 4j样板项目中的 [JoinTest.java](https://github.com/neo4j-examples/neo4j-procedure-template/blob/4.4/src/test/java/example/JoinTest.java)稍为修改一下即可，然后在Project Explorer项目树上的程序名上右键->Run As->Junit Test即可。

package apoc.coll;

import org.junit.jupiter.api.AfterAll;

import org.junit.jupiter.api.BeforeAll;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.TestInstance;

import org.neo4j.driver.Driver;

import org.neo4j.driver.GraphDatabase;

import org.neo4j.driver.Session;

import org.neo4j.harness.Neo4j;

import org.neo4j.harness.Neo4jBuilders;

import apoc.coll.MyFunctions;

import static org.assertj.core.api.Assertions.assertThat;

@TestInstance(TestInstance.Lifecycle.PER\_CLASS)

public class TestMedian {

private Neo4j embeddedDatabaseServer;

@BeforeAll

void initializeNeo4j() {

this.embeddedDatabaseServer = Neo4jBuilders.newInProcessBuilder()

.withDisabledServer()

.withFunction(MyFunctions.class)

.build();

}

@AfterAll

void closeNeo4j() {

this.embeddedDatabaseServer.close();

}

@Test

void medianNumbers() {

// This is in a try-block, to make sure we close the driver after the test

try(Driver driver = GraphDatabase.driver(embeddedDatabaseServer.boltURI());

Session session = driver.session()) {

// When

//double result = session.run( "RETURN apoc.coll.median([0.5,1,2.3,2,3]) AS result").single().get("result").asDouble();

double result = session.run( "RETURN apoc.coll.median([4,2,3,1]) AS result").single().get("result").asDouble();

// Then

//assertThat( result).isEqualTo(2);

assertThat( result).isEqualTo(2.5);

}

}

}

5、打包发布

在项目根目录上右键->Run As->Maven Install，会在项目的target目录下生成打包好的jar文件。拷贝到服务器的plugins目录下，重启服务器加载，然后在Neo4j Browser中访问测试，执行上面的测试Cypher语句。

RETURN apoc.coll.median([0.5,1,2.3,2,3]) AS result;

RETURN apoc.coll.median([4,2,3,1]) AS result;

6、在Python函数中引用

替代原来引用的apoc.coll.avg()函数即可。

def airports\_ring(tx, source, length, amplitude, threshold):

cypher = (\

'''

MATCH (china:Airport)-[:IN\_COUNTRY]->(c:Country{code:"CN"})

WITH COLLECT(china.id) AS targets

MATCH path = (source:Airport{descr:$name})-[r:HAS\_ROUTE\*2..'''+str(int(length))+"]->(target)"+\

'''

WHERE target.descr = source.descr

AND ALL( node IN nodes(path) where node.id IN targets)

WITH path, [r in relationships(path)|r.distance] AS distances, $amplitude AS amplitude, $threshold AS threshold

//WITH path, apoc.coll.avg(distances) AS avgDist, apoc.coll.min(distances) as minDist, apoc.coll.max(distances) as maxDist

//WHERE minDist/avgDist >= (1-amplitude) AND maxDist/avgDist <= (1+amplitude) AND avgDist >= threshold

WITH path, apoc.coll.median(distances) AS medianDist, apoc.coll.min(distances) as minDist, apoc.coll.max(distances) as maxDist

WHERE minDist/medianDist >= (1-amplitude) AND maxDist/medianDist <= (1+amplitude) AND medianDist >= threshold

WITH path, relationships(path) as flights

UNWIND flights AS flight

WITH DISTINCT flight

MATCH (n:Airport)-[flight]->(m:Airport)

RETURN n.descr AS source, m.descr as target, flight.distance AS distance;

'''

)

print(cypher)

start = []; target=[]; distance=[]

for record in tx.run(cypher, name=source, amplitude=amplitude, threshold=threshold ):

start.append(record["source"])

target.append(record["target"])

distance.append(record["distance"])

# 从 list建立 pandas data frame

data = pd.DataFrame({"source":start,"target":target,"distance":distance})

return data

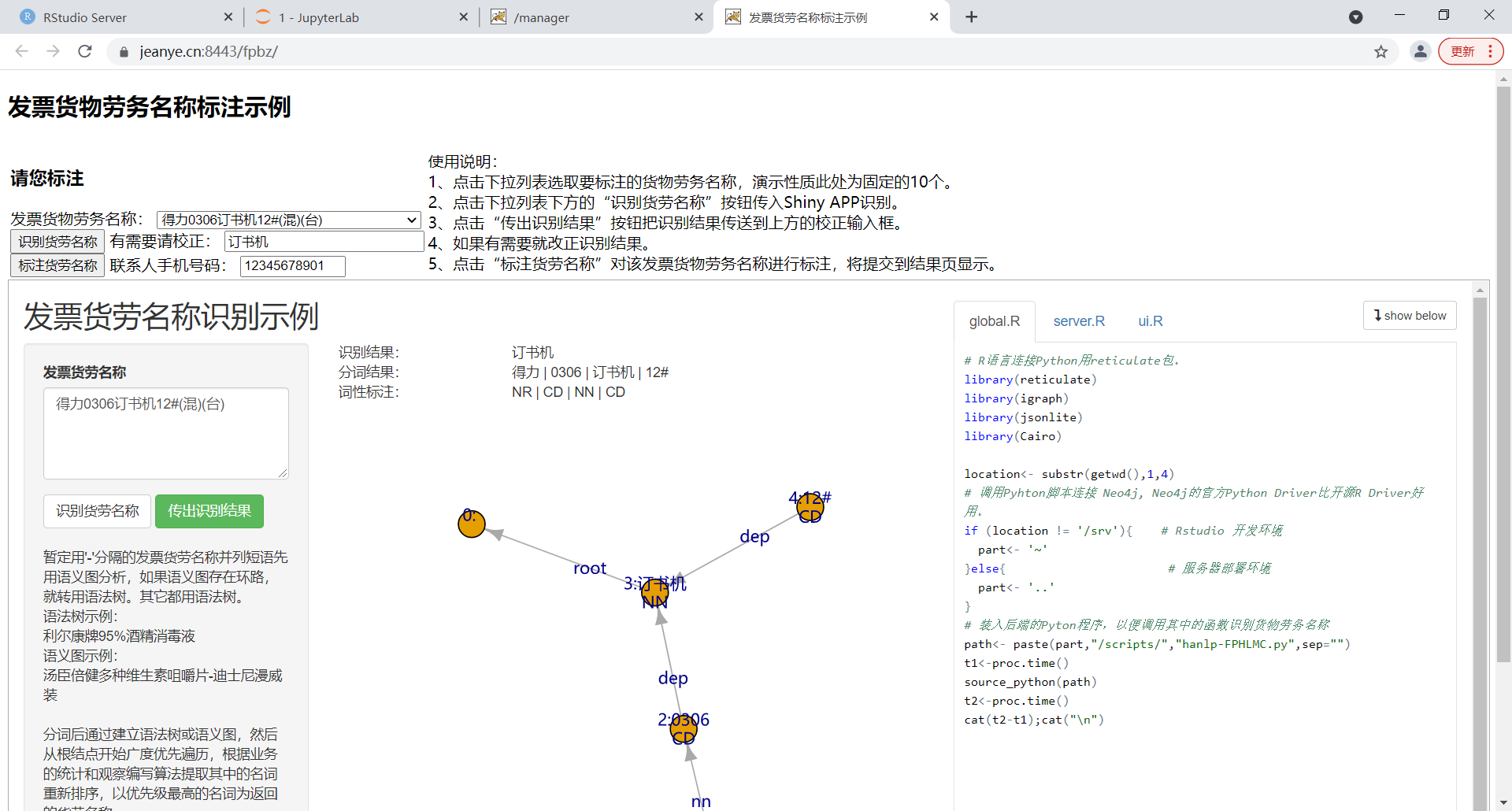
通过这个小例子可以看到，开发测试Neo4j用户自定义函数是比较简单方便的，这个机制可以解决很多应用场景中过滤结点、边和路径的特殊需要，有效的选出目标子网进一步分析。

九、本节小结

这个小例子（2D）所有运行的代码加起来不到400行（3D例子约800行），耗时5天完成，开发测试非常简便高效了。虽然只有一个可见的jsp页面，但实现了完整的多层应用体系，麻雀虽小，五脏俱全。一共用了R、Python、Java、Cypher、SQL、javascript、html 7种语言，集成了Shiny、Neo4j、Tomcat 3个平台，使用了Rstudio Server、JupyterLab、Eclipse、Neo4j Browser 4个IDE开发环境来开发，融会贯通，可谓集大成者，螺蛳壳里做道场。这说明每一种语言、工具和平台都有它的长处和用处，最佳实践是从解决问题出发，根据需要加以综合运用，不必拘束于语言与平台之争。

第四节 发票货物劳务名称识别APP

第一章《发票货物劳务名称识别》中已经开发了基于NLP技术的识别算法，并且实现了GPU加速并行处理。本节用Shiny APP封装发布一下，提供在线体验，以便更直观的解释与了解相关的技术与概念。本例运行在外网的虚拟主机上，体验地址： <https://jeanye.cn:8443/fpbz> 。实际使用时应该是在配备GPU的机器上后台并行批量处理的，甲方如果不希望把数据发送到公网上的云端，开源HanLP的Native API可以部署在内网本地运行，正好符合这个要求。



这个识别模型要有效运行，需要2个条件：

1、有一个发票专用的用户自定义词典。NLP深度学习预训练模型虽然在海量语料库上训练（HanLP是9千多万条），但发票货劳名称中有大量的专有名词，在日常语料中出现的频率较低，预训练模型可以达到80%的准确率或者更高，再提高就需要专用词典。

2、或者在现有的汇总发票货劳名称数据（汇总后数据量减少到10%左右）上训练一个发票专用的深度学习预训练模型。

这2个条件都需要对现有的汇总发票货劳名称作标注，即人工核对判断它是什么或判断识别是否正确，这是所有大数据应用中无法跨越的一步，没有标注好的数据，各种AI分类算法等机器学习算法就无法落地应用，税务系统的发票数据缺的正是这个。虽然第四章《个税明细申报数据分析》中提出了用时间序列模式识别算法来实现快速标注（风险标签），但这类方法无法应用于发票货劳名称识别这样的主题。

