Dashboards com R

Módulos

Outubro de 2021

Problema

Conforme o nosso aplicativo cresce, fica cada vez mais difícil manter o seu código em um único arquivo. Imaginem corrigir um errinho de digitação no título de um `ggplot` em um arquivo com 10 `ggplots` diferentes e mais de 5000 linhas... Cada alteração nesse arquivo vai exigir um `CTRL+F`.

Além disso, conforme cresce o número de inputs e outputs, garantir que seus IDs são únicos se torna uma tarefa morosa e muito fácil de gerar erros.

Por que não apenas usar funções?

A nossa experiência com programação em R nos diria separar o código do app em vários arquivos, transformando partes da UI e do server em objetos ou funções. Assim, bastaria colocar esses arquivos dentro de uma pasta chamada `R/` ou fazer `source("arquivo_auxiliar.R")` para cada arquivo auxiliar no início do código.

O problema é que essa solução resolve o problema do tamanho do script, mas ainda assim precisaríamos nos preocupar com a unicidade dos IDs dos inputs e outputs.

Módulos

Módulos são uma alternativa para gerenciar a complexidade de aplicativos Shiny muito grandes que resolve o problema do tamanho dos scripts e da unicidade dos IDs.

Na prática, módulos são uma **adaptação do uso de funções** que respeita a lógica UI/server de aplicativos Shiny.

Isso é feito garantindo-se que os IDs de cada módulo são únicos.

Estrutura de um módulo

O primeiro conceito que precisamos guardar é que **módulos são funções**. Então, todas as regras válidas para a criação de uma função, valem para a criação de módulos.

O segundo conceito fala sobre como enxergamos os módulos na prática. Cada módulo será uma parte do nosso aplicativo, um mini Shiny app que não pode ser rodado isoladamente. Quando modularizamos um aplicativo, o resultado final será um app composto por diversos pequenos aplicativos que funcionam conjuntamente. Assim, todo módulo possui uma UI e (opcionalmente) um `server`.

O terceiro conceito diz respeito à unicidade dos IDs. Para cada módulo criado precisamos passar um `id`. No desenvolvimento do app, precisamos garantir que o `id` de cada módulo seja único e que os `inputId` e `outputId` dentro de cada módulo sejam únicos e estejam dentro da função `ns()`. Fazendo isso, garantimos a unicidade dos IDs entre módulos, isto é, no app como um todo.

O código de um módulo

```
histograma_ui <- function(id) {
  ns <- NS(id)
  tagList(
    selectInput(
      ns("variavel_x"),
      "Selecione uma variável",
      choices = names(mtcars)
    ),
    br(),
    plotOutput(ns("grafico"))
  )
}

histograma_server <- function(id) {
  moduleServer(id, function(input, output, session) {
    output$grafico <- renderPlot({
      hist(mtcars[[input$variavel_x]])
    })
  })
}
```

A UI de um módulo

A UI de um módulo é apenas uma função de R. Essa função deve receber um `id` e devolver código HTML (um objeto com classe `shiny.tag.list`). A única diferença para a construção usual do objeto `ui` que vimos até agora é a utilização da função `ns()`. Essa função é criada pelo código `ns <- NS(id)` e deve ser usada para embrulhar todos os `inputId` e `outputId` presentes no módulo.

A função `NS()` cria uma função que cola o `id` passado para o módulo no começo de cada `inputId` e `outputId` dentro do módulo. Como o `id` de cada módulo é único, isso garante a unicidade dos IDs entre módulos.

```
ns <- shiny::NS("meu_modulo")
ns("id_do_input")
```

```
## [1] "meu_modulo-id_do_input"
```

O server de um módulo

Assim como a UI, o servidor também é uma função que recebe um `id`. A diferença é que essa função deve retornar a chamada da função `moduleServer()`.

A função `moduleServer()` recebe como primeiro argumento o `id` e como segundo a nossa função `server` habitual, isto é, a declaração de uma função com os argumentos `input`, `output` e `session` e que possua toda a lógica do servidor.

Observações

- Um módulo grande o suficiente pode (e deve) ser dividido em módulos menores, isto é, você pode criar módulos dentro de módulos.
- Módulos são parametrizáveis e podem ser utilizados diversas vezes dentro de um mesmo app. Você pode passar quantos parâmetros quiser (além do parâmetro `id`) para a UI e server de um módulo.
- A UI e o servidor de um módulo não consegue acessar objetos, inputs, outputs ou valores reativos de outros módulos. Para acessar valores da `ui` ou da função `server`, cada valor deve ser passado explicitamente aos módulos como argumentos das funções.
- Toda função UI de um módulo deve começar com `ns <- NS(id)`.
- É uma boa prática nomear igualmente as funções da UI e do servidor, trocando apenas o sufixo "ui" ou "server".

Atividade

Vamos construir um aplicativo Shiny modularizado.

Ao RStudio: 21-modulos-1/app.R e 22-modulos-2/app.R

Referências e material extra

- Artigo sobre módulos da RStudio