Caching

Janko Thyson

Thursday, October 29, 2014

Suggested

read vignette Registry prior to reading this vignette

Overview

The caching mechanism implemented by `reactr` relies on keeping a registry that stores certain information that are either required or very useful for reacting to respective system states (e.g. for deciding whether an update should be triggered or not).

The rule of thumb for caching is as follows:

> If `B` depends on `A` and `A` has not changed: –> use the last cache value if B is requested If `B` depends on `A` and `A` has changed: –> execute the reactive binding function and thus also update the cached value

Checksums

The actual decision is based on a comparison of MD5 checksums of the values of the **visible** objects as computed by `digest::digest()`.

These are stored in the environment `getOption(".reactr")$.registry` which is accessible via the convenience function `getRegistry()`.

Besides the actual checksum values, each registry entry - corresponding to the **invisible** reactive object - also contains some other information that is often required/useful:

- `.id`: object ID as specified in call to `setReactiveS3()`
- `.uid`: object UID computed as follows: `digest::digest(list(id = id, where = {where}))` where `{where}` stands for the location provided via argument `where` in the call to `setReactiveS3()`
- `{uid}`: subenvironment corresponding to the objects `UID. This contains the objects` own checksum
- `{ref-uid}`: subenvironments for each referenced object should there exist any. These in turn contain the referenced object‘s checksum that is used to determine if an update is necessary or not.

Actual registry object

The actual registry object can be retrieved by the convenience function `getRegistry()`:

```
registry <- getRegistry()
ls(registry)
```

```
## character(0)
```

```
showRegistry()
```

```
## character(0)
```

Illustrating the role of the registry

```
## Start with fresh registry //
resetRegistry()
```

```
## [1] TRUE
```

```
## Set reactive object //
setReactiveS3(id = "x_1", value = 10)
showRegistry()
```

```
## [1] "2fc2e352f72008b90a112f096cd2d029"
```

```
## --> Object with UID 2fc2e352f72008b90a112f096cd2d029 has been registered
## The object UID was computed by:
computeObjectUid("x_1")
```

```
## [1] "2fc2e352f72008b90a112f096cd2d029"
```

```
## Retrieve from registry //
(reg_x_1 <- getFromRegistry("x_1"))
```

```
## <environment: 0x0000000009aa15f8>
## attr(,"class")
## [1] "ReactiveObject.S3" "environment"
```

```
## --> same as manually selecting the respective object from the registry
## environment:
registry[[computeObjectUid("x_1")]]
```

```
## <environment: 0x0000000009aa15f8>
## attr(,"class")
## [1] "ReactiveObject.S3" "environment"
```

```
## Inspect structure of invisible object //
ls(reg_x_1, all.names = TRUE)
```

```
##  [1] ".blockUpdate"                  ".cache"
##  [3] ".caller"                       ".checkClass"
##  [5] ".checksum"                     ".class"
##  [7] ".compareChecksums"             ".computeChecksum"
##  [9] ".computeUid"                   ".condition"
## [11] ".copy"                         ".ensureIntegrity"
```

```
## [13] ".ensurePullReferencesIntegrity" ".exists_visible"
## [15] ".func"                          ".getVisible"
## [17] ".has_bidir"                     ".has_cached"
## [19] ".has_pull_refs"                 ".has_push_refs"
## [21] ".has_pushed"                    ".hasBidirectional"
## [23] ".hasPullReferences"             ".hasPushReferences"
## [25] ".id"                            ".is_invalid"
## [27] ".is_modcycle_complete"          ".is_running_push"
## [29] ".isCallerModcycleComplete"      ".must_push"
## [31] ".needs_update"                  ".pushToReferences"
## [33] ".refs_checksum"                 ".refs_pull"
## [35] ".refs_push"                     ".register"
## [37] ".registerPullReferences"        ".registerPushReferences"
## [39] ".registry"                      ".remove"
## [41] ".uid"                           ".unregister"
## [43] ".unset"                         ".updateReferenceChecksum"
## [45] ".value"                         ".where"
```

```
reg_x_1$.id
```

```
## [1] "x_1"
```

```
reg_x_1$.uid
```

```
## [1] "2fc2e352f72008b90a112f096cd2d029"
```

```
reg_x_1$.where
```

```
## <environment: R_GlobalEnv>
```

```
reg_x_1$.checksum
```

```
## [1] "2522027d230e3dfe02d8b6eba1fd73e1"
```

```
## Set additional reactive object //
setReactiveS3(
  id = "x_2",
  value = function() {
    ## object-ref: {id: x_1}
    x_1 * 2
  }
)
showRegistry()
```

```
## [1] "2fc2e352f72008b90a112f096cd2d029" "ab22808532ff42c87198461640612405"
```

```
## --> two entries

## Inspect invisible object associated to visible value `x_2`
reg_x_2 <- getFromRegistry("x_2")
ls(reg_x_2, all.names = TRUE)
```

```
##  [1] ".blockUpdate"                    ".cache"
##  [3] ".caller"                         ".checkClass"
##  [5] ".checksum"                       ".class"
##  [7] ".compareChecksums"               ".computeChecksum"
##  [9] ".computeUid"                     ".condition"
## [11] ".copy"                           ".ensureIntegrity"
## [13] ".ensurePullReferencesIntegrity" ".exists_visible"
## [15] ".func"                           ".getVisible"
## [17] ".has_bidir"                      ".has_cached"
## [19] ".has_pull_refs"                  ".has_push_refs"
## [21] ".has_pushed"                     ".hasBidirectional"
## [23] ".hasPullReferences"              ".hasPushReferences"
## [25] ".id"                             ".is_invalid"
## [27] ".is_modcycle_complete"           ".is_running_push"
## [29] ".isCallerModcycleComplete"       ".must_push"
## [31] ".needs_update"                   ".pushToReferences"
## [33] ".refs_checksum"                  ".refs_pull"
## [35] ".refs_push"                      ".register"
## [37] ".registerPullReferences"         ".registerPushReferences"
## [39] ".registry"                       ".remove"
## [41] ".uid"                            ".unregister"
## [43] ".unset"                          ".updateReferenceChecksum"
## [45] ".value"                          ".where"
```

```
reg_x_2$.id
```

```
## [1] "x_2"
```

```
reg_x_2$.uid
```

```
## [1] "ab22808532ff42c87198461640612405"
```

```
reg_x_2$.where
```

```
## <environment: R_GlobalEnv>
```

```
reg_x_2$.checksum
```

```
## [1] "c72d0198505540505acfb4b2b49911e6"
```

```
ls(reg_x_2$.refs_pull)
```

```
## [1] "2fc2e352f72008b90a112f096cd2d029"
```

```
## --> pull references --> reference to `x_1` or UID 2fc2e352f72008b90a112f096cd2d029
reg_x_1_through_x_2 <- reg_x_2$.refs_pull[["2fc2e352f72008b90a112f096cd2d029"]]
## --> same as invisible object behind `x_1` or object with
## UID 2fc2e352f72008b90a112f096cd2d029 in registry:
identical(reg_x_1, reg_x_1_through_x_2)
```

```
## [1] TRUE
```

```
## --> that way all references are always accessible which is quite handy
## for a lot of situations, including push updates and integrity checks.
```

Clean up

```
## [1] TRUE
```

```
## [1] TRUE
```

```
## [1] TRUE
```