假定一个特定的经济体在某时段内流通的货物与劳务是相对稳定的，比如全国在流通的有几百万种这样的规模（税总等可以从发票明细汇总统计出来），移动互联网时代提供了动员社会力量来完成数据标注，打通发票大数据应用瓶颈的可能。比如全国百万税务干部，每人标注几条，这个任务几天内就可以完成。如果配合税法宣传，动员全社会来做，搞个实名认证（以防恶意标注）的开放标注抽奖活动（要有吸引力），比如设个一、二、三等奖送送5G手机，估计一个季度内也可以非常好的完成。这样迭代几次，模型的准确度就非常高了。具体工作可以外包给NLP技术公司去实施，专业的事要专业的人去干。税务干部自己标注的话要省一点，公开抽奖标注的话几百万预算应该也足够了，和后续产生不可斗量的经济社会效益相比，几乎可以忽略不计。这是一项非常必要也非常可行的税收大数据应用基础工程。

回到这个示例，识别算法封装成Shiny APP后，再集成到J2EE应用（此处是Tomcat Web APP）中以完成标注的其它流程。演示性质就只是显示标注的结果。所以这个小应用虽然只有2个网页，也是可以帮助完成标注工作的。

此处用的是hanlp 2.1.0b27，用pip install hanlp[full]安装，具体可参阅[安装文档](https://hanlp.hankcs.com/docs/install.html)。为适应CPU版运行环境及单条货劳名称识别，对第一章的源码稍作修改，算法则是一样的，下面具体看看。

Tomcat WEB APP，也是通过嵌入IFRAME去集成Shiny APP。

1、index.jsp，演示性质这里用了10个随机抽样的固定货劳名称，实际应用可以按频率排序动态选择，每个用户显示有限的未标注货劳名称即可。

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>发票货劳名称标注示例</title>

<script type="text/javascript">

//接受消息

window.addEventListener('message', receiveMessage);

**function** receiveMessage(event) {

//alert(event.data);

**var** hlmcRecognized = document.getElementById('hlmcRecognized');

**var** message = event.data.trim();

hlmcRecognized.value = message;

//if (message.length >25){

// hlmcRecognized.size = message.length;

//}else{

// hlmcRecognized.size = 25;

//}

}

//发送消息

**function** setHlmc(){

**try**{

**var** hlmc = document.getElementById("hlmc");

//alert(hlmc.value);

// document.getElementById("shinyappframe") 失败。

**var** win = document.getElementsByTagName('iframe')[0].contentWindow;

// postMessage()，窗口间通信，不受浏览器同源规则限制，可用于任何内嵌的iframe。

// 第二个参数是接受窗口的源，协议+域名+端口，可用\*表示发给所有窗口。

win.postMessage(hlmc.value,'\*');

//alert("Sent!");

} **catch** (error) {

alert(error);

}

}

</script>

</head>

<body>

<table>

<tr><h2>发票货物劳务名称标注示例</h2></tr>

<tr>

</tr>

<tr>

<td>

<h3>请您标注</h3>

<form action="bz.jsp">

<label>发票货物劳务名称：</label>

<select name="hlmc" id="hlmc">

<option value="利尔康牌95%酒精消毒液">利尔康牌95%酒精消毒液</option>

<option value="汤臣倍健多种维生素咀嚼片-迪士尼漫威装">汤臣倍健多种维生素咀嚼片-迪士尼漫威装</option>

<option value="92号汽油(国Ⅴ)">92号汽油(国Ⅴ)</option>

<option value="基础电信服务">基础电信服务</option>

<option value="思厨套装电压力锅">思厨套装电压力锅</option>

<option value="快递费">快递费</option>

<option value="得力0306订书机12#(混)(台)">得力0306订书机12#(混)(台)</option>

<option value="一次性使用血液灌流器">一次性使用血液灌流器</option>

<option value="真空通孔塞孔+研磨">真空通孔塞孔+研磨</option>

<option value="转关入闸、过磅费">转关入闸、过磅费</option>

</select><br>

<input type="button" value="识别货劳名称" onclick="setHlmc()" />

有需要请校正：

<input id="hlmcRecognized" name="hlmcRecognized" type="text" value="" size=26></input><br>

<input type="submit" value="标注货劳名称" />

联系人手机号码：

<input id="mobile" name="mobile" type="text" value="12345678901" size=11></input><br>

</form>

</td>

<td>

使用说明：<br>

1、点击下拉列表选取要标注的货物劳务名称，演示性质此处为固定的10个。<br>

2、点击下拉列表下方的“识别货劳名称”按钮传入Shiny APP识别。<br>

3、点击“传出识别结果”按钮把识别结果传送到上方的校正输入框。<br>

4、如果有需要就改正识别结果。<br>

5、点击“标注货劳名称”对该发票货物劳务名称进行标注，将提交到结果页显示。<br>

</td>

</tr>

</table>

<iframe id="shinyappframe" src="https://jeanye.cn:4443/shiny/users/jean/hanlpFP/" style="border: 1px solid #AAA; width: 100%; height: 700px"></iframe>

</body>

</html>

2、bz.jsp，完成标注的处理流程，演示性质只是显示传入的参数。实际应用的话可以向手机发送一个验证码，然后输入验证码以确保实名标注。

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Insert title here</title>

</head>

<body>

<h3>感谢您的参与！</h3>

<p>标注结果：<br>

发票货物劳务名称：<%=**new** String(request.getParameter("hlmc").getBytes("ISO-8859-1"),"UTF-8") %><br>

识别标注为：<%=**new** String(request.getParameter("hlmcRecognized").getBytes("ISO-8859-1"),"UTF-8") %><br>

联系人手机号码：<%=**new** String(request.getParameter("mobile").getBytes("ISO-8859-1"),"UTF-8") %><br>

<br><br>

<a href="index.jsp">返回标注输入页面</a>

</body>

</html>

3、global.R，加载要调用的货劳名称识别Python函数。

# R语言连接Python用reticulate包。

library(reticulate)

library(igraph)

library(jsonlite)

library(Cairo)

location<- substr(getwd(),1,4)

# 调用Pyhton脚本连接 Neo4j, Neo4j的官方Python Driver比开源R Driver好用。

if (location != '/srv'){ # Rstudio 开发环境

part<- '~'

}else{ # 服务器部署环境

part<- '..'

}

# 装入后端的Pyton程序，以便调用其中的函数识别货物劳务名称

path<- paste(part,"/scripts/","hanlp-FPHLMC.py",sep="")

t1<-proc.time()

source\_python(path)

t2<-proc.time()

cat(t2-t1);cat("\n")

4、server.R，调用Python货劳名称识别函数，识别结果用igraph生成语法树或语义图，并输出成SVG图。注意renderImage()对SVG图的处理。

# renderImage()处理SVG图参阅下面的连接

# https://stackoverflow.com/questions/39093777/renderimage-and-svg-in-shiny-app

shinyServer(function(input, output, session) {

hlmc<- reactive({

# 调用Python函数识别货劳名称，函数的返回值赋给reactive变量hlmc，这是个字典，R语言自动转换为列表

parseHlmc(preprocess(input$hlmc))

})

observe({

# 控制台输出货劳名称识别结果

result<- hlmc()

updateTextAreaInput(session, "hlmc", value = input$hlmc)

})

output$resultHlmc<- renderText({

result<- hlmc()

result[[1]]

})

output$segwords<- renderText({

result<- hlmc()

gdf<- fromJSON(result[[4]])[[1]]

paste(gdf$form, collapse=" | ")

})

output$postags<- renderText({

result<- hlmc()

gdf<- fromJSON(result[[4]])[[1]]

paste(gdf$upos, collapse=" | ")

})

## create .svg file of the plot and render it with renderImage()

output$plot\_as\_svg <- renderImage({

width <- session$clientData$output\_plot\_width

height <- session$clientData$output\_plot\_height

mysvgwidth <- width/96

mysvgheight <- height/96

# 引用货劳名称识别的结果

result<- hlmc()

# 用igraph重建语法树

gdf<- fromJSON(result[[4]])[[1]]

if (result[[3]] == 0){

# 语法树

verts<- gdf[,c(1,2,5)]

vertsRoot<- data.frame(id=0, form=" ", upos=" ")

verts<- rbind(verts, vertsRoot)

names(verts)<-c("order","word","pos")

verts$name<-as.character(verts$order)

edges<-gdf[,c(1,3,4)]

# 建立 igraph图

g <- graph\_from\_data\_frame(edges, directed=TRUE, vertices=verts)

V(g)$label<-paste(paste(V(g)$name,V(g)$word,sep=":"),V(g)$pos, sep="\n")

E(g)$label<-E(g)$deprel

} else {

# 语义图

verts<- gdf[,c(1,2,4)]

vertsRoot<- data.frame(id=0, form=" ", upos=" ")

verts<- rbind(verts, vertsRoot)

names(verts)<-c("order","word","pos")

verts$name<-as.character(verts$order)

links<- gdf$deps

# 空的向量

source<- target<- relType<-c()

for (i in 1:length(links)){

# links2 是一个 n\*2的矩阵

links2<- links[[i]]

for (j in 1: length(links2[,1])){

# 第j条边

link <- links2[j,]

source[length(source)+1]<- gdf$id[[i]]

target[length(target)+1]<- link[[1]]

relType[length(relType)+1]<- link[[2]]

}

}

edges<- data.frame(id=source, head=target, deps=relType)

edges$head<- as.integer(edges$head)

# 建立 igraph图

g <- graph\_from\_data\_frame(edges, directed=TRUE, vertices=verts)

V(g)$label<-paste(paste(V(g)$name,V(g)$word,sep=":"),V(g)$pos, sep="\n")

E(g)$label<-E(g)$deps

}

# 输出SVG到临时文件

# This file will be removed later by renderImage

outfile <- tempfile(fileext='.svg')

# Cairo要指定中文字体

CairoFonts(regular="AR PL UKai CN:Book")

CairoSVG(outfile)

plot.igraph(g)

dev.off()

# Return a list containing the filename

list(src = normalizePath(outfile),

contentType = 'image/svg+xml',

width = width,

height = height,

alt = "货劳名称识别语法树")

},

# SVG图发到客户端后，renderImage删除临时文件

deleteFile = TRUE

)

})

5、ui.R，这里屏蔽了输入变量input$hlmc的响应，输好后再按“识别货劳名称”按钮触发，是为了避免输入的过程中频繁触发服务器端的更新代码。

fluidPage(

tags$head(

tags$script(HTML("

// 记录父窗口，初始为空。

parent = null;

// 处理接收到的消息。

window.addEventListener('message', function(e) {

//alert(e.data);

try{

//记录父窗口以备回发信息

parent = e.source;

//向服务器发送input变量更新消息

Shiny.setInputValue('hlmc', e.data);

} catch (error){

alert(error);

}

},false);

