Scoping

Where you define objects will determine where the objects are visible. There are three different levels of visibility that you'll want to be aware of when writing Shiny apps. Some objects are visible within the `server.R` code of each user session; other objects are visible in the `server.R` code across all sessions (multiple users could use a shared variable); and yet others are visible in the `server.R` and the `ui.R` code across all user sessions.

Per-session objects

In `server.R`, when you call `shinyServer()`, you pass it a function `func` which takes two arguments, `input` and `output`:

```
shinyServer(func = function(input, output) {
  # Server code here
  # ...
})
```

The function that you pass to `shinyServer()` is called once for each session. In other words, `func` is called each time a web browser is pointed to the Shiny application.

Everything within this function is instantiated separately for each session. This includes the `input` and `output` objects that are passed to it: each session has its own `input` and `output` objects, visible within this function.

Other objects inside the function, such as variables and functions, are also instantiated for each session. In this example, each session will have its own variable named `startTime`, which records the start time for the session:

```
shinyServer(function(input, output) {
  startTime <- Sys.time()

  # ...
})
```

Objects visible across all sessions

You might want some objects to be visible across all sessions. For example, if you have large data structures, or if you have utility functions that are not reactive (ones that don't involve the `input` or `output` objects), then you can create these objects once and share them across all user sessions, by placing them in `server.R`, but outside of the call to `shinyServer()`.

For example:

```
# A read-only data set that will load once, when Shiny starts, and will be
# available to each user session
bigDataSet <- read.csv('bigdata.csv')

# A non-reactive function that will be available to each user session
utilityFunction <- function(x) {
  # Function code here
  # ...
}

shinyServer(function(input, output) {
  # Server code here
  # ...
})
```

You could put `bigDataSet` and `utilityFunction` inside of the function passed to `shinyServer()`, but doing so will be less efficient, because they will be created each time a user connects.

If the objects change, then the changed objects will be visible in every user session. But note that you would need to use the `<<-` assignment operator to change `bigDataSet`, because the `<-` operator only assigns values in the local environment.

```
varA <- 1
varB <- 1
listA <- list(X=1, Y=2)
listB <- list(X=1, Y=2)

shinyServer(function(input, output) {
  # Create a local variable varA, which will be a copy of the shared variable
  # varA plus 1. This local copy of varA is not be visible in other sessions.
  varA <- varA + 1

  # Modify the shared variable varB. It will be visible in other sessions.
  varB <<- varB + 1

  # Makes a local copy of listA
  listA$X <- 5

  # Modify the shared copy of listB
  listB$X <<- 5

  # ...
})
```

Things work this way because `server.R` is sourced when you start your Shiny app. Everything in the script is run immediately, including the call to `shinyServer()`—but the function which is passed to `shinyServer()` is called only when a web browser connects and a new session is started.

Global objects

Objects defined in `global.R` are similar to those defined in `server.R` outside `shinyServer()`, with one important difference: they are also visible to the code in `ui.R`. This is because they are loaded into the global environment of the R session; all R code in a Shiny app is run in the global environment or a child of it.

In practice, there aren't many times where it's necessary to share variables between `server.R` and `ui.R`. The code in `ui.R` is run once, when the Shiny app is started and it generates an HTML file which is cached and sent to each web browser that connects. This may be useful for setting some shared configuration options.

Scope for included R files

If you want to split the server or ui code into multiple files, you can use `source(local=TRUE)` to load each file. You can think of this as putting the code in-line, so the code from the sourced files will receive the same scope as if you copied and pasted the text right there.

This example `server.R` file shows how sourced files will be scoped:

```
# Objects in this file are shared across all sessions
source('all_sessions.R', local=TRUE)

shinyServer(function(input, output) {
  # Objects in this file are defined in each session
  source('each_session.R', local=TRUE)

  output$text <- renderText({
    # Objects in this file are defined each time this function is called
    source('each_call.R', local=TRUE)

    # ...
  })
})
```

If you use the default value of `local=FALSE`, then the file will be sourced in the global environment.

← Previous   Next →