RStudio
250 Northern Ave, Boston, MA 02210
Phone: 844-448-1212
Email: info@rstudio.com
Web: http://www.rstudio.com

Understanding Shiny Modules

How to write big apps

Garrett Grolemund

April 2016

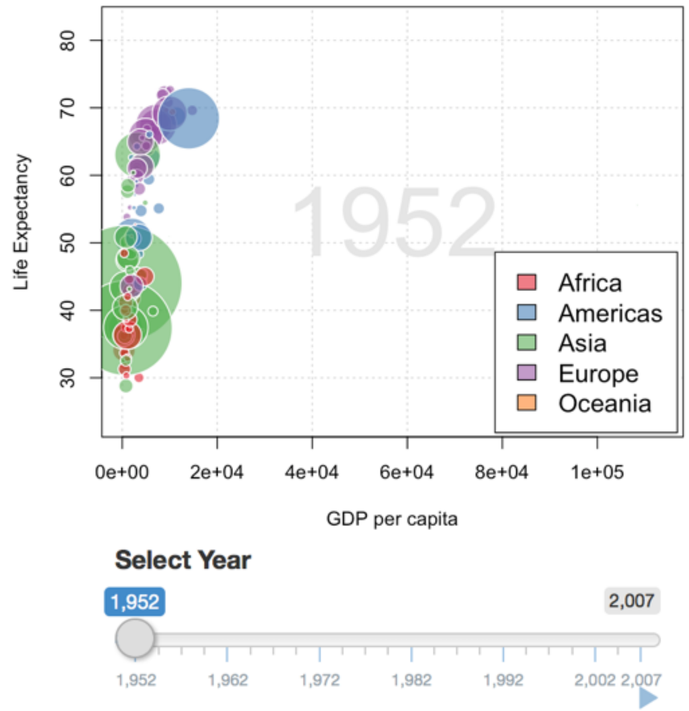
What is a module?

Conceptually: A self-contained, composable component of a Shiny App.

DEMO

What is a module?

Conceptually: A self-contained, composable component of a Shiny App.



Why use modules?

Reuse

Quickly reuse the same code in different apps, or multiple times in the same app.

Isolate

Divide code into separate modules that can be reasoned about independently.

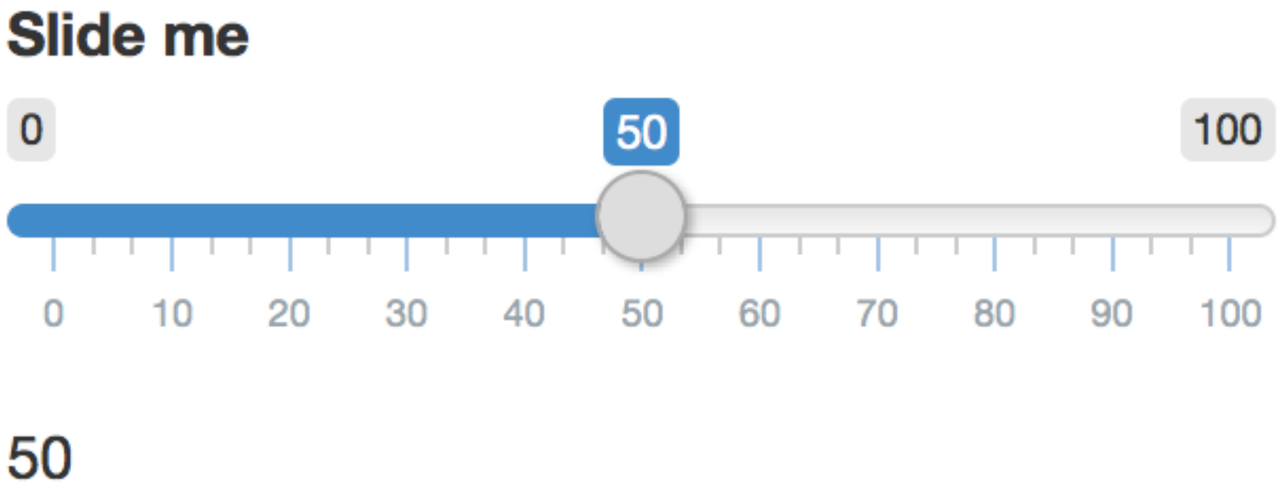

https://www.flickr.com/photos/23178876@N03/3920473828/ CC BY-SA
reactive spaghetti

Modules Pattern

What is a module?