// 点击时触发识别

$(document).on('click', '.btn-recognize', function (evt) {

evt.preventDefault();

var hlmc = document.getElementById('hlmc');

if (hlmc.value ==null || hlmc.value ==''){

alert('没有输入发票货劳名称！');

}else{

try{

Shiny.setInputValue('hlmc', hlmc.value);

} catch(error){

alert(error);

}

}

});

// 点击时传出参数

$(document).on('click', '.btn-success', function (evt) {

evt.preventDefault();

var selected = document.getElementById('resultHlmc');

if (selected ==null){

alert('没有识别结果!');

}else{

if (parent == null){

alert(selected.innerHTML);

} else {

try{

parent.postMessage(selected.innerHTML, '\*');

} catch(error){

alert(error);

}

}

}

});

// 触发了shiny:inputchanged事件，屏蔽输入引起的改变

$(document).on('shiny:inputchanged', function(event) {

if (event.name === 'hlmc') {

event.preventDefault();

}

});

"))

),

titlePanel("发票货劳名称识别示例"),

sidebarLayout(

sidebarPanel(

textAreaInput("hlmc","发票货劳名称", rows =4, resize="vertical", value ="利尔康牌95%酒精消毒液"),

actionButton("recognize", "识别货劳名称", class = "btn-recognize"),

actionButton("sendout", "传出识别结果", class = "btn-success"),

HTML("

<br><br>暂定用'-'分隔的发票货劳名称并列短语先用语义图分析，如果语义图存在环路，就转用语法树。其它都用语法树。<br>

语法树示例：<br>

利尔康牌95%酒精消毒液<br>

语义图示例：<br>

汤臣倍健多种维生素咀嚼片-迪士尼漫威装<br>

<br>

分词后通过建立语法树或语义图，然后从根结点开始广度优先遍历，根据业务的统计和观察编写算法提取其中的名词重新排序，以优先级最高的名词为返回的货劳名称。<br>

发票货劳名称一定是名词，这是NLP技术在特定场景名词短语分析中的应用实例。HanLP的Native API下载NLP深度学习预训练模型在虚拟主机本地执行操作。

"),

width = 4

),

mainPanel(

HTML("

<table>

<tr>

<td>识别结果：</td>

<td>

"),

textOutput("resultHlmc"),

HTML("

</td>

</tr>

<tr>

<td>分词结果：</td>

<td>

"),

textOutput("segwords"),

HTML("

</td>

</tr>

<tr>

<td>词性标注：</td>

<td>

"),

textOutput("postags"),

HTML("

</td>

</tr>

<tr>

<td colspan=2>

"),

# Use imageOutput to place the image on the page

imageOutput("plot\_as\_svg"),

HTML("

</td>

</tr>

</table>

")

)

))

6、hanlp-FPHLMC.PY，是这个Shiny APP的核心底座，要装入各种重量级的深度学习预训练模型，大概要1G的内存和30秒，APP启动时在global.R中一次性装入。注意HanLP2.1仍处于beta阶段，各发布版本的API比如自定义词典的组织和使用，语法树与语义图返回的格式等可能会在不同的版本之间有变化，需要按使用的版本调试调整。预训练模型第一次调用时会下载到~/.hanlp之下（Linux），大的有几百M，一共要几G。自定义词典的格式是CSV文件，每个词条1行2列：词条,词性，读入后再拆分，分别用于分词（set词典）与词性标注（dictionary词典）。比如：

服务费,NN

布洛芬缓释胶囊,NM

单任务预训练模型的精度更高，所以分词、词性标注、语法分析、语义分析都作为单任务定义，然后用pipeline机制串联起来执行。适应Shiny APP演示的性质，每次只传入处理一条货劳名称。其它请参阅第一章《发票货劳名称识别》，此处不再赘述。HanLP的具体使用请参阅其[GitHub主页](https://github.com/hankcs/HanLP)。

# pipeline只处理单个货劳名称识别，非并行模式。如果传入多个句子则是并行模式，CPU与GPU版都支持。

# hanlp 2.1.0b27，HanLP 2.1仍处于alpha状态，各alpha版本间会有变化，

# 比如自定义词典的格式与使用方式，依存句法分析与语义依存分析的输出等。

import hanlp

# 用于挂载用户自定义词典分词，

# 发票货物劳务名称中有大量的专有名词，并不常见于训练用的语料库中，导致分词不够准确，需要发票专用的自定义词典。

from hanlp.components.tokenizers.transformer import TransformerTaggingTokenizer

# 用于为上面的用户自定义词典作词性标注，提取货物劳务名称需要标注词性，货物劳务名称显然都是名词，这是个名词短语。

from hanlp.components.mtl.tasks.pos import TransformerTagging

import pandas as pd

import time, random, math

import json,re

import collections

from igraph import \*

# 装入后面用到的预训练模型，都是TensorFlow或PyTorch上的NLP深度学习预训练模型

# 第一次调用时会下载到~/.hanlp之下，大的有几百M，一共要几G。

# TensorFlow 2.6 , Pytorch 1.11.0

# 中文分词模型

# COARSE\_ELECTRA\_SMALL\_ZH预训练模型的效果没有FINE\_ELECTRA\_SMALL\_ZH好，前者是粗粒度，后者是细粒度

# LARGE\_ALBERT\_BASE预训练模型加载失败，可能是需要GPU。

# tokenizer = hanlp.load(hanlp.pretrained.tok.COARSE\_ELECTRA\_SMALL\_ZH)

tokenizer: TransformerTaggingTokenizer = hanlp.load(hanlp.pretrained.tok.FINE\_ELECTRA\_SMALL\_ZH)

# 词性标注模型

tagger: TransformerTagging = hanlp.load(hanlp.pretrained.pos.CTB9\_POS\_ELECTRA\_SMALL)

# 依存句法分析模型

syntactic\_parser = hanlp.load(hanlp.pretrained.dep.CTB9\_DEP\_ELECTRA\_SMALL)

# 语义依存分析模型

# semantic\_parser = hanlp.load('SEMEVAL16\_ALL\_ELECTRA\_SMALL\_ZH')

semantic\_parser = hanlp.load(hanlp.pretrained.sdp.SEMEVAL16\_ALL\_ELECTRA\_SMALL\_ZH)

# 从CSV文件装入用户自定义词典

types = {"word":str,"pos":str}

dicZdy = pd.read\_csv('~/data/词汇导入demo-HanLP.csv', sep=',', header=None, dtype=types, encoding="GBK")

# 分词字典是个集合

dicTag = set(dicZdy.iloc[:,0])

tokenizer.dict\_force = dicTag

# 词性标注字典是个字典

dictPos = dict(collections.OrderedDict(zip(dicZdy.iloc[:,0],dicZdy.iloc[:,1])))

tagger.dict\_tags= dictPos

# pipeline模式，只处理单个货劳名称.

pipeline = hanlp.pipeline() \

.append(tokenizer, output\_key='tokens') \

.append(tagger, input\_key='tokens', output\_key='part\_of\_speech\_tags') \

.append(syntactic\_parser, input\_key='tokens', output\_key='syntactic\_dependencies') \

.append(semantic\_parser, input\_key='tokens', output\_key='semantic\_dependencies')

# 自定义函数， 返回依存句法分析json结果字符串，抽取key子集

def syntacticParse(parser, text):

res = parser(text)

# items = res.syntactic\_dependencies # HanLP 2.0 -->2.1

sentence = res["syntactic\_dependencies"]

pos = res["part\_of\_speech\_tags"]

items2 = {}; items2.setdefault("items",[])

i= 0

for word in sentence:

item2 ={key:value for key ,value in word.items() \

# if key in ["cpos","deprel","form","head","id"]} # HanLP 2.0 -->2.1

if key in ["deprel","form","head","id"]}

item2["upos"] = pos[i]

i = i +1

# print(item2)

items2.get("items").append(item2)

reStr = json.dumps(items2,ensure\_ascii=False)

return reStr

# 建立语法树（图）函数

def getTree(text):

res = json.loads(text)

try:

items = res["items"]

except KeyError:

print(text)

return None

ids = []; pos = []; words = []; head = []; deprel = []

for item in items:

# ids.append(item["id"]); words.append(item["form"]); pos.append(item["cpos"]) # HanLP 2.0 -->2.1

ids.append(item["id"]); words.append(item["form"]); pos.append(item["upos"])

head.append(item["head"]); deprel.append(item["deprel"])

vs = pd.DataFrame({"id":ids,"word":words,"postag":pos})

es = pd.DataFrame({"id":ids,"head":head,"deprel":deprel})

vd = vs.to\_dict(orient='records')

ed = es.to\_dict(orient='records')

g = Graph.DictList(vd,ed, vertex\_name\_attr="id",edge\_foreign\_keys=('id', 'head'),directed=True)

# 给根节点赋非空值

vs = g.vs.select(id=0)

vs["word"] = ""; vs["postag"] = ""

g.vs["label"]=[str(v["id"])+":"+v["word"]+"\n"+v["postag"] for v in g.vs]

g.es["label"]=g.es["deprel"]

return g

# 自定义函数， 返回语义依存分析json结果字符串，抽取key子集

def semanticParse(parser, text):

res = parser(text)

# items = res.syntactic\_dependencies # HanLP 2.0 -->2.1

sentence = res["semantic\_dependencies"]

pos = res["part\_of\_speech\_tags"]

items2 = {}; items2.setdefault("items",[])

i= 0

for word in sentence:

item2 ={key:value for key ,value in word.items() \

# if key in ["cpos","deprel","form","head","id"]} # HanLP 2.0 -->2.1

if key in ["deps","form","id"]}

item2["upos"] = pos[i]

i = i +1

#print(item2)

items2.get("items").append(item2)

reStr = json.dumps(items2,ensure\_ascii=False)

return reStr

# 建立语义网络（图）函数， 语义分析结果为一个有向无环图，称为语义依存图（Semantic Dependency Graph）。

# 不过测试例子显示返回结果中存在环路。

def getTree2(text):

res = json.loads(text)

try:

items = res["items"]

except KeyError:

print(text)

return None

ids = []; pos = []; words = []

source = []; target=[]; reltype = []

for item in items:

# ids.append(item["id"]); words.append(item["form"]); pos.append(item["cpos"]) # HanLP2.0 -->2.1

ids.append(item["id"]); words.append(item["form"]); pos.append(item["upos"])

# 展开语义图的边

for link in item["deps"]:

source.append(item["id"])

target.append(link[0])

reltype.append(link[1])

