Package-Wide Variables/Cache in R Packages

Posted by Jeff Allen on Apr 8, 2013 | 10 comments

It's often beneficial to have a variable shared between all the functions in an R package. One obvious example would be the maintenance of a package-wide cache for all of your functions. I've encountered this situation multiple times and always forget at least one important step in the process, so I thought I'd document it here for myself and anyone else who might encounter this issue. I setup a simple project on GitHub to demonstrate the various attempts you may take to solve the problem and, ultimately, a solution: https://github.com/trestletech/RCache. The rest of this article will presume some knowledge of authoring R packages. If that's not you, check out RStudio's guide to authoring R packages.

The fundamental problem comes down to R's management of *environments*. (I'll introduce the basics here, but for a thorough discussion on the topic, be sure to check out Hadley Wickham's writings on the topic.) An environment is responsible for mapping data to named objects. When I execute

```
> a <- "hello"
> a
[1] "hello"
```

there was an environment responsible for initially associating the length-one character vector containing the text `hello` with the variable named `a`. Later, when I go to reference a variable by name, an environment is responsible for retrieving the data I had associated with this variable. Functions have their own environments in which they run, which is why you observe behavior like:

```
> x <- 1
> foo <- function(){
  print (x)
  x <- 2
  print(x)
  x
}
> x
```

```
[1] 1
> foo()
[1] 1
[1] 2
[1] 2
> x
[1] 1
```

You can see that the variable named `x`, depending on which environment it's in, can take one of two values in this example. Inside of the `foo` function, the `x` variable initially only existed in the parent environment, so it took on the value of `x` in that environment and was `1`. Later, it was assigned a value of `2` and retained that value when it was returned. Outside of the function, however, `x` was associated with the value `1`, and this binding was maintained despite another variable named `x` being created inside of a function's environment.

This is a powerful feature of R, when properly understood. This allows you to create variables that only exist inside of a function or, more relevant to today's discussion, only within a particular package. So it should be possible to leverage these environments to create a variable named `cache` which is accessible to all the functions in my package, but won't accidentally likely be overwritten or modified by the user or, even more likely, collide with another variable named `cache` used in some other package.

Example Package – rev. a804

For the rest of the demonstration, we'll use the example of creating an R package which merely downloads files from the Internet. Of course, it might be appropriate to cache data in such a package so that the same URL won't have to be retrieved remotely multiple times. One simple approach would be to use a `list` to associated character strings (such as URLs) with some data (such as the content of the web page). We can first create a function which will download data without using any cache:

```
#' Download a file.
#'
#' @importFrom httr GET
#' @importFrom httr content
#' @export
download <- function(url){
  content(GET(url))
}
```

Example Package – rev. a81e

Creating a variable that is available within all functions of your package is as simple as binding a variable to data outside of any functions in a .R file in your package.

```
cache <- list()
```

One might think we could then put data into that cache variable by assigning named elements within to it from

packages that can access it. For instance:

```
download <- function(url){
  if (!is.null(cache[[url]])){
    return(cache[[url]])
  }

  file <- content(GET(url))
  cache[[url]] <- file

  file
}
```

Unfortunately, this, like our first example, is assigning data to a separate variable named `cache` which exists only inside of this function. If you were to build the package and run it, you'd find that the code:

```
down <- download("http://www.gutenberg.org/cache/epub/2500/pg2500.txt")
RCache:::cache
```

would successfully download the book, *Siddhartha*, but the package cache would be empty. (If you're unfamiliar with the `:::` operator, it checks inside of the package — named on the left — for a variable — named on the right and returns it. So we can inspect the `cache` variable from code outside of our package.)

Special Assignment – rev. 5548

In order to alter a variable created in the parent environment, one must use the *special* assignment operator, `<<-`. This will adjust a binding not in the current environment, but in a parent environment. So we can adjust our line in which we assign a value to the cache variable to use this operator:

```
  file <- content(GET(url))
  cache[[url]] <<- file
```

However, if we run this, we'll find that we get an interesting error:

```
down <- download("http://www.gutenberg.org/cache/epub/2500/pg2500.txt")
Error in cache[[url]] <<- file :
  cannot change value of locked binding for 'cache'
```

What's the deal? R has a concept of "Locked Bindings" which allow you to ~~forbid~~ *discourage* changes in variables by locking either a particular variable binding or an entire environment. In this case, the `cache` binding has been locked and can't be altered by constituent functions. So we'll need to take a different approach altogether.