Logistically: A pattern of code

- A function that creates UI elements
- A function that loads server logic
- shiny.rstudio.com/articles/modules.html




```
sliderTextUI <- function(){ #UI }
sliderText <- function(){ #Server logic }
```

Naming practice

A shared root that describes the module

Suffix the ui function with UI, Input, or Output

```
sliderTextUI <- function(){ #UI }
sliderText <- function(){ #Server logic }
```

Naming practice

A shared root that describes the module

Suffix the ui function with UI, Input, or Output

```
sliderTextUI <- function(){ #UI }
sliderText <- function(){ #Server logic }
```

Modules
UI

```
ui <- fluidPage(
  sliderInput("slider", "Slide Me", 0, 100, 1),
  textOutput("num")
)

server <- function(input, output) {
  output$num <- renderText({
    input$slider
  })
}

shinyApp(ui, server)
```

Will this work?

```
sliderTextUI <- function() {
  tagList(
    sliderInput("slider", "Slide Me", 0, 100, 1),
    textOutput("num")
  )
}
ui <- fluidPage(
  sliderTextUI()
)
server <- function(input, output) {
  output$num <- renderText({input$slider})
}
shinyApp(ui, server)
```

Will this work?

```
sliderTextUI <- function() {
  tagList(
    sliderInput("slider", "Slide Me", 0, 100, 1),
    textOutput("num")
  )
}
ui <- fluidPage(
  sliderTextUI(),
  sliderTextUI()
)
server <- function(input, output) {
  output$num <- renderText({input$slider})
  output$num <- renderText({input$slider})
}
shinyApp(ui, server)
```

Will this work?

```
sliderTextUI <- function(slideId, textId) {
  tagList(
    sliderInput(slideId, "Slide Me", 0, 100, 1),
    textOutput(textId)
  )
}
ui <- fluidPage(
  sliderTextUI("slider1", "num1"),
  sliderTextUI("slider2", "num2")
)
server <- function(input, output) {
  output$num1 <- renderText({input$slider1})
  output$num2 <- renderText({input$slider2})
}
shinyApp(ui, server)
```

Will this work?

NS( )

Adds a prefix to an id to create a "namespace"

```
NS("hello")
## function (id)
## {
##     paste(c(namespace, id), collapse = ns.sep)
## }
## <environment: 0x1034976e8>
```

NS( )

Adds a prefix to an id to create a "namespace"

```
ns <- NS("hello")

ns("world")
## [1] "hello-world"
ns("y'all")
## Error: Don't you mean "Howdy"?
```

NS( )

Adds a prefix to an id to create a "namespace"

```
ns <- NS("hello")

ns("world")
## [1] "hello-world"
ns("y'all")
## [1] "hello-y'all"
```

NS( )

Adds a prefix to an id to create a "namespace"

```
ns <- NS("howdy")

ns("world")
## [1] "howdy-world"
ns("y'all")
## [1] "howdy-y'all"
```

Module UI

Task 1 - Return Shiny UI.

```
sliderTextUI <- function(){


    sliderInput("slider", "Slide me", 0, 100, 1)
    textOutput("number")

}
```

Module UI

Task 1 - Return Shiny UI. Wrap multiple elements in `tagList()`

```
sliderTextUI <- function(){

  tagList(
    sliderInput("slider", "Slide me", 0, 100, 1),
    textOutput("number")
  )
}
```

Module UI

Task 2 - Assign module elements to a unique namespace with `NS()`

```
sliderTextUI <- function(id){
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide me", 0, 100, 50, 1),
    textOutput(ns("number"))
  )
}
```