# 补充根结点, 并给属性赋非空值

ids.append(0); words.append(""); pos.append("")

vs = pd.DataFrame({"id":ids,"word":words,"postag":pos})

es = pd.DataFrame({"id":source,"head":target,"deps":reltype})

vd = vs.to\_dict(orient='records')

ed = es.to\_dict(orient='records')

g = Graph.DictList(vd,ed, vertex\_name\_attr="id",edge\_foreign\_keys=('id', 'head'),directed=True)

g.vs["label"]=[str(v["id"])+":"+v["word"]+"\n"+v["postag"] for v in g.vs]

g.es["label"]=g.es["deps"]

return g

# 根据业务上的观察与统计规则，遍历图并输出名词的顺序（对包含的名词重新排序）。

# 从根结点开始遍历语法树，取最先出现的名词作货劳名称，所以层级高的优先，同层级的在后面的优先

def get\_hlmc(g, verbose = False):

# 建立邻接表

adj = [[n.index for n in v.neighbors("in")] for v in g.vs]

spcialWords = re.compile("[\.|\-|%]") #特殊名称， "."软件版本号， "-"具体型号，跳过

# 不知为何，上面的语句在Rstudio中执行有问题，在JupyterHub中及Spyder中都没有问题。先跳过。

# spcialWords = re.compile("[%]")

spcialPuncs = ["-", "/", "\*", "+"] #特殊标点符号，标示里面包含并列关系,

skipWords = re.compile("[顶|装]")

# 广度优先搜索遍历语法树

def traverse(adj, paths, depth):

# 记录遍历的深度

depth += 1; nxt\_paths = []

# 遍历所有的路径

for path in paths['paths']:

v = g.vs[path[-1]]

# 如果不是并列词，则反转邻接表，改成同层级的后面的优先，排在后面的名词优先，

# 发票语法结构的特点，前面的名词多数是修饰语

if v["word"] not in spcialPuncs and depth>1:

adj[path[-1]].reverse()

for idx in adj[path[-1]]:

v1 = g.vs[idx]

# 如 汤臣倍健多种维生素咀嚼片-迪士尼漫威装, 根结点是 漫威装

if v1["word"] in spcialPuncs:

adj[path[-1]].reverse() # 子结点中有并列关系，邻接表恢复为正常顺序。

break

# 遍历路径中最后一个结点的所有邻居

for nxt in adj[path[-1]]:

v = g.vs[nxt]

if verbose:

print(depth,v["word"],v["postag"])

if "N" in v["postag"]: # 如果是名词，则加入候选货劳清单中

flag = True

if spcialWords.findall(v["word"]) and v["postag"]!= "NM" : # 软件版本号，具体型号，跳过，药名除外。

flag = False

if v["postag"]== "NT": # 品牌，跳过

flag = False

matched = skipWords.search(v["word"]) # 检查是否包含要跳过的特殊名词

if matched:

if matched.span()[0]+1 == len(v["word"]): # 漫威装 在最后，跳过；安装费 在中间，接受

flag = False

if flag:

paths["hlmc"].append(v["word"])

paths["pos"].append(v["postag"])

nxt\_path = path + [nxt] # 在路径中加入下一个结点，生成新的路径

nxt\_paths.append(nxt\_path)

# 路径的深度增加了一级

paths['paths'] = nxt\_paths

if len(nxt\_paths)==0:

# 没有新路径了

return paths

else:

# 往路径深度递归调用一级

return traverse(adj, paths, depth)

# 从根结点开始递归遍历语法树

v = g.vs.select(id=0)[0]

# 返回候选货劳名称列表，排在前面的在语法树中的层级较高，同层级中则排在较后的优先级较高

res = traverse(adj, {'paths': [[v.index]], 'hlmc': [], "pos": []}, 0)

# 没有找到名词，返回一级结点作货劳名称

if len(res["hlmc"]) == 0:

v1 = v.neighbors("in")[0]

res["hlmc"].append(v1["word"]); res["pos"].append(v1["postag"])

return res

# 句法依存分析或语义依存分析前对汇总后的货物劳务名称做预处理

def preprocess(text):

# 去掉前后空格等特殊处理

text = text.strip()

if text == "." or text == "/" or text == "-":

text = ""

# 去掉几种括号及其中内容，如果去掉后没有内容了，则予以保留

text1 = text

text = re.sub("\\(.\*?\\)|（.\*?）", "", text)

if len(text) == 0:

text = text1

text1 = text

text = re.sub("\\[.\*?\\]|【.\*?】", "", text)

if len(text) == 0:

text = text1

text1 = text

text = re.sub("\\{.\*?\\}|｛.\*?｝", "", text)

if len(text) == 0:

text = text1

# 去掉商品及货物分类简称

text1 = text

text = re.sub("\\\*.\*?\\\*", "", text)

if len(text) == 0:

text = text1

# 去掉单边括号后的内容，这种情况不少

text1 = text

text = text.split("（")[0] # 722 维达（Vinda 822 荣耀8全网通（FRD 825 彩电挂式安装调试卡（

if len(text) == 0:

text = text1

text1 = text

text = text.split("）")[0]

if len(text) == 0:

text = text1

text1 = text

text = text.split("(")[0]

if len(text) == 0:

text = text1

text1 = text

text = text.split(")")[0]

if len(text) == 0:

text = text1

text1 = text

text = re.sub("及", " ", text) # 国际快件费及附加 -> 国际快件费 附加

if len(text) == 0:

text = text1

text1 = text

text = text.split(" ")[0] # 空格分隔的多个并列，只留第一个，如 "小米5C 移动版 3"

if len(text) == 0:

text = text1

text = re.sub("[0-9]L", "", text) # 胶瓶美汁源果粒橙12\*1.25L，去掉L，引起分词错误

return text

# 分析并提取货劳名称

def parseHlmc(text):

# 分别存放返回的json结果，识别的货物劳务名称与它的词性

result = {}

Re1 = re.compile("-")

Re2 = re.compile(r"[1-9a-zA-Z][-]")

Re3 = re.compile(r"[-][1-9a-zA-Z]")

Re4 = re.compile(r"[1-9a-zA-Z][-][1-9a-zA-Z]")

# 此处只处理单条

# 某些情况， 在图上形成了循环，没有 id=0的根结点，引起出错。

# "-" 并列式，先用语义分析尝试，效果要好一点

reStr = None

if Re1.search(text) and not Re4.search(text):

try: # 用语义依存分析尝试，有向无环图。

text2 = re.sub("-", "\*", text) # 替换"-"为"\*"，以帮助正确分词

reStr = semanticParse(pipeline, text2)

g = getTree2(reStr) # 转换成语义图

hlmc = get\_hlmc(g, False) # 遍历语义图得到货物劳务名称

result["hlmc"]=hlmc["hlmc"][0]; result["pos"]=hlmc["pos"][0]; result["method"]=1

except Exception as e: #KeyError:

print(repr(e))

try: # 调用依存句法分析再尝试。

reStr = syntacticParse(pipeline, text)

g = getTree(reStr) # 转换成语法树

hlmc = get\_hlmc(g, False) # 遍历语法树得到货物劳务名称

result["hlmc"]=hlmc["hlmc"][0]; result["pos"]=hlmc["pos"][0]; result["method"]=0

except Exception: # 图上形成了循环，没有 id=0的根结点，引起出错。

result["hlmc"]=None; result["pos"]=None; result["method"]=None

else:

try: # 调用依存句法分析

reStr = syntacticParse(pipeline, text)

g = getTree(reStr) # 转换成语法树

hlmc = get\_hlmc(g, False) # 遍历语法树得到货物劳务名称

result["hlmc"]=hlmc["hlmc"][0]; result["pos"]=hlmc["pos"][0]; result["method"]=0

except Exception as e: #图上形成了循环，没有 id=0的根结点，引起出错。

print(repr(e))

try: # 用语义依存分析再尝试，有向无环图。

reStr = semanticParse(pipeline, text)

g = getTree2(reStr) # 转换成语义图

hlmc = get\_hlmc(g, False) # 遍历语义图得到货物劳务名称

result["hlmc"]=hlmc["hlmc"][0]; result["pos"]=hlmc["pos"][0]; result["method"]=1

except Exception: #KeyError:

result["hlmc"]=None; result["pos"]=None; result["method"]=None

result["res"] = reStr

return result

# text = "利尔康牌95%酒精消毒液"

# res = parseHlmc(pipeline, preprocess(text))

# print(res)

第五节 房价回归分析APP

这个APP展示了一个典型的税收大数据AI落地应用场景，所以还是有必要补充收录一下。

1、集成进业务应用系统的业务流程中。与其它APP一样用嵌入的方式，业务流程也简单，但所有的元素都实现了。

2、交互式UI，交互式图表分析数据，所有分析最后都要业务的参与，由业务使用，并从业务得到反馈改进。技术最后是为业务服务，业务能否与技术有效融合是应用成功的关键。这是本例通用典型的地方。

3、应用机器学习AI算法分析数据，并且非常有效。AI算法的作用是总结了业务经验与数据统计规律，全量扫描，这样就突破了案头分析的人工与经验局限，便于进行规模化、常态化、自动化的精准分析。这也是本例通用典型的地方。

4、有效的深度分析都需要编程，机器学习与深度学习等AI算法的应用都需要编程。甲方参与到哪个程度与层面，会影响整个系统的体系设计与UI，所以甲方应该准备好培养一支掌握当代大数据分析技术的队伍，并且要精通业务或形成业务与技术有效融合的团队。

5、有效的落地应用，一定是有效集成各个系统与平台，综合应用各种技术。在实际落地应用中，要根据应用场景的需要组合运用，一种技术，一个平台，或者一种语言，无法包打天下，这要成为一个基本的认识。

下面具体看看。

（一）外层Tomcat Web APP

这个Web APP很简单，只有两个页面，主要是演示Shiny APP集成嵌入传统J2EE平台业务系统的业务流程。具体可以到我的研究网站上[体验一下](https://jeanye.cn:8443/melbourne)。

1、index.jsp

这个页面与前面的例子一样，通过IFRAME嵌入对部署在不同主域上的Shiny APP的引用，通过HTML5页面的postMessage机制在不同主域的页面之间传递数据。



<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>墨尔本房价回归示例</title>

<script type="text/javascript">

//接受消息

window.addEventListener('message', receiveMessage);

**function** receiveMessage(event) {

//alert(event.data);

**var** outliers = document.getElementById('outliers');

**var** message = event.data.trim();

outliers.value = message;

**if** (message.length >24){

outliers.size = message.length;

}**else**{

outliers.size = 24;

}

}

//发送消息

**function** setAlgo(){

**try**{

**var** algo = document.getElementById("algo");

//alert(algo.value);