Environments – rev. 320e

It seems we can properly access a package-wide variable from within a function of a package, but we're not allowed to overwrite it (or create new variables at that level from within a function). Perhaps we could leverage environments to solve this problem. As it turns out, environments were likely a cleaner solution to our problem all along. Instead of creating a cache `list`, we can create it a cache `environment`:

```
cacheEnv <- new.env()
```

As long as this environment is created in one of our package's `.R` files and not inside of a function, it will be accessible across our entire package. We can do all the things with this environment that we're used to doing in our regular R environment (whether we knew we were using environments or not): create new variables (`assign`), modify existing variables (`assign`), remove variables (`rm`), retrieve data associated with a variable (`get`), list the variables in an environment (`ls`), etc.

All of the functions mentioned above accept an `envir` argument which specifies in which environment you'd like to perform the operation. The default is your current environment, but you could just as easily point these functions at your new environment to do something like `assign(url, file, envir=cacheEnv)` to assign the value currently stored in the `file` variable to a new variable who's name is the value currently contained in the `url` variable within our cache environment. Then we could use `get(url, envir=cacheEnv)` to get the variable who name matches the current value of the `url` variable in the cache environment.

For instance:

```
> url <- "http://mytext.com"
> file <- "This is the content I downloaded"
> cacheEnv <- new.env()
> assign(url, file, envir=cacheEnv)
> get(url, envir=cacheEnv)
[1] "This is the content I downloaded"
```

Now we can incorporate this into our package by changing the `download` function to use these facilities:

```
download <- function(url){
  if (exists(url, envir=cacheEnv)){
    return(get(url, envir=cacheEnv))
  }

  file <- content(GET(url))
  assign(url, file, envir=cacheEnv)

  file
}
```

And we finally have a working cache. When a URL is requested via our package's `download` function, the data will first be stored in this package's `cacheEnv` environment before being returned. The next time that URL is requested, the `cacheEnv` environment will be checked to see if we already downloaded the content of that URL. Because a variable by that name already `exists` in our `cacheEnv` environment, the value will be pulled from there and returned rather than retrieved remotely.

Conclusion

Hopefully you've learned a thing or two about environments in R and how to use them from within R packages. Environments can be very valuable tools for advanced R programmers. They're one of the tried-and-true ways to replicate "pass-by-reference" programming in R (though that's typically unexpected — thus discouraged — behavior for an R object). They also have some unique lookup properties being built on hashes that allow them to more expediently map character strings to data when there is a very large number of possible keys.

Happy packaging!

10 Comments

1. 

09-04-2013
Alex Ishkin says:

Hi Jeff,

Thanks for the post, it is very clearly written! I use the environment-based cache in my packages as well. In my code, the cache environment is created by a function and explicitly placed into global environment (so it is in the search path for whatever operations executed in the console. I'd like to ask: if you just create the cache env somewhere in R script (I guess in this case it is created when the package is loaded?), is it visible from global environment or only within namespace of the package?

Thanks,
Alex

Reply

   - 

09-04-2013
Jeff Allen says:

Thanks for the kind words, Alex. The way I have it setup, the cache variable is in the package's namespace, not the global environment. So it's accessible from within my package's code, but not elsewhere. For instance:

```
> library(RCache)
> down < - download("http://www.gutenberg.org/cache/epub/2500/pg2500.txt")
> cacheEnv
Error: object 'cacheEnv' not found
> RCache:::cacheEnv
```

Reply

2. 

09-04-2013
Hansi says:

What about?

```
item <- something()
options(MY.LIB.CACHEDITEM=blah)

[snip]

needItemAgain <- getOption("MY.LIB.CACHEDITEM")
if(is.null(needItemAgain)){needItemAgain <- something()}
```

Reply

09-04-2013
Jeff Allen says:

I'd have a couple of concerns with that approach.

1. One of my goals was to keep the cache variable/environment out of the global namespace with the intent that we don't want the user or another package accidentally breaking our cache. You were careful to preface your variable name with `MY.LIB`, though, so you bypass that argument. This argument may just descend to a point at which I'm blindly subscribing to "best practice"-ism.
2. I'd worry about the performance of this approach at scale. One of the benefits of environments being hashes is that they offer `O(1)` lookup regardless of size. I believe `options` are list-based (correct me if I'm wrong), which don't offer such performance at scale. This may be a more theoretical argument, however, as the couple of times I've actually compared them, the lists outperformed the hashes until you got to a few hundred/thousand elements.

All in all, I think this would be a perfectly functional approach (no pun intended) if you're willing to accept some "pollution" in the global namespace.

Reply

10-04-2013
Hansi says:

1) Sure I guess it's just a question of ownership of the cache. If releasing a public package I think your method would most likely be better received.

