R Shiny Package: What is ‘Reactivity’?

Section 3: Isolation

Section 3: Isolation and Avoiding Dependency

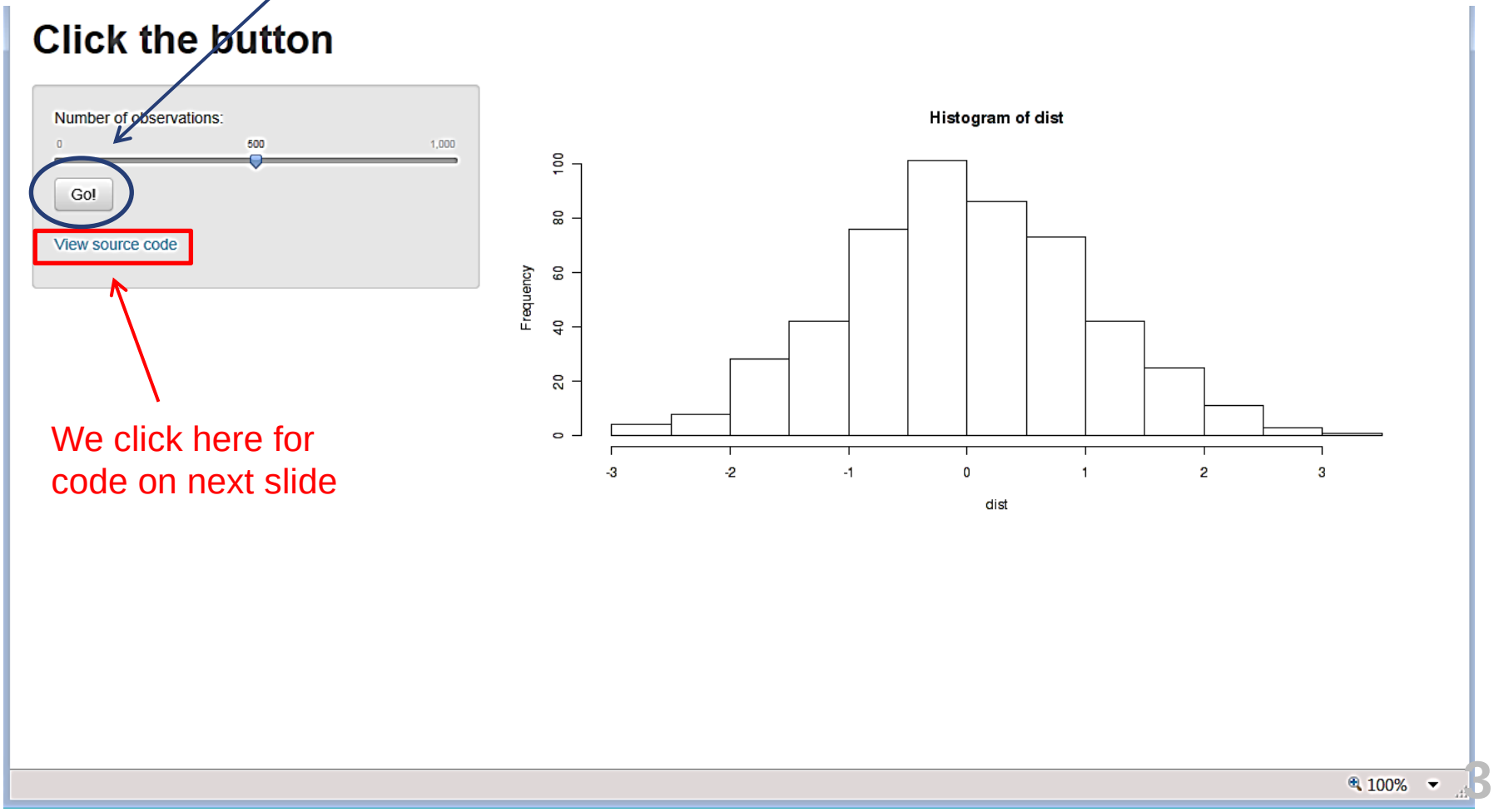
Sometimes it's useful for an observer/endpoint to access a reactive value or expression, but not to take a dependency on it. For example, if the observer performs a long calculation or downloads large data set, you might want it to execute only when a button is clicked.