// document.getElementById("shinyappframe") 失败。

**var** win = document.getElementsByTagName('iframe')[0].contentWindow;

// postMessage()，窗口间通信，不受浏览器同源规则限制，可用于任何内嵌的iframe。

// 第二个参数是接受窗口的源，协议+域名+端口，可用\*表示发给所有窗口。

win.postMessage(algo.value,'\*');

//alert("Sent!");

} **catch** (error) {

alert(error);

}

}

</script>

</head>

<body>

<table>

<tr><h2>墨尔本房价回归示例</h2></tr>

<tr>

</tr>

<tr>

<td>

<form action="details.jsp" method="post">

<label>回归算法：</label>

<select name="algo" id="algo">

<option value="SVM">SVM</option>

<option value="RandomForest">RandomForest</option>

<option value="GBR">GBR</option>

<option value="XGB">XGB</option>

<option value="LigthGBM">LigthGBM</option>

<option value="CatBoost" selected>CatBoost</option>

<option value="Blend">Blend</option>

</select>

<input type="button" value="识别异常房价" onclick="setAlgo()" /><br>

房价异常行号：

<input id="outliers" name="outliers" type="text" value="" size=24></input><br>

<input type="submit" value="进一步处理" />

</form>

</td>

<td>

使用说明：<br>

1、点击下拉列表选取要使用的回归算法。<br>

2、点击下拉列表旁边的“识别异常房价”按钮传入Shiny APP识别。<br>

3、可以调整异常值阀值以改变异常值的范围。<br>

4、在页面底部的异常房价列表中选择要传出处理的行，可以翻页选中多行（点击）或取消选中某行（重复点击）。<br>

5、点击“传出选中的行”按钮把识别结果传送到上方的房价异常行号输入框。<br>

6、点击“进一步处理”提交到结果页显示或进一步处理，演示性质只是简单按行号匹配原始数据并显示。<br>

</td>

</tr>

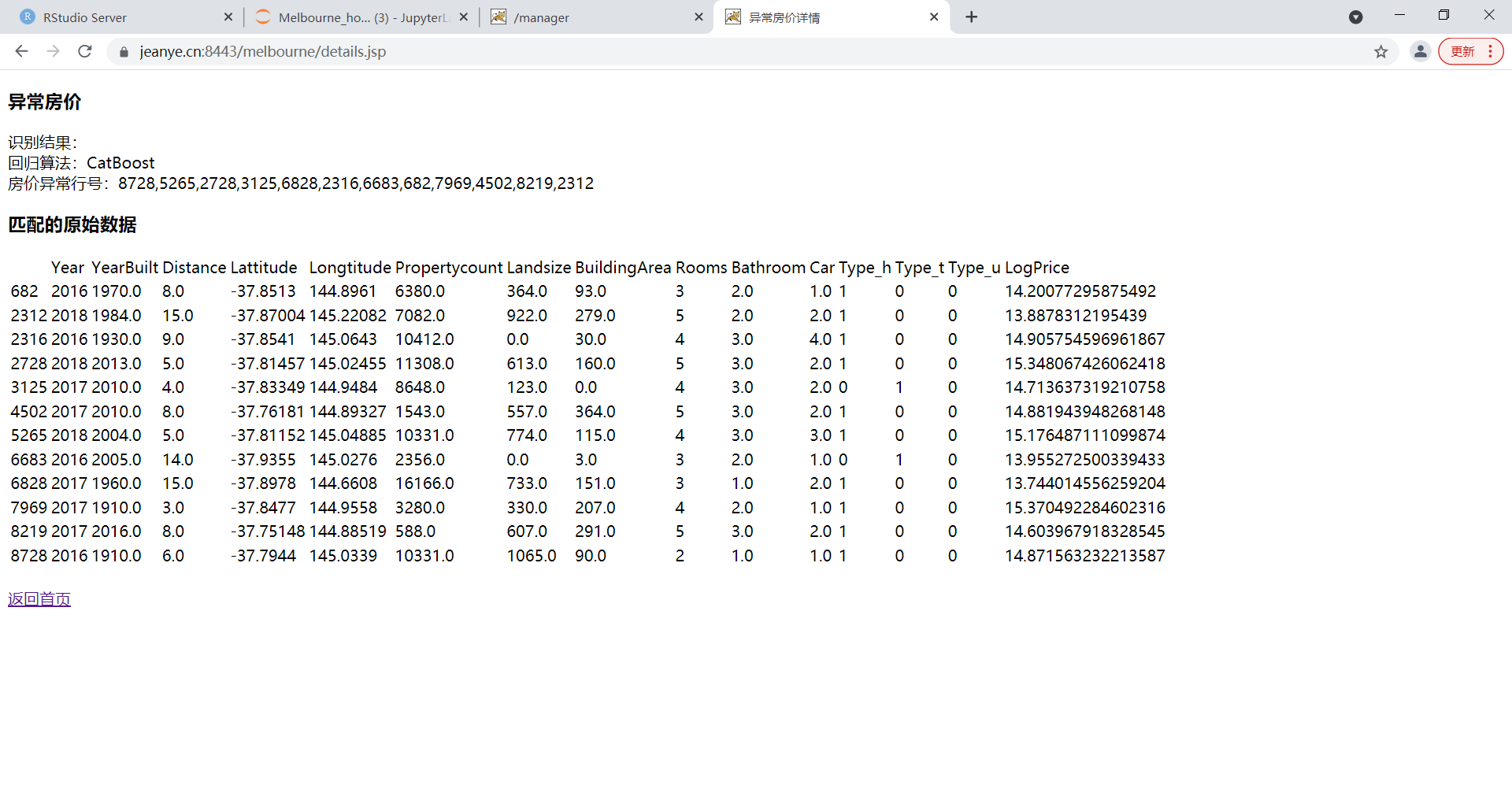
</table>

<iframe id="shinyappframe" src="https://jeanye.cn:4443/shiny/users/jean/melbourne/" style="border: 1px solid #AAA; width: 100%; height: 1000px"></iframe>

</body>

</html>

2、detail.jsp，演示业务流程中对Shiny APP数据分析结果的后续处理，演示性质这里只是简单匹配并显示原始数据。原始数据是经过预处理的CSV文件，有9千多行。根据Shiny APP交互式分析中选中感兴趣的数据行号，遍历原始数据选出，再格式化输出。



<%@ page language=*"java"* contentType=*"text/html; charset=UTF-8"* pageEncoding=*"UTF-8"*%>

<%@ page import=*"java.util.Set,java.util.HashSet,java.io.BufferedReader,java.io.FileReader"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"UTF-8"*>

<title>异常房价详情</title>

</head>

<body>

<h3>异常房价</h3>

<p>识别结果：<br>

回归算法：<%=**new** String(request.getParameter("algo").getBytes("ISO-8859-1"),"UTF-8") %><br>

房价异常行号：<%=**new** String(request.getParameter("outliers").getBytes("ISO-8859-1"),"UTF-8") %><br>

<h3>匹配的原始数据</h3>

<table>

<tr>

<%

String outliers[] = (**new** String(request.getParameter("outliers").getBytes("ISO-8859-1"),"UTF-8")).split(",");

Set<Integer> set = **new** HashSet<Integer>();

**for** (**int** i=0; i< outliers.length; i++){

set.add(Integer.parseInt(outliers[i]));

}

String path = request.getRealPath(".");

System.out.println(path);

BufferedReader reader = **new** BufferedReader(**new** FileReader(path+"/Melbourne\_housing\_pre.csv"));

// skip header

String header[] =reader.readLine().split(",");

**for** (**int** i=0; i<header.length;i++){

%>

<td><%=header[i]%></td>

<%}%></tr>

<%

String line = **null**;

**int** index=0;

//读取每行，直到为空

**while**((line=reader.readLine())!=**null**){

**if** (set.contains(index)){

String items[] = line.split(",");

%><tr><%

**for** (**int** i=0; i< items.length; i++){

%><td><%=items[i] %></td>

<%

}%></tr><%

}

index++;

}

%>

</table>

<br>

<a href=*"index.jsp"*>返回首页</a>

</body>

</html>

（二）Shiny APP

1.global.R

用各种回归算法模型的最优参数，一次过加载所有模型，训练好，拟合好（预测）训练集与验证集，并把各算法训练集与验证集预测的结果生成dataframe，以便后面在交互式的UI中快速反复引用。数据集不大，速度也很快，一般不到2分钟（取决于服务器的负荷）。各种回归算法模型的训练与调参当时是用Python做的，Python程序见后面。

library(reticulate)

library(ggplot2)

library(DT)

# 确定各回归模型Python脚本的目录位置

print(getwd())

location<- substr(getwd(),1,4)

# 调用Pyhton脚本运行各回归模型

if (location != '/srv'){ # Rstudio 开发环境

part<- '~'

}else{ # 服务器部署环境

part<- '..'

}

path<- paste(part,"/scripts/","Melbourne\_Regress.py",sep="")

print(path)

# 装入脚本，用经过预处理的数据集，在装入时一次性加载所有回归模型并完成训练集与验证集的拟合。

# 模型超参数调优另外用Python脚本完成。

# 这些脚本都可以用R语言完成，不过原来已经用Python写好了，通过reticulate包调用也可以，就不改了。

t3<- proc.time()

source\_python(path)

t4<- proc.time()

# 总耗时、系统耗时、用户耗时，...，...

# An object of class "proc\_time" which is a numeric vector of length 5,

# containing the user, system, and total elapsed times for the currently running R process,

# and the cumulative sum of user and system times of any child processes

# spawned by it on which it has waited.

seconds<-t4-t3

# 各模型训练与验证的结果，封装为data frame

train<- PredictTrain()

valid<- PredictValid()

# 各模型的性能数据

perf<- performance()

# 回归算法列表

algos<-names(train)