1. Add an id argument
2. Make a namespace function
3. Wrap all inputId's and outputId's

Module UI

Task 2 - Assign module elements to a unique namespace with `NS()`

```
sliderTextUI <- function(id){
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide me", 0, 100, 1),
    textOutput(ns("number"))
  )
}
```

1. Add an id argument
2. Make a namespace function
3. Wrap all inputId's and outputId's

How to use Module UI

Call the module UI function where you want the UI elements to go. Supply a unique ID.

```
ui <- fluidPage(
  sliderTextUI("hello")
)

server <- function(input, output) {}

shinyApp(ui, server)
```

How to use Module UI

Call the module UI function where you want the UI elements to go. Supply a unique ID.

```
ui <- fluidPage(
  sliderTextUI("hello"),
  sliderTextUI("howdy")
)

server <- function(input, output) {}

shinyApp(ui, server)
```

Modules
Server

Module server

Handles the server logic for the module. Very similar to a Shiny app server function.

```
sliderText <- function(input, output, session){
  output$num <- renderText({
    input$slider
  })
}
```

Module server

Handles the server logic for the module. Very similar to a Shiny app server function.

1. You must use all 3 arguments: input, output, session

```
sliderText <- function(input, output, session){
  output$num <- renderText({
    input$slider
  })
}
```

Module server

Handles the server logic for the module. Very similar to a Shiny app server function.

1. You must use all 3 arguments: input, output, session

```
sliderText <- function(input, output, session){
  output$num <- renderText({
    input$slider
  })
}
```

2. Do **not** use ns() to refer to inputs and outputs from the module

How to use Module Server

Load the module server function in your app's server function with `callModule()`

```
ui <- fluidPage(
  sliderTextUI("hello")
)
server <- function(input, output) {
  callModule(sliderText, "hello")
}
shinyApp(ui, server)
```

How to use Module Server

Load the module server function in your app's server function with `callModule()`

```
ui <- fluidPage(
  sliderTextUI("hello")
)
server <- function(input, output) {
  callModule(sliderText, "hello")
}
shinyApp(ui, server)
```

1 Call from within server function

How to use Module Server

Load the module server function in your app's server function with `callModule()`

```
ui <- fluidPage(
  sliderTextUI("hello")
)
server <- function(input, output) {
  callModule(sliderText, "hello")
}
shinyApp(ui, server)
```

1. Call from within server function
2. 1st argument = module function

How to use Module Server

Load the module server function in your app's server function with `callModule()`

```
ui <- fluidPage(
  sliderTextUI("hello")
)
server <- function(input, output) {
  callModule(sliderText, "hello")
}
shinyApp(ui, server)
```

1. Call from within server function
2. 1st argument = module function
3. 2nd argument = same id as UI

```
sliderTextUI <- function(id) {
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide Me", 0, 100, 1),
    textOutput(ns("num"))
  )
}
sliderText <- function(input, output, session) {
  output$num <- renderText({input$slider})
}
```

```
sliderTextUI <- function(id) {
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide Me", 0, 100, 1),
    textOutput(ns("num"))
  )
}
sliderText <- function(input, output, session) {
  output$num <- renderText({input$slider})
}
```

```
sliderTextUI <- function(id) {
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide Me", 0, 100, 1),
    textOutput(ns("num"))
  )
}
sliderText <- function(input, output, session) {
  output$num <- renderText({input$slider})
}
```

```
sliderTextUI <- function(id) {
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide Me", 0, 100, 1),
    textOutput(ns("num"))
  )
}
sliderText <- function(input, output, session) {
  output$num <- renderText({input$slider})
}
ui <- fluidPage(
  sliderTextUI("hello"),
  sliderTextUI("howdy")
)
server <- function(input, output) {
  callModule(sliderText, "hello")
  callModule(sliderText, "howdy")
}
shinyApp(ui, server)
```

```
sliderTextUI <- function(id) {
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide Me", 0, 100, 1),
    textOutput(ns("num"))
  )
}
sliderText <- function(input, output, session) {
  output$num <- renderText({input$slider})
}
ui <- fluidPage(
  sliderTextUI("hello"),
  sliderTextUI("howdy")
)
server <- function(input, output) {
  callModule(sliderText, "hello")
  callModule(sliderText, "howdy")
}
shinyApp(ui, server)
```


