Enable bookmarking for a Shiny application

Description

There are two types of bookmarking: saving an application's state to disk on the server, and encoding the application's state in a URL. For state that has been saved to disk, the state can be restored with the corresponding state ID. For URL-encoded state, the state of the application is encoded in the URL, and no server-side storage is needed.

URL-encoded bookmarking is appropriate for applications where there not many input values that need to be recorded. Some browsers have a length limit for URLs of about 2000 characters, and if there are many inputs, the length of the URL can exceed that limit.

Saved-on-server bookmarking is appropriate when there are many inputs, or when the bookmarked state requires storing files.

Usage

```
enableBookmarking(store = c("url", "server", "disable"))
```

Arguments

`store`	Either `"url"`, which encodes all of the relevant values in a URL, `"server"`, which saves to disk on the server, or `"disable"`, which disables any previously-enabled bookmarking.

Details

For restoring state to work properly, the UI must be a function that takes one argument, `request`. In most Shiny applications, the UI is not a function; it might have the form `fluidPage(....)`. Converting it to a function is as simple as wrapping it in a function, as in `function(request) { fluidPage(....) }`.

By default, all input values will be bookmarked, except for the values of passwordInputs. fileInputs will be saved if the state is saved on a server, but not if the state is encoded in a URL.

When bookmarking state, arbitrary values can be stored, by passing a function as the `onBookmark` argument. That function will be passed a `ShinySaveState` object. The `values` field of the object is a list which can be manipulated to save extra information. Additionally, if the state is being saved on the server, and the `dir` field of that object can be used to save extra information to files in that directory.

For saved-to-server state, this is how the state directory is chosen:

- If running in a hosting environment such as Shiny Server or Connect, the hosting environment will choose the directory.
- If running an app in a directory with `runApp()`, the saved states will be saved in a subdirectory of the app called shiny_bookmarks.
- If running a Shiny app object that is generated from code (not run from a directory), the saved states will be saved in a subdirectory of the current working directory called shiny_bookmarks.

When used with `shinyApp()`, this function must be called before `shinyApp()`, or in the

`shinyApp()`'s `onStart` function. An alternative to calling the `enableBookmarking()` function is to use the `enableBookmarking` *argument* for `shinyApp()`. See examples below.

See Also

`onBookmark`, `onBookmarked`, `onRestore`, and `onRestored` for registering callback functions that are invoked when the state is bookmarked or restored.

Also see `updateQueryString`.

Examples

```
## Only run these examples in interactive R sessions
if (interactive()) {

# Basic example with state encoded in URL
ui <- function(request) {
  fluidPage(
    textInput("txt", "Text"),
    checkboxInput("chk", "Checkbox"),
    bookmarkButton()
  )
}
server <- function(input, output, session) { }
enableBookmarking("url")
shinyApp(ui, server)


# An alternative to calling enableBookmarking(): use shinyApp's
# enableBookmarking argument
shinyApp(ui, server, enableBookmarking = "url")


# Same basic example with state saved to disk
enableBookmarking("server")
shinyApp(ui, server)


# Save/restore arbitrary values
ui <- function(req) {
  fluidPage(
    textInput("txt", "Text"),
    checkboxInput("chk", "Checkbox"),
    bookmarkButton(),
    br(),
    textOutput("lastSaved")
  )
}
server <- function(input, output, session) {
  vals <- reactiveValues(savedTime = NULL)
  output$lastSaved <- renderText({
    if (!is.null(vals$savedTime))
      paste("Last saved at", vals$savedTime)
    else
      ""
  })

  onBookmark(function(state) {
```

```
    vals$savedTime <- Sys.time()
    # state is a mutable reference object, and we can add arbitrary values
    # to it.
    state$values$time <- vals$savedTime
  })
  onRestore(function(state) {
    vals$savedTime <- state$values$time
  })
}
enableBookmarking(store = "url")
shinyApp(ui, server)


# Usable with dynamic UI (set the slider, then change the text input,
# click the bookmark button)
ui <- function(request) {
  fluidPage(
    sliderInput("slider", "Slider", 1, 100, 50),
    uiOutput("ui"),
    bookmarkButton()
  )
}
server <- function(input, output, session) {
  output$ui <- renderUI({
    textInput("txt", "Text", input$slider)
  })
}
enableBookmarking("url")
shinyApp(ui, server)


# Exclude specific inputs (The only input that will be saved in this
# example is chk)
ui <- function(request) {
  fluidPage(
    passwordInput("pw", "Password"), # Passwords are never saved
    sliderInput("slider", "Slider", 1, 100, 50), # Manually excluded below
    checkboxInput("chk", "Checkbox"),
    bookmarkButton()
  )
}
server <- function(input, output, session) {
  setBookmarkExclude("slider")
}
enableBookmarking("url")
shinyApp(ui, server)


# Update the browser's location bar every time an input changes. This should
# not be used with enableBookmarking("server"), because that would create a
# new saved state on disk every time the user changes an input.
ui <- function(req) {
  fluidPage(
    textInput("txt", "Text"),
    checkboxInput("chk", "Checkbox")
  )
}
server <- function(input, output, session) {
```

```
  observe({
    # Trigger this observer every time an input changes
    reactiveValuesToList(input)
    session$doBookmark()
  })
  onBookmarked(function(url) {
    updateQueryString(url)
  })
}
enableBookmarking("url")
shinyApp(ui, server)


# Save/restore uploaded files
ui <- function(request) {
  fluidPage(
    sidebarLayout(
      sidebarPanel(
        fileInput("file1", "Choose CSV File", multiple = TRUE,
          accept = c(
            "text/csv",
            "text/comma-separated-values,text/plain",
            ".csv"
          )
        ),
        tags$hr(),
        checkboxInput("header", "Header", TRUE),
        bookmarkButton()
      ),
      mainPanel(
        tableOutput("contents")
      )
    )
  )
}
server <- function(input, output) {
  output$contents <- renderTable({
    inFile <- input$file1
    if (is.null(inFile))
      return(NULL)

    if (nrow(inFile) == 1) {
      read.csv(inFile$datapath, header = input$header)
    } else {
      data.frame(x = "multiple files")
    }
  })
}
enableBookmarking("server")
shinyApp(ui, server)

}
```