# 这个是显示在下拉列表中供选择的算法名称，作为命名列表的名字，它的内容是dataframe的列名。

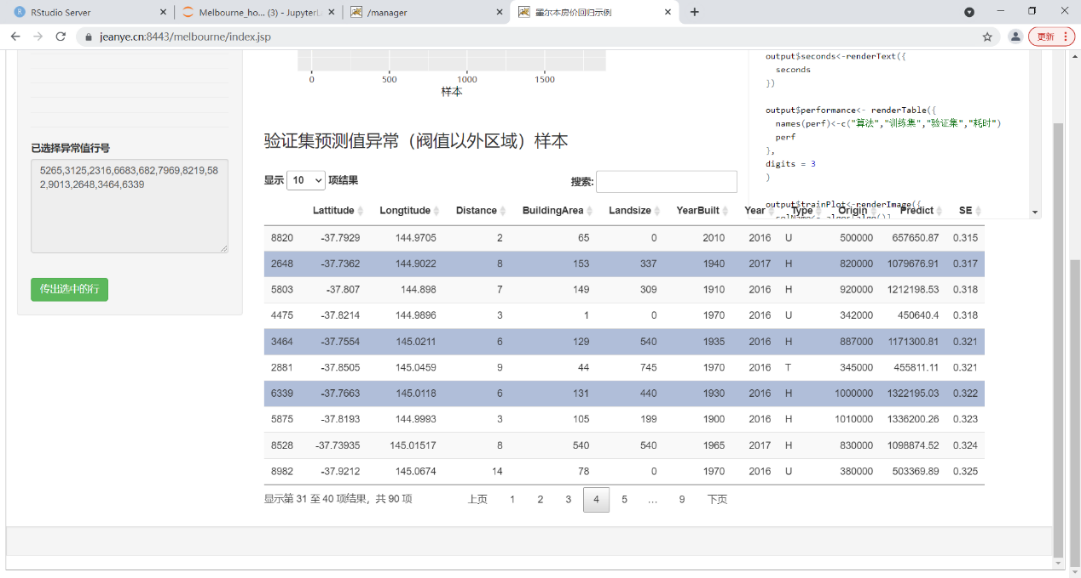
# 从第2项开始。

names(algos)<-c("Origin","SVM","RandomForest","GBR","XGB","LigthGBM","CatBoost","Blend")

2.ui.R

与前面的其它例子一样，也有与外层Tomcat Web APP通讯交换数据的javascript，主要的差别是引用的Shiny的Data Table UI Widget，可以翻页浏览dataframe，并且可以选定多行，以传递给外层的Tomcat Web APP，这个是比较典型的交互式操作。Shiny中对data table的操作都会记录在一组内置的reactive输入变量中，具体可参阅[Shiny的文档](https://rstudio.github.io/DT/shiny.html)。

输入变量algo定义了要使用的回归算法，异常值阀值threshold定义了从验证集预测值中过滤出异常值的偏差百分比，可以根据自己的需要设置，具体在server.R中引用并生成下图的异常值dataframe，以进行人工的案头分析。



# 浏览器端UI函数。

fluidPage(

# Javascript 处理父窗口传入的参数等。

tags$head(

tags$script(HTML("

// 记录父窗口，初始为空。

parent = null;

// 处理接收到的消息。

window.addEventListener('message', function(e) {

//alert(e.data);

try{

//记录父窗口以备回发信息

parent = e.source;

//向服务器发送input变量更新消息

Shiny.setInputValue('algo', e.data);

} catch (error){

alert(error);

}

},false);

// 点击时传出参数

$(document).on('click', '.btn-success', function (evt) {

evt.preventDefault();

var selected = document.getElementById('rows\_selected');

if (selected ==null || selected.value.length==0){

alert('没有选中的行!');

}else{

if (parent == null){

alert(selected.value);

} else {

try{

parent.postMessage(selected.value, '\*');

} catch(error){

alert(error);

}

}

}

});

"))

),

sidebarLayout(

# Sidebar with a selectioninput and a numericInput

sidebarPanel(

h3("墨尔本房价回归模型示例"),

selectInput(

'algo',

'回归算法',

names(algos)[2:8],

selected = 'CatBoost'

),

sliderInput("threshold",

"异常值阀值%：",

min = 20, max = 40, value = 30),

HTML("<br>总耗时<br>"),

textOutput("seconds"),

HTML("<br>模型性能<br>"),

tableOutput("performance"),

HTML("说明：这是对ln(房价)，即房价自然对数的回归，上表的ACC指标也是如此，因为ln(房价)接近正态分布，这样回归精度更高。<br>

exp(Predict)换算回原量纲后，回归精度略有下降，也接近88%。"),

HTML("<br><br><br><br><br><br><br><br><br><br><br><br>"),

textAreaInput("rows\_selected","已选择异常值行号", rows =6, resize="vertical", value =""),

# 插入javascript，禁止自己修改 rows\_selected textAreaInput

tags$script(HTML("

var rows\_selected = document.getElementById('rows\_selected');

rows\_selected.disabled = true;

")),

HTML("<br>"),

actionButton("sendout", "传出选中的行", class = "btn-success")

),

# Show the network

mainPanel(

# 画拟合效果

h3(textOutput("parameters")),

HTML("<br>"),

imageOutput("trainPlot"),

HTML("<br>"),

imageOutput("validPlot"),

HTML("<br><h3>验证集预测值异常（阀值以外区域）样本</h3><br>"),

dataTableOutput("outliers")

)

)

)

3、server.R

server.R中主要的处理是在renderImage()中，把ggplot作的图输出到SVG临时文件中，以支持中文标注，因为从源码安装R时选了Cairo而不是默认的TCL/TK后端图形支持。如果编译安装R时就选了TCL/TK图形支持，就不需要SVG临时文件，直接在renderImage()或renderPlot()中用ggplot就可以了，Windows上就是这样。

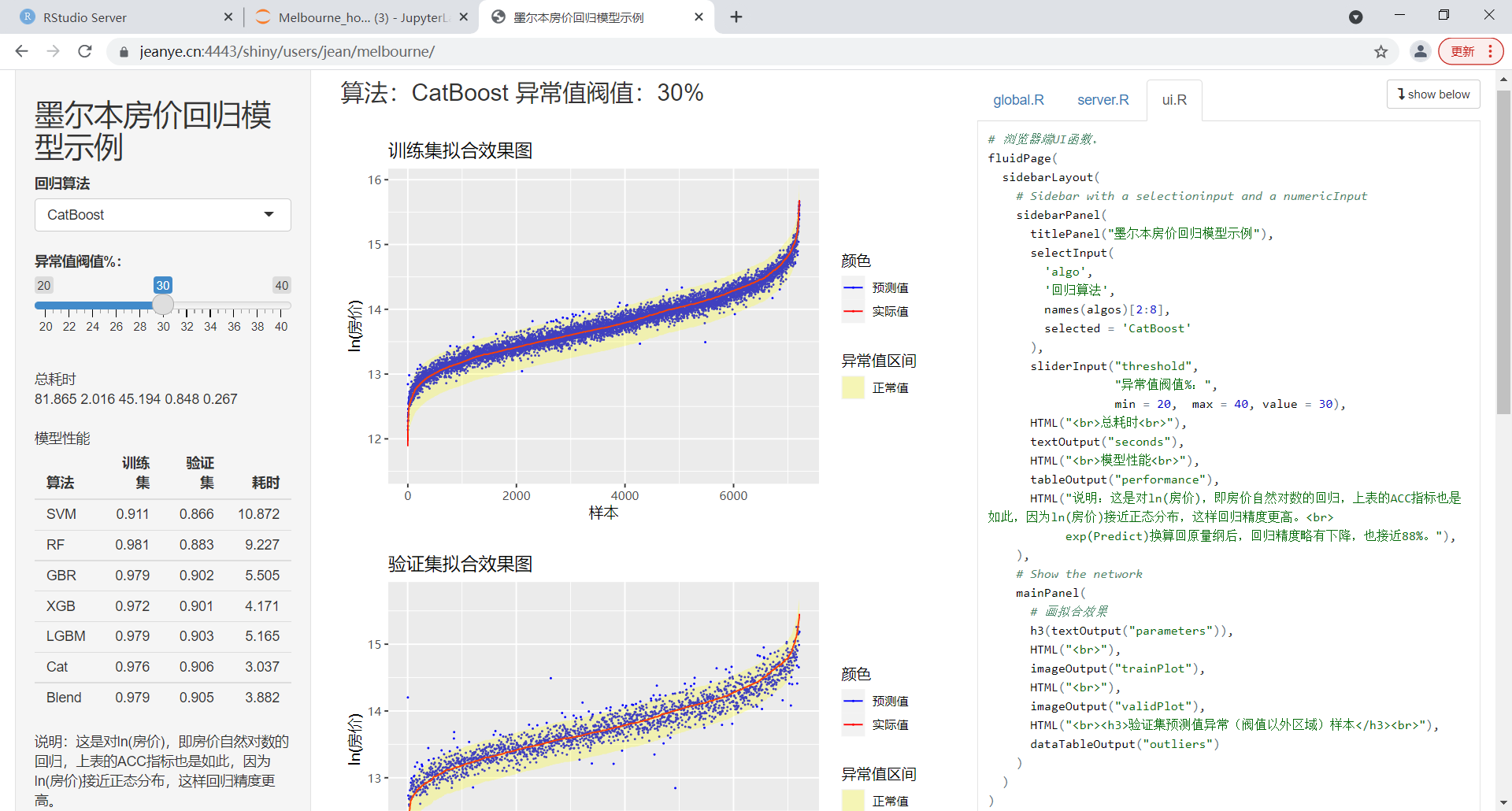
然后是Shiny Data Table菜单的中文支持，它的资源文件也可以在离线运行时从本地加载，不需要每次都联网加载。在APP目录下建立子目录"www"，把菜单语言资源文件Chinese.json拷贝进去，然后在renderDataTable()的options中设置语言url为下面即可：

language = list(url = 'Chinese.json')

Shiny Data Table支持多选，对它的操作会记录在input$下的一系列reactive输入变量中，可以引用，具体参阅[Shiny Data Table文档](https://rstudio.github.io/DT/shiny.html)，这里引用它的input$tabel\_ID\_rows\_selected，更新显示UI中的textAreaInput输入框，以便在UI中传出选中的行到外层的Tomcat Web APP。

注意Shiny Data Table中记录的选中行号是异常值dataframe outliers（行数较少）中的行号，而不是原始数据集（包含训练集与验证集）中的行号，这里通过把outliers存入session用户数据中，然后在更新UI中显示的选中行号时，把outliers的行号转换为行名来处理，行名就是原始数据中的行号，这样在外层Tomcat Web APP中就可以匹配到正确的原始数据。

另外也把房价用指数运算转换回原来的量纲，回归是对ln(房价)进行。



# 输出图形到SVG临时文件以支持中文标注

# https://stackoverflow.com/questions/39093777/renderimage-and-svg-in-shiny-app/39263320#39263320

# 多图层图例设置参阅

# https://stackoverflow.com/questions/18394391/custom-legend-for-multiple-layer-ggplot

# Shiny data table国际化语言支持参阅

# <https://rstudio.github.io/DT/004-i18n.html>

# Shiny Data Table的操作可以参考

# https://rstudio.github.io/DT/shiny.html

# 选中行的操作可以参考

# https://yihui.shinyapps.io/DT-rows/

shinyServer(function(input, output, session) {

algo<- reactive({

input$algo

})

threshold<- reactive({

input$threshold

})

