

packages

To be ready for later, please have installed:

CRAN

`shiny`, `shinyjs`, `shinyBS`, `dplyr`, `htmltools`, `lubridate`,
`stringr`, `dygraphs`, `xts`

Github

```
install.packages("devtools")
devtools::install_github("hadley/readr")
devtools::install_github("ijlyttle/shinychord")
```

shinychord

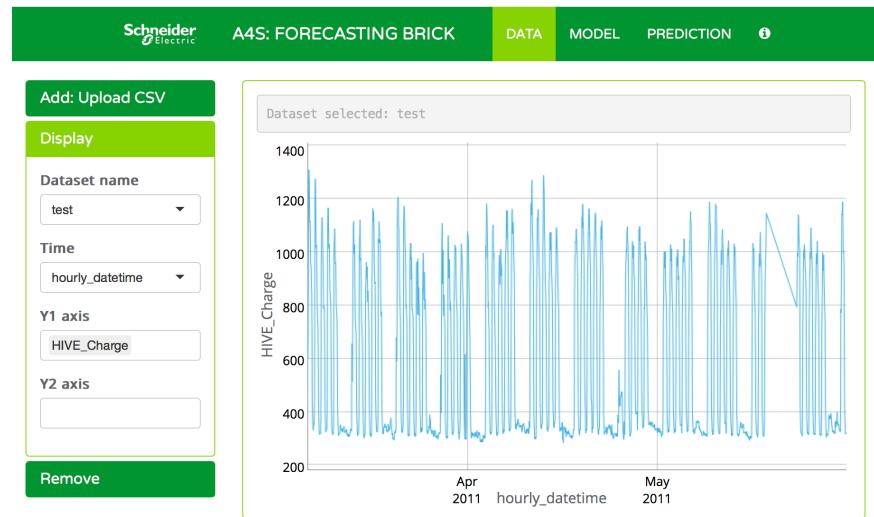
modular shiny code to reduce reactive spaghetti

Ian Lyttle

Thanks: Alex Shum, Andee Kaplan, Eric Hare, Joe Cheng, Garrett Grolemund

how this came about

- This summer, Alex Shum built a complex shiny app.
- The app (thanks to me) always needed “one more thing”.
- Each time I asked for “one more thing”, I got a look of pain.
- However, Alex would figure out how to do it, and get it done.



here's the problem

Part of your shiny app makes a dygraph from a dataframe.

The rest of your shiny app does stuff to that dataframe.

Issues:

- You have to keep track of all the id's of all the elements (including the rest of the app).
- You have to manage the server logic (including the server logic for the rest of the app).
- It's not easy to copy-and-paste the parser from this app into a different app.



reactive spaghetti

inspiration!

 **Andee Kaplan**
@andeekaplan



Slides for our talk this morning on intRo, software for introductory statistics at #JSM2015 @ericrhare [#rstats](http://j.mp/intRo-JSM2015)

RETWEETS 10 FAVORITES 19 

10:49 AM - 11 Aug 2015 

Andee and Eric's slides:
<http://j.mp/intRo-JSM2015>

my first response:
<https://gist.github.com/ijlyttle/1237cd57d55a40bb47a1>

 **Ian Lyttle** @ijlyttle · Aug 11
@andeekaplan @ericrhare Resuable shiny-server code - modules - is fascinating. Been banging my head for a while on this. Coffee in Ames?


 **Eric Hare** @ericrhare · Aug 11
@ijlyttle @andeekaplan definitely!


 **Ian Lyttle** @ijlyttle · Aug 11
@ericrhare @andeekaplan Excellent! To be clear, all I have so far is a sore head :/ - looking forward to talking more!

[View other replies](#)

 **Andee Kaplan** @andeekaplan · Aug 11
@ijlyttle @ericrhare great idea! Pick the time and place, looking forward to it.

[View other replies](#)

 **Ian Lyttle** @ijlyttle · Aug 12
@andeekaplan @ericrhare Here's what I have so far. It works for *this* case; I suspect it could be made more robust: gist.github.com/ijlyttle/1237cd57d55a40bb47a1

[View summary](#)

 **Ian Lyttle** @ijlyttle · Aug 12
@andeekaplan @ericrhare Is this sort-of what you are after, in terms of modularity? If nothing else, a conversation starter for coffee.


a solution

Let's borrow some ideas from the design pattern:

model-view-controller

In shiny, these correspond:

- ui inputs - **controller**
- ui outputs - **view**
- server() - **model**

controller

Add: Upload CSV

Display

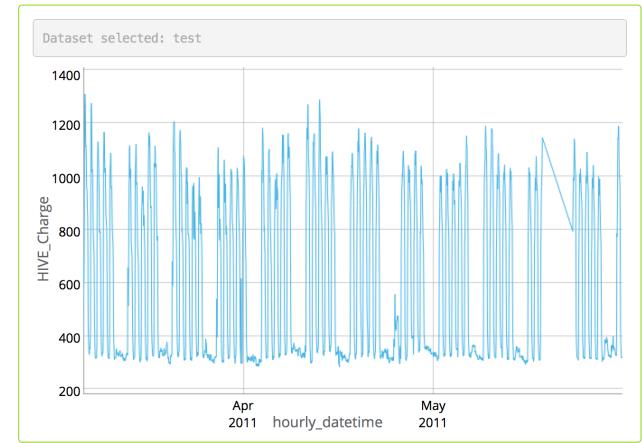
Dataset name: test

Time: hourly_datetime

Y1 axis: HIVE_Charge

Y2 axis:

Remove



model

```
# reactives
rct_data <- reactive{

  shiny::validate(
    shiny::need(rctval_data[[item_data]], "No data")
  )

  rctval_data[[item_data]]
}

rct_var_time <- reactive({
  df_names_inherits(rct_data(), c("POSIXct"))
})

rct_var_num <- reactive({
  df_names_inherits(rct_data(), c("numeric", "integer"))
})
```

shinychord

- a way to produce a coherent set of model, view, and controller elements
- so, what is a coherent set? (this is the question)
- in this case, a dygraph
- using a shinychord, a dygraph can be integrated, just a little more easily, into a shiny app

controller

Add: Upload CSV

Display

Dataset name: test

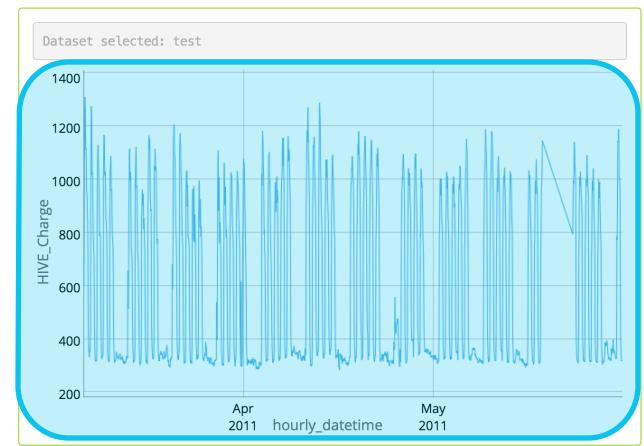
Time: hourly_datetime

Y1 axis: HIVE_Charge

Y2 axis: (empty)

Remove

view



model

```
# reactives
rct_data <- reactive{

  shiny::validate(
    shiny::need(rctval_data[[item_data]], "No data")
  )

  rctval_data[[item_data]]
}

rct_var_time <- reactive({
  df_names_inherits(rct_data(), c("POSIXct"))
})

rct_var_num <- reactive({
  df_names_inherits(rct_data(), c("numeric", "integer"))
})
```

how to build a shinychord?

Build a function that:

- takes at least one argument:
 - `id`
- returns a list with three items:
 - `ui_controller`
 - `ui_view`
 - `server_model`
- item names are just “convention”,
not strictly mandatory

```
template <- function(id){  
  ui_controller <- shiny::tagList()  
  ui_view <- shiny::tagList()  
  server_model <- function(  
    input, output, session  
>{  
  }  
  list(  
    ui_controller = ui_controller,  
    ui_view = ui_view,  
    server_model = server_model  
> )  
}
```

what's with the id?

You know this:

- ui-element id's must be unique
- non-unique id's -> 😞😡😳

You give each shinychord a unique id

The shinychord function manages the ui-element id's

shinychords << # ui-elements

Easier for you to manage

```
ch_dygraph <- function(id){  
  id_name <- function(...){  
    paste(list(id, ...), collapse = "_")  
  }  
  
  name_out <- function(x){  
    paste(x, ".out.", sep = "_")  
  }  
  
  ui_controller <- shiny::tagList()  
  # time axis  
  id_controller_time <-  
    id_name("controller", "time")  
  
  ui_controller$time <-  
    shiny::uiOutput(  
      name_out(id_controller_time)  
    )  
}
```

ui_controller?

In this case, the controller elements are shiny **uiOutputs**

These are managed by the **server_model()** function

Recall that we use the **id_name()** and **name_out()** functions to manage the **ids**

```
ch_dygraph <- function(id){  
  ui_controller <- shiny::tagList()  
  
  # time axis  
  id_controller_time <-  
    id_name("controller", "time")  
  ui_controller$time <-  
    shiny::uiOutput(  
      name_out(id_controller_time)  
    )  
  
  # y1 axis  
  id_controller_y1 <-  
    id_name("controller", "y1")  
  ui_controller$y1 <-  
    shiny::uiOutput(  
      name_out(id_controller_y1)  
    )  
  
  # y2 axis  
  id_controller_y2 <-  
    id_name("controller", "y2")  
  ui_controller$y2 <-  
    shiny::uiOutput(  
      name_out(id_controller_y2)  
    )  
}
```

ui_view?

