How to use weaver for Sweave document processing

Seth Falcon

8 June, 2006

1 Introduction

The *weaver* package provides extensions to the Sweave utilities included in R's *utils* package. The focus of the extensions is on caching computationally expensive (time consuming) code chunks in Sweave documents.

Why would I want to cache code chunks? If your Sweave document includes one or more code chunks that take a long time to compute, you may find it frustrating to make small changes to the document. Each run requires recomputing the "expensive" code chunks. If these chunks aren't changing, you can benefit from the caching provided by *weaver*.

How does it work? The details are in the code, of course, but in a few words... You tell *weaver* which code chunks you want cached by setting a chunk option (`cache=TRUE`). A digest (md5 sum) of the text representation of *each* expression in the code chunk is computed and the result of the expression is stored in a file named by the expression's digest. Dependencies on previously cached expressions are determined using functions from the *codetools* package. After the cache files have been created, subsequent runs load the cache instead of evaluating the expression (this means side-effects are completely lost!). When changes in the dependencies of an expression are detected (or when the expression itself has changed), it is recomputed and the cache file is updated.

2 Using the expression caching feature

If you add the chunk option `cache=TRUE`, then caching will be turned on for all expressions in the chunk. Here's an example:

```
<<someChunk, cache=TRUE>>=
## All expressions will be cached
##    NO SIDE EFFECTS
b <- rnorm(3)
c <- runif(3)
z <- b + c
@
```

Side-effects, such as printing, plotting, definging S4 classes or methods, or setting global options are not captured by the caching mechanism. Avoid doing such things in a code chunk that has `cache=TRUE`. Treat cached code chunks as if you had set the option `results=hide`.

2.1 Warnings about using the caching feature

Do not stare directly at the cache! May cause blindness, headache, shortness of breath, and dizzyness.

- Printing doesn't work in cached chunks since it is a side effect.

- The dependency detection is imperfect and will fail you. When you've made important changes, you should remove all cache files and rebuild the document. By default, the cache database is stored in a directory named `r_env_cache` in the current working directory. Removing this directory is the best way to be certain that the following run will not use any cached data. A log file is produced in the current working directory named `weaver_debug_log.txt`. Reviewing it can be useful in determining what the *weaver* system thinks the dependencies of a given expression are.
- Caching is performed separately on each expression in a chunk which has the option `cache=TRUE` set. Be especially careful with repeated calls to random number based functions like `rnorm`. Repeated calls within cached chunks will pull from the cache rather than computing a new stream of random numbers.
- The cache is not document specific. If you have two documents in the same working directory that contain equivalent expressions within a chunk that has caching turned on, you will get the cached value. I think this is a feature and will be useful for testing purposes, but could be surprising.

3 Processing a document from inside R

To process a document using *weaver*, load the *weaver* package and then use `weaver()` as the `driver` argument to `Sweave`. Here is an example:

```
> library("weaver")
> testDocPath <- system.file("extdata/doc1.Rnw", package="weaver")
> curDir <- getwd()
> setwd(tempdir())
> z <- capture.output(Sweave(testDocPath, driver=weaver()),
+                     file=tempfile())
> setwd(curDir)
```

Note that the calls to `setwd` are only needed here because we are processing an Sweave document inside an Sweave document. Also, `capture.output` was used to keep this document short and to encourage you to run the examples yourself[1]

Now we run another sample document.

```
> testDocPath <- system.file("extdata/doc2.Rnw", package="weaver")
> curDir <- getwd()
> setwd(tempdir())
> z <- capture.output(Sweave(testDocPath, driver=weaver()),
+                     file=tempfile())
> setwd(curDir)
```

Finally, we run our first example document again. This time, you can see that data from the cache is being used.

```
> testDocPath <- system.file("extdata/doc1.Rnw", package="weaver")
> curDir <- getwd()
> setwd(tempdir())
> z <- capture.output(Sweave(testDocPath, driver=weaver()),
+                     file=tempfile())
> setwd(curDir)
```

[1] In addition, some of the output is sent to stderr and this is not captured when running Sweave inside Sweave.

4 Sample convenience shell script

You can use this shell script to make processing `Rnw` files with *weaver* easier.

```
#!/bin/bash

echo "library(weaver); Sweave(\"$1\", driver=weaver())" \
  | R --no-save --no-restore
```

If you put that into a file `weaver.sh`, then you can do:

```
weaver.sh somefile.Rnw
```

to process `somefile.Rnw` with *weaver*. Another useful script is one that does the processing without using any cached data. This is useful, for example, when you are ready to produce a final draft of your document.

```
#!/bin/bash

echo "library(weaver); Sweave(\"$1\", driver=weaver(), use.cache=FALSE)" \
  | R --no-save --no-restore
```

5 Session Info

```
> toLatex(sessionInfo())
```

- R version 4.2.1 (2022-06-23), `aarch64-apple-darwin20`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Running under: `macOS Ventura 13.0`
- Matrix products: default
- BLAS: `/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib`
- LAPACK: `/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib`
- Base packages: base, datasets, grDevices, graphics, methods, stats, tools, utils
- Other packages: codetools 0.2-18, digest 0.6.29, weaver 1.64.0
- Loaded via a namespace (and not attached): compiler 4.2.1