# 更新由宿主页面传入的参数

observe({

print(algo())

# print(threshold())

# 更新由宿主页面传入的参数

updateTextInput(session, "algo", value = algo())

# 注意这里用updateTextInput更新文本到浏览器端，如果用updateSelectInput()会报错，可能要求的是下标。

# updateSelectInput(session, "algo", value = algo())

})

output$parameters<-renderText({

paste("算法：",algo()," 异常值阀值：",threshold(),"%",sep="")

})

output$seconds<-renderText({

seconds

})

output$performance<- renderTable({

names(perf)<-c("算法","训练集","验证集","耗时")

perf

},

digits = 3

)

output$trainPlot<-renderImage({

colName<- algos[algo()]

data<- data.frame(origin = train["origin"], predict = train[colName])

names(data)<-c("origin","predict")

data<-data[order(data["origin"]),]

data["index"]<-1:nrow(data)

data["upper"]<-data["origin"]+log(1+threshold()/100)

data["lower"]<-data["origin"]+log(1-threshold()/100)

image<-ggplot(data = data)+

geom\_point(mapping = aes(x = index, y = predict, color = "blue"), size=0.1)+

geom\_line(mapping = aes(x = index, y = origin, color = "red"))+

geom\_ribbon(aes(index, ymax=upper, ymin=lower, fill="yellow"), alpha=0.25)+

ggtitle("训练集拟合效果图")+

xlab("样本")+

ylab("ln(房价)")+

scale\_fill\_identity(name = '异常值区间', guide = 'legend',labels = c('正常值')) +

scale\_colour\_manual(name = '颜色',

values =c('blue'='blue','red'='red'), labels = c('预测值','实际值'))

# 输出SVG到临时文件

# This file will be removed later by renderImage

outfile <- tempfile(fileext='.svg')

#This actually save the plot in a image

# ggsave(file=outfile, plot=image, width=mysvgwidth, height=mysvgheight)

# 单位是英寸

ggsave(file=outfile, plot=image, width=6, height=4)

# Return a list containing the filename

list(src = normalizePath(outfile),

contentType = 'image/svg+xml',

# 单位是像素

#width = 400,

height = 400,

alt = "训练集拟合效果图")

},

# SVG图发到客户端后，renderImage删除临时文件

deleteFile = TRUE

)

output$validPlot<-renderImage({

colName<- algos[algo()]

data<- data.frame(origin = valid["origin"], predict = valid[colName])

names(data)<-c("origin","predict")

data<-data[order(data["origin"]),]

data["index"]<-1:nrow(data)

data["upper"]<-data["origin"]+log(1+threshold()/100)

data["lower"]<-data["origin"]+log(1-threshold()/100)

image<-ggplot(data = data)+

geom\_point(mapping = aes(x = index, y = predict, color = "blue"), size=0.1)+

geom\_line(mapping = aes(x = index, y = origin, color = "red"))+

geom\_ribbon(aes(index, ymax=upper, ymin=lower, fill="yellow"), alpha=0.25)+

ggtitle("验证集拟合效果图")+

xlab("样本")+

ylab("ln(房价)")+

scale\_fill\_identity(name = '异常值区间', guide = 'legend',labels = c('正常值')) +

scale\_colour\_manual(name = '颜色',

values =c('blue'='blue','red'='red'), labels = c('预测值','实际值'))

# 输出SVG到临时文件

# This file will be removed later by renderImage

outfile <- tempfile(fileext='.svg')

#This actually save the plot in a image

#ggsave(file=outfile, plot=image, width=mysvgwidth, height=mysvgheight)

ggsave(file=outfile, plot=image, width=6, height=4)

# Return a list containing the filename

list(src = normalizePath(outfile),

contentType = 'image/svg+xml',

#width = width,

height = 400,

alt = "验证集拟合效果图")

},

# SVG图发到客户端后，renderImage删除临时文件

deleteFile = TRUE

)

output$outliers<- renderDataTable({

colName<- algos[algo()]

data<- data.frame(origin = valid["origin"], predict = valid[colName])

names(data)<-c("origin","predict")

data["outlier"]<- FALSE

data[which(data["predict"] < data["origin"]+log(1-threshold()/100)),"outlier"] <-TRUE

data[which(data["predict"] > data["origin"]+log(1+threshold()/100)),"outlier"] <-TRUE

valid\_X3<- valid\_X2[,c("Lattitude","Longtitude","Distance","BuildingArea","Landsize","YearBuilt","Year")]

valid\_X3["Type"]<-valid\_X2["Type\_h"]\*4+valid\_X2["Type\_t"]\*2+valid\_X2["Type\_u"]

valid\_X3[which(valid\_X3["Type"]==4),"Type"]<-"H"

valid\_X3[which(valid\_X3["Type"]==2),"Type"]<-"T"

valid\_X3[which(valid\_X3["Type"]==1),"Type"]<-"U"

#valid\_X3["Origin"]<-data["origin"]

#valid\_X3["Predict"]<-data["predict"]

# 转换为原量纲

valid\_X3["Origin"]<-round(exp(data["origin"]),2)

valid\_X3["Predict"]<-round(exp(data["predict"]),2)

outliers<-valid\_X3[which(data["outlier"]==TRUE),]

outliers["SE"]<-round((outliers["Predict"] - outliers["Origin"])/outliers["Origin"],3)

#outliers["ID"]<- row.names(outliers)

outliers<-outliers[order(outliers["SE"]),]

# 放入 session 中以便后面选择行时引用，不能用行索引，它不是全局的， 行名才是全局的。

session$userData$outliers<- outliers

},

#rownames = FALSE,

options = list(

pageLength = 10,

#language = list(url = '//cdn.datatables.net/plug-ins/1.10.11/i18n/Chinese.json')

language = list(url = 'Chinese.json')

)

)

# 返回并显示选中行的行名，行名是全局的，用于定位对应的行，更新浏览器端显示。

observe({

outliers<- session$userData$outliers

rows<- outliers[input$outliers\_rows\_selected,]

outliersStr<- paste(row.names(rows), collapse=",")

updateTextAreaInput(session, "rows\_selected", value = outliersStr)

})

})

（三）Python回归算法模型

Python回归算法模型的迭代调试程序比较多，Shiny中直接引用各种算法的最优参数加载即可，其它相关的程序可以[从我的研究网站下载](http://jeanye.cn/melbourne/)，尤其是数据集的探索与预处理程序是比较重要的，贝叶斯优化程序也是比较重要的。不过要记住，在具体训练集上优化的超参数，必然是与具体的数据集相关的，人们只能期望它有较好的抗过拟合与泛化能力，在新的数据集上表现足够好，至于是否最好，那就未必了。

4、Melbourne\_Regress.py

这个程序从经过预处理的原始数据加载，共9千多条，房价已经转换为它的自然对数，因为转换后接近正态分布，回归算法的精度有较大的提高。然后80%划分为测试集共7212条，20%划分为验证集共1803条，这个样本量对训练与验证来说都是可以的。如果有需要，也可以分为训练集、验证集与测试集3个子集，后面可以看到，训练集上都有很高的准确率，验证集的作用是测试模型抗过拟合的性能，测试集则可以测试模型在新数据上的泛化性能。也可以在剩余的2万多条数据中选一些缺失项较少的来做测试集。

Blend算法是取RF、GBR、XGB、LightGBM、CatBoost五个算法各占20%的权重组合而成，属于水平堆叠，主要的作用是增强模型的抗过拟合能力与泛化能力，效果也不错。理论上选不同类型的算法效果要好一点，但在本数据集中，SVM等其它算法与各GBDT算法的准确率差距较大，就没有选。Stack层次堆叠回归算法因为速度太慢，精度上改善也不大，这里没有列入。在房价回归模型这个问题上，深度学习DNN回归的效果比不上上面的那些算法，可能是还没有找到有效的神经网络结构，这个方向暂时也没有继续研究。

最后，把各回归算法在训练集与验证集上的预测值，以及它们的性能数据，封装为dataframe，以便在R语言中直接引用，reticulate包可以直接转换dataframe这些基础数据类型。

# Imports

# Ignore Warnings

import warnings

warnings.filterwarnings('ignore')

# Basic Imports

import numpy as np

import pandas as pd

import time

# Preprocessing

from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score

# Metrics

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

# ML Models

from lightgbm import LGBMRegressor

from xgboost import XGBRegressor

from catboost import CatBoostRegressor

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn import svm

from mlxtend.regressor import StackingCVRegressor

# Reading a CSV File

# 9015

df\_NN = pd.read\_csv("~/data/Melbourne\_housing\_pre.csv", encoding="utf-8")

X=df\_NN[['Year','YearBuilt','Distance','Lattitude','Longtitude','Propertycount',

'Landsize','BuildingArea', 'Rooms','Bathroom', 'Car','Type\_h','Type\_t','Type\_u']]

y=df\_NN['LogPrice']

train\_X, valid\_X, train\_y, valid\_y = train\_test\_split(X,y, test\_size = .20, random\_state=42)

train\_X2 = train\_X.copy()

valid\_X2 = valid\_X.copy()

# Data standardization

mean = train\_X.mean(axis=0)

train\_X -= mean

std = train\_X.std(axis=0)

train\_X /= std

valid\_X -= mean

valid\_X /= std

##% evaluateRegressor

# from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

def evaluateRegressor(true,predicted,message = "Test set"):

R2 = r2\_score(true,predicted)

print(message)

print("R2 :", R2)

# 以下是调参后优化的性能，CatBoost在本数据集中的性能最优。

# CatBoost，LightGBM，GBR, XGB, Stack, Blend这些模型测试集中预测的准确率都超过了90%。

# 以下各字典用于记录各回归模型的性能与结果，以便在R语言中引用

train\_acc = {}

valid\_acc = {}

predict\_train = {}

predict\_valid = {}

timetake = {}

# SVM best--------------------------------------------------------------------------

t1 = time.time()

params = {'C': 6.673350889023755, 'gamma': 0.05106238973376298}

svr\_best = svm.SVR(\*\*params)

svr\_model\_best = svr\_best.fit(train\_X, train\_y)

train\_acc["SVM"] = svr\_model\_best.score(train\_X,train\_y)

valid\_acc["SVM"] = svr\_model\_best.score(valid\_X,valid\_y)

predict\_train["SVM"] = svr\_model\_best.predict(train\_X)

predict\_valid["SVM"] = svr\_model\_best.predict(valid\_X)

t2 = time.time()

timetake["SVM"] = t2-t1

print("SVM: ",t2-t1, "\n")