This will work

```
sliderTextUI <- function(id) {
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide Me", 0, 100, 1),
    textOutput(ns("num"))
  )
}
sliderText <- function(input, output, session) {
  output$num <- renderText({input$slider})
}
ui <- fluidPage(
  sliderTextUI("hello"),
  sliderTextUI("howdy")
)
server <- function(input, output) {
  callModule(sliderText, "hello")
  callModule(sliderText, "howdy")
}
shinyApp(ui, server)
```


This will work

```
sliderTextUI <- function(id) {
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide Me", 0, 100, 1),
    textOutput(ns("num"))
  )
}
sliderText <- function(input, output, session) {
  output$num <- renderText({input$slider})
}
ui <- fluidPage(
  sliderTextUI("hello"),
  sliderTextUI("howdy")
)
server <- function(input, output) {
  callModule(sliderText, "hello")
  callModule(sliderText, "howdy")
}
shinyApp(ui, server)
```

This will work

```
sliderTextUI <- function(id) {
  ns <- NS(id)
  tagList(
    sliderInput(ns("slider"), "Slide Me", 0, 100, 1),
    textOutput(ns("num"))
  )
}
sliderText <- function(input, output, session) {
  output$num <- renderText({input$slider})
}
ui <- fluidPage(
  sliderTextUI("hello"),
  sliderTextUI("howdy")
)
server <- function(input, output) {
  callModule(sliderText, "hello")
  callModule(sliderText, "howdy")
}
shinyApp(ui, server)
```


This will work

Where to define the module functions?

1. In the preamble of a single file app (app.R)
2. In a file that is sourced in the preamble of a single file app (app.R)
3. In global.R
4. In a file sourced by global.R
5. In a package that the app loads

Reusing modules

Give the module a unique id each time you call it.

```
ui <- fluidPage(
  sliderTextUI("first"),
  sliderTextUI("second"),
  sliderTextUI("third")
)
server <- function(input, output) {
  callModule(sliderText, "first")
  callModule(sliderText, "second")
  callModule(sliderText, "third")
}
shinyApp(ui, server)
```

Communicate with the app

To make modules Modular:

1. Pass all inputs into the module as arguments of a module function

2. Return all outputs as the return value of a module function

Arguments

Module functions are functions: you can use arguments.

```
sliderTextUI <- function(id, label = "Slide me"){
  ns <- NS(id)

  tagList(
    sliderInput(ns("slider"), label, 0, 100, 1),
    textOutput(ns("number"))
  )
}
```

Arguments

callModule passes extra arguments in order to module server function

```
sliderText <- function(input, output, session, add) {
  output$num <- renderText({input$slider + add})
}

server <- function(input, output) {
  callModule(sliderText, "hello", add = 5)
}
```

Return Values

callModule returns anything returned by the module server function

```
sliderText <- function(input, output, session, add) {
  output$num <- renderText({input$slider + add})
  "hi"
}

server <- function(input, output) {
  hi <- callModule(sliderText, "hello", add = 5)
}
```

How to pass reactive values?

Principle?

Reactive expressions are the most portable format for passing reactive information between functions

```
num <- reactive({input$number})
```

`num()` - value **(reactive)**

`num` - reference **(not reactive)**

To pass **reactive input** to a module

```
sliderText<-function(input,output,session,show){
  output$number <- renderText({
    if (show()) input$slider
    else NULL
  })
}
ui <- fluidPage(
  checkboxInput("display", "Show Value"),
  sliderTextUI("module")
)
server <- function(input, output) {
  display <- reactive({ input$display })
  callModule(sliderText, "module", display)
}
shinyApp(ui, server)
```