2) Not 100% on this but assignment can be direct with name or as a list for multiple assignment. Internally I think it's handled in the same/similar manner as enviroments: http://svn.r-project.org/R/trunk/src/main/options.c so there shouldn't be any overhead.

Reply

3. 

22-04-2014
Flora says:

Hi Jeff,

I really liked your post it seems really promising for my problem. Though I am not sure how to use it. I will explain you my problem and I hope you will understand. I am working in an existing code(someone wrote it and asked me to fix it) in this code I have for 30 different R files and they are linked the first file for example calls another one and this goes on for the rest files.
Because they are used many different files some of the variables are setted as globals (var<<-) when I run the code in some global variables it appears an error :Error in nobs (which is one global variable) : cannot change value of locked binding for 'nobs'
this error is exactly as yours and I think it means that the global variable can be read but not overwritten also if it helps the variables that seems to cause the problem reappear in different files and they are setted as globals to many files (also I want to mention that in general not all globals have problem).
My idea was to try the assign function so I could overwrite those global variables who had a problem though I was not sure which environment should I write in the function so I wrote assign(vars,value,env=.GlobalEnv) this seems to work and no error appears though the results are not what they should be. It seems I messed up the environments(because also the code is not mine and to be a bit more honest I dont know exactly how environments work in R) So, I though to try something with cacheEnv but I don't know how I should use it exactly. In other words if you have any type of suggestion or you think something can work please let me know.

Kind Regards,
Flora

Reply

- 

  22-04-2014
  Jeff Allen says:

  Yeah, you'll need to create your own environment where you store your globals like we do in `set-cache.R`:
  https://github.com/trestletech/RCache/tree/320e3df3b9078a3d0ad81286a68f9a2e83de93a2/R .

  So any current code you have that alters `nobs` will use `assign("nobs", envir=myNewEnv")` and anything that gets `nobs` will need use `get("nobs", envir=myNewEnv)`, assuming that you created a new environment in your package somewhere named `myNewEnv`.

  Reply

23-04-2014
Flora says:

Thank you very much for your reply!!:)
My kind of extra problem is that the files are not in a specific package so I cannot call the specific environment package:::MyNewEnvir.
The idea is that they gave me 30 files which interacts (like a package).
The code which is given is source("path/" etc) and like that I load all the different files..so I mean it works like a package but the format is not like those of a package, this is why I find difficult to find the environments. What I thought with your solution was to create an extra R file set-cache (as you told me) and call it whenever is needed, but even if it could worked (which did not happen) each time I would call a new environment and this is not what I think is needed. Maybe if I just do not use a set-cache environment and I just call in the assign("var",value,env=.GlobalEnv) only for the variables that the value change..but this does not work either I have results but the results are not what they should be.
Do you think I should make it work as a package?? (no idea how I can do it and if it is right to do it the code is not even mine). Any suggestions or can you imagine why such a problematic results are produced?? (I want also to mention that I work in Rstudio the latest version and the code without modifications works fine in R 2.7.1 but when I run it in later versions has the problem with the binding..and all the other problems which I mentioned them before). Any suggestion would be really appreciated!

Kind Regards and thank you again for your reply,
Flora

Reply

23-04-2014
Jeff Allen says:

Yeah, I think you'll find it easier to manage the code if it's an R package, and it's certainly easier to distribute to users in that format. You'd just have to be sure you have the rights to modify/distribute the code in that way.

Here's a helpful place to get started with package authorship.
http://www.rstudio.com/ide/docs/packages/overview

Reply

25-04-2014
Flora says:

Thank you very much!
But I figure out that my problem might be caused from computational precision which must be different in R2.7 and R studio.

Regards,
Flora

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment

You may use these HTML tags and attributes: `<a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <em> <i> <q cite=""> <strike> <strong> <pre class="" title="" data-url=""> <span class="" title="" data-url="">`

Submit Comment

Search

Recent Posts

- Ace Code Editor in Shiny (shinyAce)
- Interactive 3D in Shiny (shinyRGL)
- Reproducing R: Scripts, Documents, and Packages
- Dallas R Users: Creating R Packages this Saturday, 6/29
- Package-Wide Variables/Cache in R Packages

Recommended Sites

- R Bloggers

Archives

- November 2013
- October 2013
- June 2013
- April 2013
- February 2013
- January 2013
- December 2012
- October 2012
- June 2012
- April 2012
- November 2011
- August 2011
- April 2011
- May 2010

Powered by WordPress
Twitter
Facebook
LinkedIn
GitHub
Stack Overflow
RSS