# Random forest best ---------------------------------------------------------------

t1 = time.time()

params = {'max\_depth': 26, 'max\_features': 4, 'max\_leaf\_nodes': 1781, 'max\_samples': 0.9703002612349153, 'min\_samples\_leaf': 0, 'min\_samples\_split': 0, 'n\_estimators': 381}

params['max\_depth'] = params['max\_depth']+3

params['min\_samples\_split'] = params['min\_samples\_split']+2

params['max\_leaf\_nodes'] = params['max\_leaf\_nodes']+2

params['min\_samples\_leaf'] = params['min\_samples\_leaf']+1

params['n\_estimators'] = params['n\_estimators']+50

params['max\_features'] = params['max\_features']+3

rf\_best = RandomForestRegressor( random\_state = 0,verbose=0, \*\*params)

rf\_model\_best = rf\_best.fit(train\_X, train\_y)

train\_acc["RF"] = rf\_model\_best.score(train\_X,train\_y)

valid\_acc["RF"] = rf\_model\_best.score(valid\_X,valid\_y)

predict\_train["RF"] = rf\_model\_best.predict(train\_X)

predict\_valid["RF"] = rf\_model\_best.predict(valid\_X)

t2 = time.time()

timetake["RF"] = t2-t1

print("Random forest: ",t2-t1, "\n")

# GBR best -------------------------------------------------------------------------

t1 = time.time()

params = {'alpha': 0.9014933457984278, 'learning\_rate': 0.035668343067947715, 'max\_depth': 6, 'max\_features': 5, 'max\_leaf\_nodes': 943, 'min\_impurity\_decrease': 0.015183480929538314, 'min\_samples\_leaf': 2, 'min\_samples\_split': 30, 'n\_estimators': 440, 'subsample': 0.6109167820106534}

params['n\_estimators'] = params['n\_estimators']+50

params['min\_samples\_split'] = params['min\_samples\_split']+2

params['min\_samples\_leaf'] = params['min\_samples\_leaf']+1

params['max\_depth'] = params['max\_depth']+3

params['max\_features'] = params['max\_features']+3

params['max\_leaf\_nodes'] = params['max\_leaf\_nodes']+2

gbr\_best = GradientBoostingRegressor(random\_state=0,verbose=0, \*\*params)

gbr\_model\_best = gbr\_best.fit(train\_X, train\_y)

train\_acc["GBR"] = gbr\_model\_best.score(train\_X,train\_y)

valid\_acc["GBR"] = gbr\_model\_best.score(valid\_X,valid\_y)

predict\_train["GBR"] = gbr\_model\_best.predict(train\_X)

predict\_valid["GBR"] = gbr\_model\_best.predict(valid\_X)

t2 = time.time()

timetake["GBR"] = t2-t1

print("GBR: ",t2-t1, "\n")

# XGB best -------------------------------------------------------------------------

t1 = time.time()

params = {'colsample\_bytree': 0.8413894273173292, 'gamma': 0.008478702584417519, 'learning\_rate': 0.05508679239402551, 'max\_bin': 4, 'max\_depth': 5, 'min\_child\_weight': 24.524635200338793, 'n\_estimators': 578, 'reg\_alpha': 0.809791155072757, 'reg\_lambda': 1.4490119256389808, 'subsample': 0.8429852720715357}

params['max\_bin'] = params['max\_bin']+50

params['max\_depth'] = params['max\_depth']+3

params['n\_estimators'] = params['n\_estimators']+100

xgb\_best = XGBRegressor(objective ='reg:squarederror', seed = 0,verbosity=0, \*\*params) # CPU 4.96s/trial

xgb\_model\_best = xgb\_best.fit(train\_X, train\_y)

train\_acc["XGB"] = xgb\_model\_best.score(train\_X,train\_y)

valid\_acc["XGB"] = xgb\_model\_best.score(valid\_X,valid\_y)

predict\_train["XGB"] = xgb\_model\_best.predict(train\_X)

predict\_valid["XGB"] = xgb\_model\_best.predict(valid\_X)

t2 = time.time()

timetake["XGB"] = t2-t1

print("XGB: ",t2-t1, "\n")

# LigthGBM best --------------------------------------------------------------------

t1 = time.time()

params = {'colsample\_bytree': 0.5142540541056978, 'learning\_rate': 0.014284678929509775, 'max\_bin': 161, 'max\_depth': 4, 'min\_child\_samples': 5, 'min\_child\_weight': 4.534457967283932, 'min\_split\_gain': 0.0006363777341674458, 'n\_estimators': 2006, 'num\_leaves': 93, 'reg\_alpha': 0.0037820689583625278, 'reg\_lambda': 2.947360470949046, 'subsample': 0.9448608935296047, 'subsample\_freq': 2}

params['max\_bin'] = params['max\_bin']+50

params['max\_depth'] = params['max\_depth']+3

params['num\_leaves'] = params['num\_leaves']+20

params['min\_child\_samples'] = params['min\_child\_samples']+10

params['subsample\_freq'] = params['subsample\_freq']+1

params['n\_estimators'] = params['n\_estimators']+1000

lgbm\_best = LGBMRegressor(seed=0, \*\*params)

lgb\_model\_best = lgbm\_best.fit(train\_X, train\_y)

train\_acc["LGBM"] = lgbm\_best.score(train\_X,train\_y)

valid\_acc["LGBM"] = lgbm\_best.score(valid\_X,valid\_y)

predict\_train["LGBM"] = lgbm\_best.predict(train\_X)

predict\_valid["LGBM"] = lgbm\_best.predict(valid\_X)

t2 = time.time()

timetake["LGBM"] = t2-t1

print("LightGBM: ",t2-t1, "\n")

# CatBoost best --------------------------------------------------------------------

t1 = time.time()

params = {'bagging\_temperature': 0.5402870554069704, 'border\_count': 183, 'depth': 5, 'fold\_len\_multiplier': 4.43906516804156, 'iterations': 899, 'l2\_leaf\_reg': 8.334167765336101, 'learning\_rate': 0.0997818676941431, 'random\_strength': 6.564979609549752, 'rsm': 0.8975065545697877, 'subsample': 0.857395221266925}

params['border\_count'] = params['border\_count']+150

params['depth'] = params['depth']+2

params['iterations'] = params['iterations']+500

cat\_best = CatBoostRegressor(task\_type='CPU',

random\_seed=0,

leaf\_estimation\_iterations=1,

max\_ctr\_complexity=0,

verbose=False, \*\*params) # CPU 44.64s/trial

cat\_model\_best = cat\_best.fit(train\_X, train\_y)

train\_acc["Cat"] = cat\_model\_best.score(train\_X,train\_y)

valid\_acc["Cat"] = cat\_model\_best.score(valid\_X,valid\_y)

predict\_train["Cat"] = cat\_model\_best.predict(train\_X)

predict\_valid["Cat"] = cat\_model\_best.predict(valid\_X)

t2 = time.time()

timetake["Cat"] = t2-t1

print("CatBoost: ",t2-t1, "\n")

# Blend models in order to make the final predictions more robust to overfitting

t1 = time.time()

def blended\_predictions(X):

return (

(0.20 \* cat\_model\_best.predict(X)) + \

(0.20 \* lgb\_model\_best.predict(X)) + \

(0.20 \* xgb\_model\_best.predict(X)) + \

(0.20 \* gbr\_model\_best.predict(X)) + \

(0.20 \* rf\_model\_best.predict(X)))

train\_acc["Blend"] = r2\_score(train\_y, blended\_predictions(train\_X))

valid\_acc["Blend"] = r2\_score(valid\_y, blended\_predictions(valid\_X))

predict\_train["Blend"] = blended\_predictions(train\_X)

predict\_valid["Blend"] = blended\_predictions(valid\_X)

t2 = time.time()

timetake["Blend"] = t2-t1

print("Blend: ",t2-t1, "\n")

def PredictTrain():

df = pd.DataFrame({"origin":train\_y, "svm":predict\_train["SVM"], "rf":predict\_train["RF"],\

"gbr":predict\_train["GBR"], "xgb":predict\_train["XGB"],"lgbm":predict\_train["LGBM"],\

"cat":predict\_train["Cat"],"blend":predict\_train["Blend"]})

return(df)

def PredictValid():

df = pd.DataFrame({"origin":valid\_y, "svm":predict\_valid["SVM"], "rf":predict\_valid["RF"],\

"gbr":predict\_valid["GBR"], "xgb":predict\_valid["XGB"],"lgbm":predict\_valid["LGBM"],\

"cat":predict\_valid["Cat"],"blend":predict\_valid["Blend"]})

return(df)

def performance():

df = pd.DataFrame({"algo":list(train\_acc.keys()),"train":list(train\_acc.values()), \

"valid":list(valid\_acc.values()), "time":list(timetake.values())})

return(df)

第六节 小结

前面几节了解了Shiny平台的基本框架，介绍了它独特的reactive编程模式，了解了Shiny APP开发的基本过程。解决了Shiny APP嵌入传统WEB界面系统以及通过R语言与Python连接云端等各种不同的数据源的落地应用集成问题。这样Shiny平台就具备了非常强大的能力和变化无穷的灵活性，可以适应任何的应用场景，并为扩展传统WEB界面系统的功能，盘活存量的软硬件和数据资产提供了简便易行的解决方案。

1、数据层通过R语言或Python连接各种后端系统。

2、中间层充分发挥R语言或Python的深度数据分析能力，如机器学习与深度学习建模。

3、应用层无缝嵌入各种传统的WEB界面系统，扩充系统功能。

4、WEB客户端可以适配手机、平板、笔记本、PC等各种大小屏，适应面非常广。

从这些小例子可以看到Shiny平台有它独特的优势，是大数据集成应用中非常优秀的数据中台，不管是直接使用还是参考研发相关的产品，都是有很价值的。这是个通用的开源解决方案，对于相关各方，比如各厂商，肯定是构成压力的，希望大家可以把压力转换为动力，做出更好的产品。Shiny集成各厂商产品的能力，提供了一条超越内卷的途径，让用户可以超越平台及语言的内卷，按自己的需要和习惯去驾驭大数据，从而把精力集中在业务和分析上，最有效率的做事。这就是我选择它作为解决方案来封装发布本书的各种深度分析实例分析结果的原因。

今天我们开发建设大数据平台，需要超越十几年来数据仓库BI的层次，对标国际水准，向深度分析拓展，这样才是有竞争力立得住的。

更多的Shiny APP，可以看它的示例[Gallery](https://shiny.rstudio.com/gallery/),都是开源，参考或修改使用都可以。安装配置可以参考《附录一 软件与数据》之《六、Shiny Server安装配置》。