This one is simpler

We have only to show the
dygraph

```
ch_dygraph <- function(id){  
  ui_view <- shiny::tagList()  
  # dygraph  
  id_view_dygraph <-  
    id_name("view", "dygraph")  
  ui_view$dygraph <-  
    dygraphs::dygraphOutput(id_view_dygraph)  
}
```

server_model()

`server_model()` function has mandatory arguments:

- `input`, `output`, `session`

Additional arguments for the dygraph server function:

- `rctval_data[[item_data]]`
input dataframe
- `rctval_dyopt[[item_dyopt]]`
list of dygraph options

```
ch_dygraph <- function(id){  
  server_model <- function(  
    input, output, session,  
    rctval_data, item_data,  
    rctval_dyopt = rctval_data, item_dyopt  
  ){  
  }  
}  
  
rctval_data <-  
  shiny::reactiveValues(  
    df = mtcars  
  )  
item_data <- "df"  
  
Not reactive: rctval_data, item_data  
Reactive: rctval_data[[item_data]]
```

server_model

`id_controller_time` is defined in the enclosing environment

reactive logic for *this* dygraph is encapsulated in *this* `server_model()` function

```
server_model <- function(
  input, output, session,
  rctval_data, item_data,
  rctval_dyopt = rctval_data, item_dyopt
){

  rct_var_time <- reactive({
    df_names_inherits(
      rct_data(), c("POSIXct")
    )
  })

  selection <- reactiveValues(
    time = NULL
  )

  shiny::observe({
    selection$time <-
      input[[id_controller_time]]
  })

  output[[name_out(id_controller_time)]] <-
    renderUI(
      selectizeInput(
        inputId = id_controller_time,
        label = "Time",
        choices = rct_var_time(),
        selected = selection$time
      )
    )
}
```

your turn

using shinychords

Start with just a shell of an app:

- create a new R file
- put this code in there
- run to make sure it works

```
library("shiny")
library("shinychord")

shinyApp(
  ui = fluidPage(
    titlePanel("shell"),
    sidebarLayout(
      sidebarPanel(),
      mainPanel()
    )
  ),
  server = function(
    input, output, session
  ){
  }
)
```

dygraph shinychord

we create the shinychord outside the app

```
library("shiny")
library("shinychord")
chord_dygraph <- ch_dygraph(id = "dyg")
shinyApp(
  ui = fluidPage(
    titlePanel("dygraph demo"),
    sidebarLayout(
      sidebarPanel(chord_dygraph$ui_controller),
      mainPanel(chord_dygraph$ui_view)
    )
  ),
  server = function(input, output, session){
    rctval <- reactiveValues(
      data = wx_ames,
      dyopt = list(useDataTimezone = TRUE)
    )
    chord_dygraph$server_model(
      input, output, session,
      rctval_data = rctval, item_data = "data",
      item_dyopt = "dyopt"
    )
  }
)
```

we put ui_controller and ui_view into ui

we create reactive values
(wx_ames is part of shinychord package)

server_model() called from server()

reactive values:

- are called by-reference (like pointers)
- act as function inputs and/or outputs

readr + dygraph shinychord

add new packages

(should not have to do this, looking for bug)

```
library("shiny")
library("shinychord")
library("dplyr")
library("shinyjs")
library("shinyBS")
```

we add the readr shinychord

```
chord_readr <- ch_read_delim(id = "readr")
chord_dygraph <- ch_dygraph(id = "dyg")
```

we add the ui elements

```
shinyApp(
  ui = fluidPage(
    useShinyjs(),
    titlePanel("readr & dygraph demo"),
    sidebarLayout(
      sidebarPanel(
        chord_readr$ui_controller,
        chord_dygraph$ui_controller
      ),
      mainPanel(
        chord_readr$ui_view,
        chord_dygraph$ui_view
      )
    ),
    server = function(input, output, session){
      ...
    }
  )
)
```

server function shown on next slide

readr + dygraph shinychord

rctval\$data set by readr shinychord

rctval_data[[item_data]]
• acts as an **output** for this function
• will be placed in rctval[["data"]]

rctval_data[[item_data]]
• acts as an **input** for this function
• will be taken from rctval[["data"]]

```
server = function(input, output, session){  
  
  rctval <- reactivevalues(  
    data = NULL,  
    dyopt = list(useDataTimezone = TRUE)  
  )  
  
  chord_readr$server_model(  
    input, output, session,  
    rctval_data = rctval, item_data = "data"  
  )  
  
  chord_dygraph$server_model(  
    input, output, session,  
    rctval_data = rctval, item_data = "data",  
    item_dyopt = "dyopt"  
  )  
}
```

You may test your app using the csv file found at

http://ijlyttle.github.io/shinychord/doc/wx_ames.csv

some sort of summary

- You don't need the shinychord package to write your own.
- It's more of an idea, an idea that can be improved-upon.
- It has allowed us to (confidently) build shiny apps an order-of-magnitude more complex than before.
- How:
 - letting us focus on the parts that are exposed
 - hiding the messiness



reactive spaghetti

reactive ravioli

