Win-Vector Blog

The Win-Vector LLC data science blog

For loops in R can lose class information

March 24, 2016 John Mount Uncategorized R, R as it is, R is not your friend

Did you know R's `for()` loop control structure drops class annotations from vectors?

Consider the following code R code demonstrating three uses of a `for-loop` that one would expect to behave very similarly.

```
dates <- c(as.Date('2015-01-01'),as.Date('2015-01-02'))

for(ii in seq_along(dates)) {
  di <- dates[ii]
  print(di)
}
## [1] "2015-01-01"
## [1] "2015-01-02"

for(di in as.list(dates)) {
  print(di)
}
## [1] "2015-01-01"
## [1] "2015-01-02"

for(di in dates) {
  print(di)
}
## [1] 16436
## [1] 16437
```

Notice in the third for loop the `di` print as numbers. This is because running through the `dates` in this way loses the class annotations. To me this is a huge undesirable surprise (given that indexing does not lose class information). Remember with the class information missing many more behaviors that just printing may be broken. The third loop is the "most natural" as it doesn't introduce an index or re-process the vector prior to iterating. But the third loop seems to not be safe to use with code that depends on class annotations being preserved.

The work arounds are shown prior to the failure (introducing an index or converting the vector into a list).

Also notice `vapply()` also loses the class info, even when you explicitly supply it:

```
vapply(dates,function(x) {x+0},as.Date(0))
## [1] 16436 16437
```

I understand the `vapply()` case, as `is.vector(dates)` is false (due to the class annotation) and one would expect to return something that has `is.vector()` true. But the for-loop behavior is a real head-scratcher that took a while to believe was actually happening when a partner ran into it.

For a lot of languages `for(di in dates) { ... }` is roughly syntactic sugar for `for(ii in seq_along(dates)) { di <- dates[ii]; ... }` (assuming `ii` is not used elsewhere in the code, i.e. we have a so-called "hygienic" substitution). So it is a big surprise that these two code fragments behave so differently in R. It also make R a bit harder to teach.

I didn't anticipate these issues from R's basic documentation which says:

> *For each element in vector the variable name is set to the value of that element and statement1 is evaluated.*

I am guessing this is to be read with the additional understanding that `dates[i]` is *not* the just value in the `i`-th position of `dates` but in fact a length-1 vector with the same class annotation as `dates` containing that value (so think of `[]` is performing extra work to copy over this information). Of course the for loop can't be working only through "naked" values as R doesn't expose scalars to user code, therefore any values a for-loop is iterating over must also appear re-wrapped in a vector structure.

If anybody has some good teaching material on this I'd love to see it.

SHARE THIS:

3 8

RELATED

Factors are not first-class citizens in R
In "Computer Science"

Don't use stats::aggregate()
When working with an analysis system (such as R) there are usually good reasons to prefer using functions from the "base" system over using functions
In "data science"

What is new in the vtreat library?
The Win-Vector LLC vtreat library is a library we supply (under a GPL license) for automating the simple domain independent part of variable
In "Practical Data Science"

March 24, 2016 John Mount Uncategorized R, R as it is, R is not your friend

9 thoughts on “For loops in R can lose class information”

Elin

March 25, 2016 at 6:19 am

The index has to be numeric and you are correct that it will not just assume that you really meant to start with 1 and go to the last one, which actually is quite useful at times such as when you don’t want to do that. So it is converting the dates to a number. I guess it doesn’t warn you because people might be doing this on purpose.

```
for(di in dates) {
# Convert it back to a date.
a <- as.Date(di, origin = "1960-01-02")
print(a)
}
```

or you could not treat the date value as the index and instead treat the dates index as the index.

```
for(di in 1:length(dates)) {
print(dates[di])
}
```

John Mount

March 25, 2016 at 8:42 am

Thanks Elin. I thought the “seq_along()” would be a safer way to move to indexes (as 1:length(dates) doesn’t work when dates is zero length). The converting back seems too dangerous to rely on in production.

John Mount

March 25, 2016 at 8:40 am

I am guessing all of this is an artifact of the S3 class system entering the S/R ecosystem after the for-loop was already established. Its funny in Java the collection type for-loop entered later, so it was defined to emulate the index style for-loop semantics.

Mark

March 25, 2016 at 2:55 pm

ifelse() has the same problem, I believe:
ifelse(dates == as.Date(“2015-01-01”), dates, dates + 1)
[1] 16436 16438

John Mount

March 25, 2016 at 3:49 pm

Thanks, `ifelse()` is pretty interesting. If you look at the code (`print(ifelse)`) you can see it goes to great lengths to stuff the answers into a vector that is built from the test-vector using explicit commands (such as `as.logical()`) to strip off attributes (such as `class`).

macherki

March 25, 2016 at 3:18 pm

Some explicit “for”{} overcomes this problem. For some case, the problem is tha class procedure itself! Then a wrapped function is evident.

Henrik Bengtsson

March 25, 2016 at 5:15 pm

There has been some quite recent attempts to fix this in R devel (https://github.com/wch/r-source/commit/13ed8e897212b5da21652b5079a852dd6f896706) but they rolled it back again because of unanticipated problems. But, I'm pretty sure they'll figure it out later.

John Mount

March 26, 2016 at 8:11 am

I am guessing the issue is that in the S3 object system the value `c(as.Date('2015-01-01'),as.Date('2015-01-02'))` "should" be interpreted as a single object of class "Date" that happens to contain two values in its implementation, and not as two Date values in a vector (similar for POSIXct).

For example in defining a class such "PlaneCoordinate" as a class over numeric vectors it might make sense to expect such objects are always implemented over vectors of length 2, and splitting them up doesn't "isn't sensible."

This is sort of following the principle that in object oriented programming objects tend to preserve invariants (you tend not to rip into their member slots directly).

However, the `c()`, `[]`, and `[[]]` operators clearly all treat such vectors as vectors of objects (not as a single object encapsulating a vector of values). So it is a bit inconsistent for `for()` to have the opposite interpretation.

Likely some code breaks if we were to wish in a new behavior for `for()`.

---

nzumel

April 9, 2016 at 4:44 pm

Oddly enough, this doesn't happen with factors, which should have the same problem:

```
> tmp = as.factor(c('a', 'b', 'c'))
> for(t in tmp) print(t)
[1] "a"
[1] "b"
[1] "c"

> class(tmp)
```

```
[1] "factor"
> typeof(tmp)
[1] "integer"
```

Likely, it caused enough problems that they special-cased factors.

Comments are closed.