To pass **reactive input** to a module

```
sliderText<-function(input,output,session,show){
  output$number <- renderText({
    if (show()) input$slider
    else NULL
  })
}
ui <- fluidPage(
  checkboxInput("display", "Show Value"),
  sliderTextUI("module")
)
server <- function(input, output) {
  display <- reactive({ input$display })
  callModule(sliderText, "module", display)
}
shinyApp(ui, server)
```

To pass **reactive input** to a module

```
sliderText<-function(input,output,session,show){
  output$number <- renderText({
    if (show()) input$slider
    else NULL
  })
}
ui <- fluidPage(
  checkboxInput("display", "Show Value"),
  sliderTextUI("module")
)
server <- function(input, output) {
  display <- reactive({ input$display })
  callModule(sliderText, "module", display)
}
shinyApp(ui, server)
```

To pass **reactive input** to a module

```
sliderText<-function(input,output,session,show){
  output$number <- renderText({
    if (show()) input$slider
    else NULL
  })
}
ui <- fluidPage(
  checkboxInput("display", "Show Value"),
  sliderTextUI("module")
)
server <- function(input, output) {
  display <- reactive({ input$display }) 1
  callModule(sliderText, "module", display)
}
shinyApp(ui, server)
```

1. Wrap the input as a **reactive expression**, e.g.

```
foo <- reactive({ rv$foo })
```

To pass **reactive input** to a module

```
sliderText<-function(input,output,session,show){
  output$number <- renderText({
    if (show()) input$slider
    else NULL
  })
}
ui <- fluidPage(
  checkboxInput("display", "Show Value"),
  sliderTextUI("module")
)
server <- function(input, output) {
  display <- reactive({ input$display }) 1
  callModule(sliderText, "module", display) 2
}
shinyApp(ui, server)
```

1. Wrap the input as a reactive expression, e.g.
   `foo <- reactive({ rv$foo })`

2. Pass the **reactive expression**, not the value, to the module, i.e
   do NOT use ( )'s.

To pass **reactive input** to a module

```
sliderText<-function(input,output,session,show){
  output$number <- renderText({
    if (show()) input$slider 3
    else NULL
  })
}
ui <- fluidPage(
  checkboxInput("display", "Show Value"),
  sliderTextUI("module")
)
server <- function(input, output) {
  display <- reactive({ input$display }) 1
  callModule(sliderText, "module", display) 2
}
shinyApp(ui, server)
```

1. Wrap the input as a reactive expression, e.g.
   `foo <- reactive({ rv$foo })`

2. Pass the **reactive expression**, not the value, to the module, i.e
   do NOT use ( )'s.

3. Treat the argument as a **reactive expression** within the module, i.e. *do use ( )'s*.

To return **reactive output** from a module

```
sliderText <- function(input, output, session){
  output$num <- renderText({input$slider})
  reactive({input$slider})
}

ui <- fluidPage(
  sliderTextUI("module"),
  h2(textOutput("value"))
)
server <- function(input, output) {
  num <- callModule(sliderText, "module")

  output$value <- renderText({ num() })
}
shinyApp(ui, server)
```

To return **reactive output** from a module

```
sliderText <- function(input, output, session){
  output$num <- renderText({input$slider})
  reactive({input$slider}) ❶
}

ui <- fluidPage(
  sliderTextUI("module"),
  h2(textOutput("value"))
)
server <- function(input, output) {
  num <- callModule(sliderText, "module")

  output$value <- renderText({ num() })
}
shinyApp(ui, server)
```

❶ Return reactive output as a **reactive expression** or a list of reactive expressions

To return **reactive output** from a module

```
sliderText <- function(input, output, session){
  output$num <- renderText({input$slider})
  reactive({input$slider}) 1
}

ui <- fluidPage(
  sliderTextUI("module"),
  h2(textOutput("value"))
)
server <- function(input, output) {
  num <- callModule(sliderText, "module")

  output$value <- renderText({ num() }) 2
}
shinyApp(ui, server)
```

1. Return reactive output as a **reactive expression** or a list of reactive expressions

2. Call value as a **reactive expression**, i.e. with ( )'s.

Thank you

Read more at

shiny.rstudio.com/articles/modules.html