For this, we'll use `actionButton`. We'll define a `ui.R` that is a slight modification of the one from 01_hello – the only difference is that it has an actionButton labeled "Go!". You can see it in action at http://glimmer.rstudio.com/winston/actionbutton/.

The actionButton includes some JavaScript code that sends numbers to the server. When the web browser first connects, it sends a value of 0, and on each click, it sends an incremented value: 1, 2, 3, and so on.

```
shinyUI(pageWithSidebar(
  headerPanel("Click the button"),
  sidebarPanel(
    sliderInput("obs", "Number of observations:",
                min = 0, max = 1000, value = 500),
    actionButton("goButton", "Go!")
  ),
  mainPanel(
    plotOutput("distPlot")
  )
))
```

Actionbutton live screen



actionbutton code

server.r

```r
shinyServer(function(input, output) {
  output$distPlot <- renderPlot({
    # Take a dependency on input$goButton
    input$goButton

    # Use isolate() to avoid dependency on input$obs
    dist <- isolate(rnorm(input$obs))
    hist(dist)
  })
})
```

ui.r

```r
library(shinyIncubator)

shinyUI(pageWithSidebar(
  headerPanel("Click the button"),
  sidebarPanel(
    sliderInput("obs", "Number of observations:",
                min = 0, max = 1000, value = 500),
    actionButton("goButton", "Go!"),
    p(br(), a("View source code", href="https://gist.github.com/wch/4963887"))
  ),
  mainPanel(
    plotOutput("distPlot")
  )
))
```

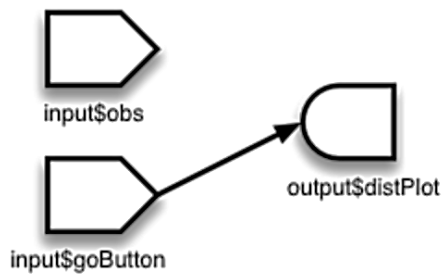
Isolation: Avoiding Dependency

In our `server.R`, there are two changes to note. First, `output$distPlot` will take a dependency on `input$goButton`, simply by accessing it. When the button is clicked, the value of `input$goButton` increases, and so `output$distPlot` re-executes.

The second change is that the access to `input$obs` is wrapped with `isolate()`. This function takes an R expression, and it tells Shiny that the calling observer or reactive expression should not take a dependency on any reactive objects inside the expression.

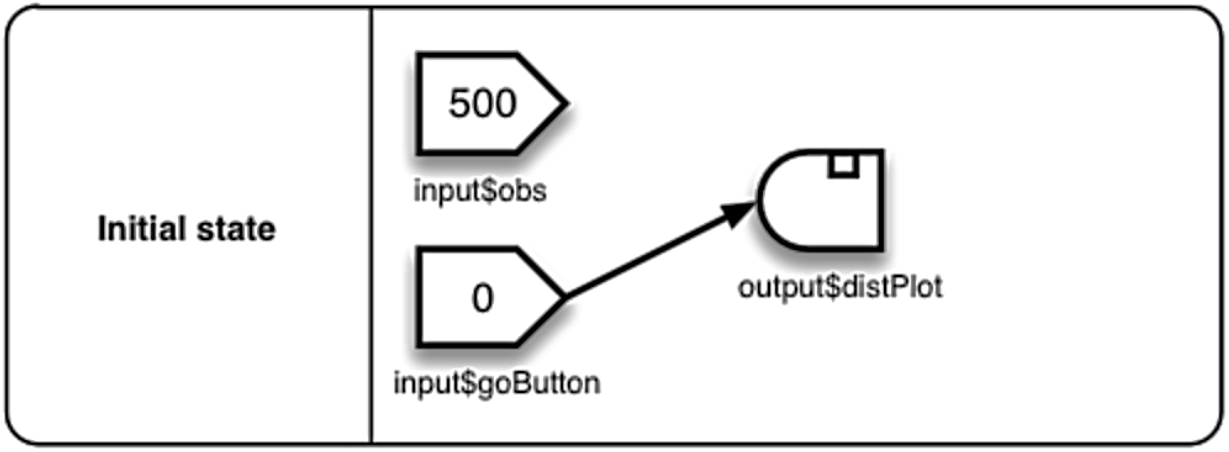
```
shinyServer(function(input, output) {
  output$distPlot <- renderPlot({
    # Take a dependency on input$goButton
    input$goButton

    # Use isolate() to avoid dependency on input$obs
    dist <- isolate(rnorm(input$obs))
    hist(dist)
  })
})
```

