This tutorial is deprecated. Learn more about Shiny at our new location, shiny.rstudio.com.

Dynamic UI

Shiny apps are often more than just a fixed set of controls that affect a fixed set of outputs. Inputs may need to be shown or hidden depending on the state of another input, or input controls may need to be created on-the-fly in response to user input.

Shiny currently has three different approaches you can use to make your interfaces more dynamic. From easiest to most difficult, they are:

- **The `conditionalPanel` function**, which is used in `ui.R` and wraps a set of UI elements that need to be dynamically shown/hidden
- **The `renderUI` function**, which is used in `server.R` in conjunction with the `htmlOutput` function in `ui.R`, lets you generate calls to UI functions and make the results appear in a predetermined place in the UI
- **Use JavaScript** to modify the webpage directly.

Let's take a closer look at each approach.

Showing and Hiding Controls With `conditionalPanel`

`conditionalPanel` creates a panel that shows and hides its contents depending on the value of a JavaScript expression. Even if you don't know any JavaScript, simple comparison or equality operations are extremely easy to do, as they look a lot like R (and many other programming languages).

Here's an example for adding an optional smoother to a ggplot, and choosing its smoothing method:

```
# Partial example
checkboxInput("smooth", "Smooth"),
conditionalPanel(
  condition = "input.smooth == true",
  selectInput("smoothMethod", "Method",
              list("lm", "glm", "gam", "loess", "rlm"))
)
```

In this example, the select control for `smoothMethod` will appear only when the `smooth` checkbox is checked. Its condition is `"input.smooth == true"`, which is a JavaScript expression that will be evaluated whenever any inputs/outputs change.

The condition can also use `output` values; they work in the same way (`output.foo` gives you the value of the output `foo`). If you have a situation where you wish you could use an R expression as your `condition` argument, you can create a reactive expression in `server.R` and assign it to a new output, then refer to that output in your `condition` expression. For example:

ui.R

```
# Partial example
selectInput("dataset", "Dataset", c("diamonds", "rock", "pressure", "cars")),
conditionalPanel(
  condition = "output.nrows",
  checkboxInput("headonly", "Only use first 1000 rows"))
```

server.R

```
# Partial example
datasetInput <- reactive({
   switch(input$dataset,
          "rock" = rock,
          "pressure" = pressure,
          "cars" = cars)
})

output$nrows <- reactive({
  nrow(datasetInput())
})
```

However, since this technique requires server-side calculation (which could take a long time, depending on what other reactive expressions are executing) we recommend that you avoid using `output` in your conditions unless absolutely necessary.

Creating Controls On the Fly With `renderUI`

Note: This feature should be considered experimental. Let us know whether you find it useful.

Sometimes it's just not enough to show and hide a fixed set of controls. Imagine prompting the user for a latitude/longitude, then allowing the user to select from a checklist of cities within a certain radius. In this case, you can use the `renderUI` expression to dynamically create controls based on the user's input.

ui.R

```
# Partial example
numericInput("lat", "Latitude"),
numericInput("long", "Longitude"),
uiOutput("cityControls")
```

server.R

```
# Partial example
output$cityControls <- renderUI({
  cities <- getNearestCities(input$lat, input$long)
  checkboxGroupInput("cities", "Choose Cities", cities)
})
```

`renderUI` works just like `renderPlot`, `renderText`, and the other output rendering functions you've seen before, but it expects the expression it wraps to return an HTML tag (or a list of HTML tags, using `tagList`). These tags can include inputs and outputs.

In `ui.R`, use a `uiOutput` to tell Shiny where these controls should be rendered.

Use JavaScript to Modify the Page

Note: This feature should be considered experimental. Let us know whether you find it useful.

You can use JavaScript/jQuery to modify the page directly. General instructions for doing so are outside the scope of this tutorial, except to mention an important additional requirement. Each time you add new inputs/outputs to the DOM, or remove existing inputs/outputs from the DOM, you need to tell Shiny. Our current recommendation is:

- Before making changes to the DOM that may include adding or removing Shiny inputs or outputs, call `Shiny.unbindAll()`.
- After such changes, call `Shiny.bindAll()`.

If you are adding or removing many inputs/outputs at once, it's fine to call `Shiny.unbindAll()` once at the beginning and `Shiny.bindAll()` at the end – it's not necessary to put these calls around each individual addition or removal of inputs/outputs.

← Previous Next →