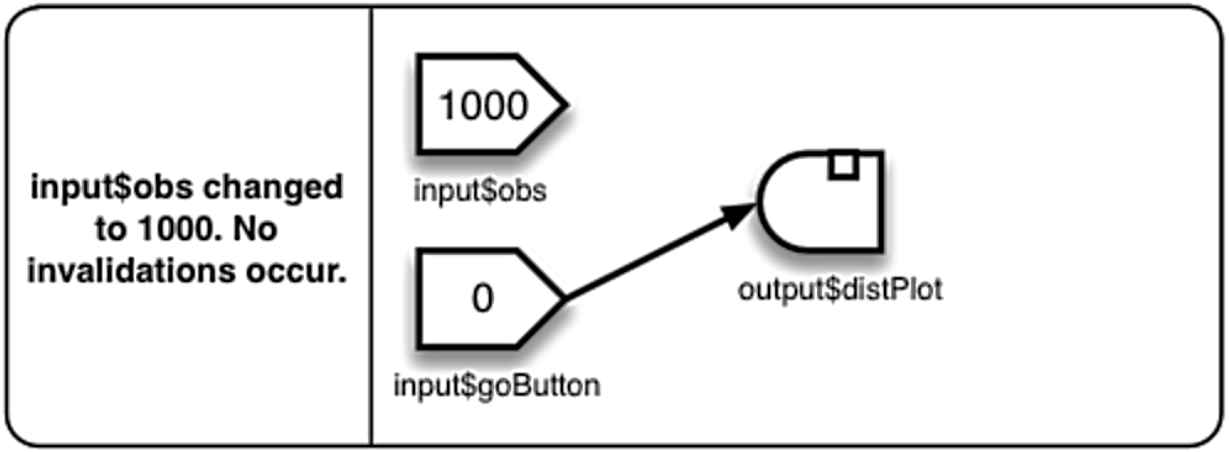
The resulting graph looks like this:



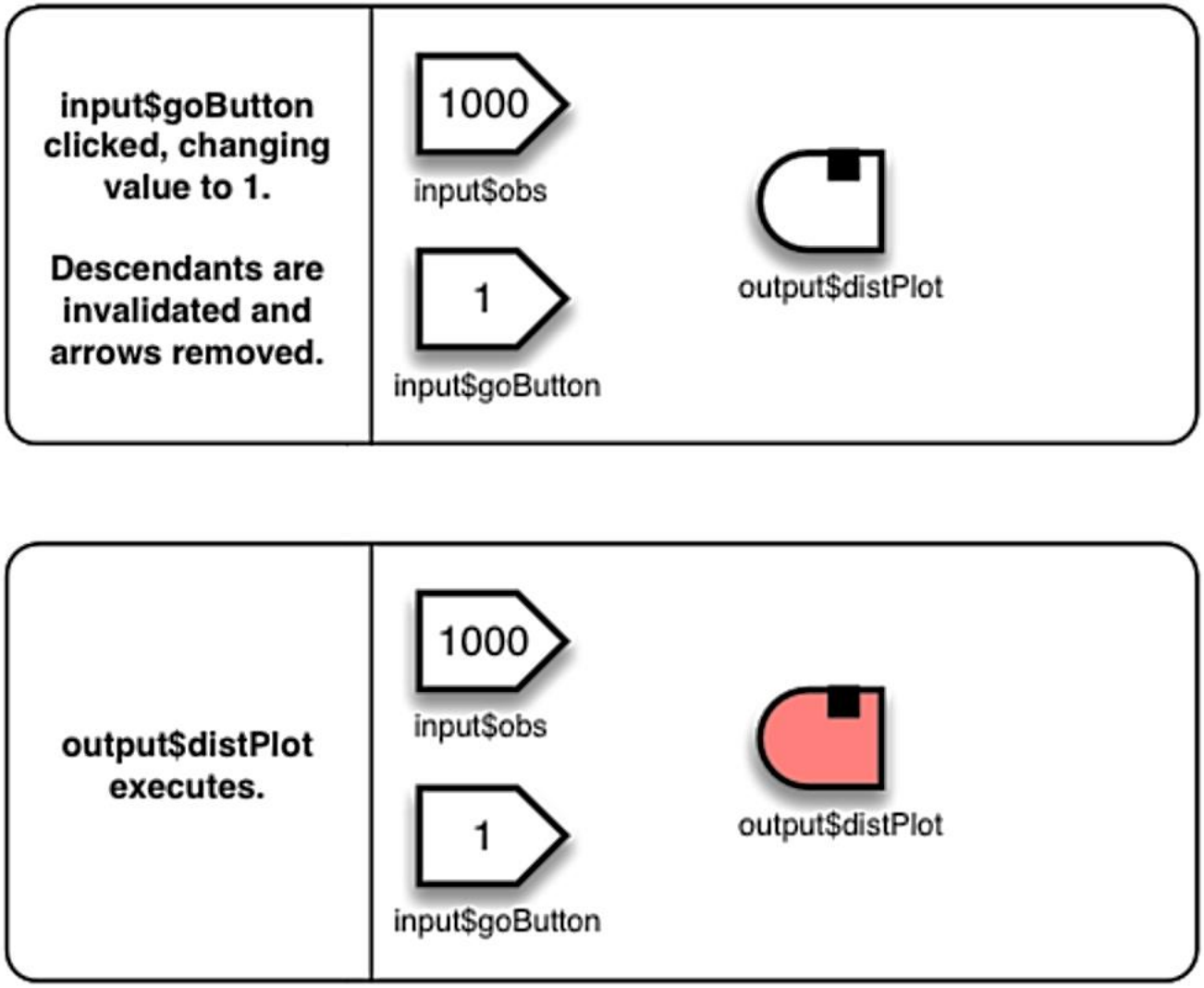
Isolation: Avoiding Dependency

And here's a walkthrough of the process when `input$obs` is set to 1000, and then the Go button is clicked:

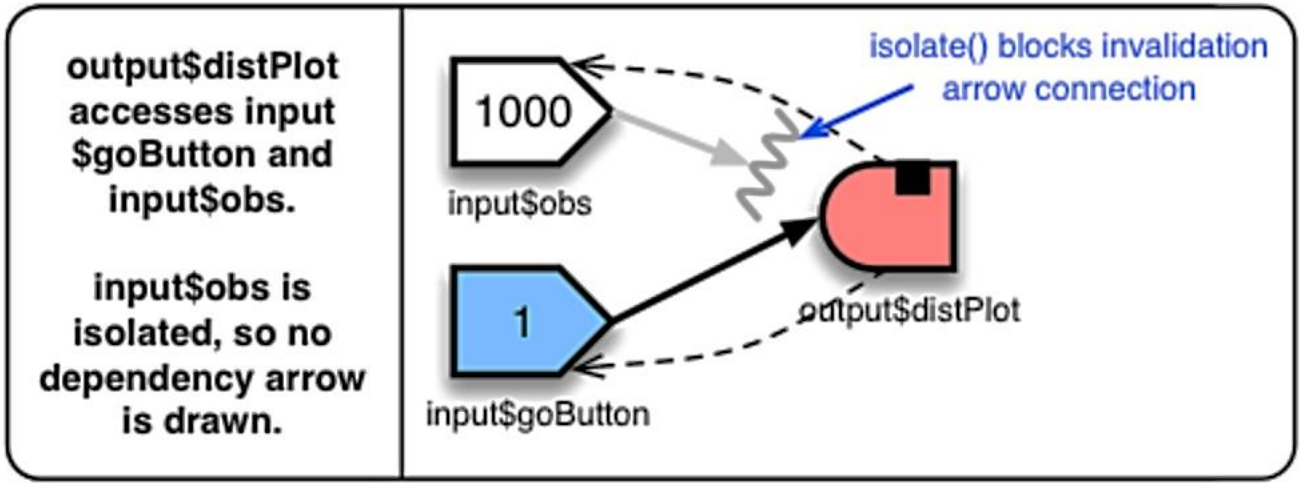


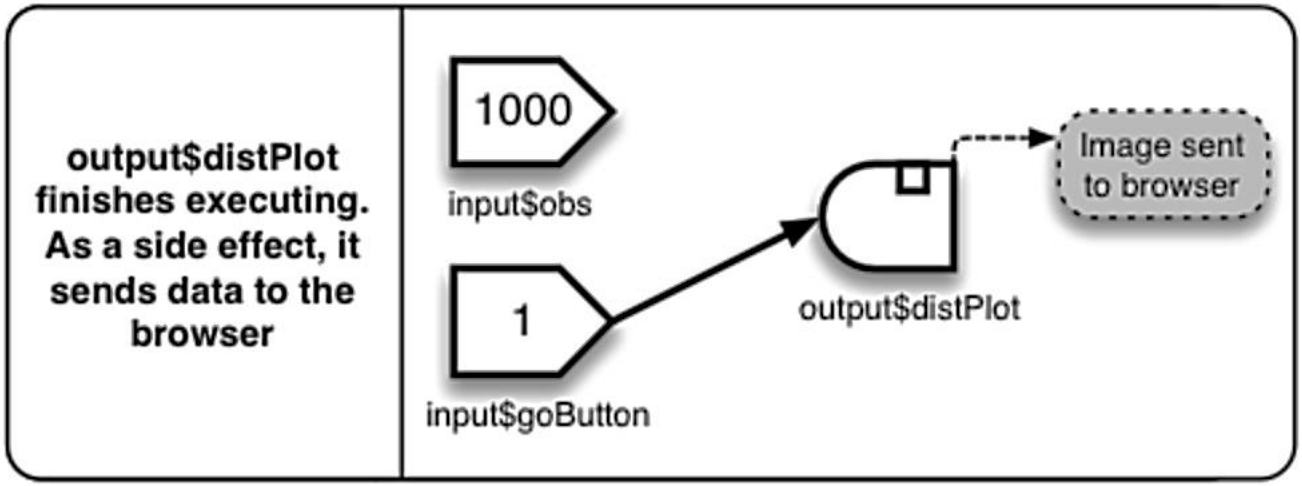


Isolation: Avoiding Dependency



Isolation: Avoiding Dependency





Isolation: Avoiding Dependency

In the `actionButton` example, you might want to prevent it from returning a plot the first time, before the button has been clicked. Since the starting value of an `actionButton` is zero, this can be accomplished with the following:

```
output$distPlot <- renderPlot({
  if (input$goButton == 0)
    return()

  # plot-making code here
})
```

Reactive values are not the only things that can be isolated; reactive expressions can also be put inside an `isolate()`. Building off the Fibonacci example from above, this would calculate the *n*th value only when the button is clicked:

```
output$nthValue <- renderText({
  if (input$goButton == 0)
    return()

  isolate({ fib(as.numeric(input$n)) })
})
```

Isolation: Avoiding Dependency

It's also possible to put multiple lines of code in `isolate()`. For example here are some blocks of code that have equivalent effect:

```
# Separate calls to isolate --------------------------------
x <- isolate({ input$xslider }) + 100
y <- isolate({ input$yslider })  * 2
z <- x/y

# Single call to isolate -----------------------------------
isolate({
  x <- input$xslider + 100
  y <- input$yslider * 2
  z <- x/y
})

# Single call to isolate, use return value -----------------
z <- isolate({
  x <- input$xslider + 100
  y <- input$yslider * 2
  x/y
})
```


In all of these cases, the calling function won't take a reactive dependency on either of the `input` variables.