How can I make slot to be filled with multiple same-type objects in R?

Asked 10 years, 3 months ago Modified 5 years, 7 months ago Viewed 1k times

Part of R Language Collective

6

Let's say I want to define two classes classes, `Sentence` and `Word`. Each word object has a character string and a part of speech (pos). Each sentence contains some number of words and has an additional slot for data.

The `Word` class is straightforward to define.

```
wordSlots <- list(word = "character", pos = "character")
wordProto <- list(word = "", pos = "")
setClass("Word", slots = wordSlots, prototype = wordProto)
Word <- function(word, pos) new("Word", word=word, pos=pos)
```

Now I want to make a `Sentence` class which can contain some `Word`s and some numerical data.

If I define the `Sentence` class as so:

```
sentenceSlots <- list(words = "Word", stats = "numeric")
sentenceProto <- list(words = Word(), stats = 0)
setClass("Sentence", slots = sentenceSlots, prototype = sentenceProto)
```

Then the sentence can contain only one word. I could obviously define it with many slots, one for each word, but then it will be limited in length.

However, if I define the `Sentence` class like this:

```
sentenceSlots <- list(words = "list", stats = "numeric")
sentenceProto <- list(words = list(Word()), stats = 0)
setClass("Sentence", slots = sentenceSlots, prototype = sentenceProto)
```

it can contain as many words as I want, but the slot `words` can contain objects which are not of the class `Word`.

Is there a way to accomplish this? This would be similar to the C++ thing where you can have a vector of objects of the same type.

R r class object slot

Share

Improve this question

Follow

edited May 1, 2019 at 15:20

NelsonGon

13.3k ● 7 ● 31 ● 58

asked Sep 15, 2014 at 19:10

Will Beason

3,551 ● 2 ● 30 ● 50

I think my previous suggestion (which I deleted), is good. In sentence change it to a vector of words instead of a list of words. I don't do much OO programming in R, but I think that should work. – DMT Sep 15, 2014 at 19:40

It doesn't interpret it as a vector, but a list. With `words="vector"` and `x <- new("Sentence")`, `x@words <- c(Word(),Word(),3)` causes no error and makes `x@words` a list. – Will Beason Sep 15, 2014 at 19:51

understandable right? Because you have two elements of type Word and one of type numeric? It's going to be coerced before the setting even takes place. Does the 3 correspond to stats in the sentence object? – DMT Sep 15, 2014 at 20:00

it seems to me like you would want to set words in the sentence class by x@words<-c(Word(), Word()) and then stats as x@stats<-3, if I'm understanding what you're trying to do in your comment correctly – DMT Sep 15, 2014 at 20:02

1 A work-around could be to check the class of the components of the list in your `Sentence` constructor. See, as an example of this, the `Polygons` constructor of the `sp` package. Then you can redifine the `@<-` operators to avoid that the user set the `word` slot bypassing your constraints. – nicola Sep 15, 2014 at 20:33

2 Answers

Sorted by: Highest score (default)

7

Remembering that R works well on vectors, a first step is to think of 'Words' rather than 'Word'

```
## constructor, accessors, subset (also need [[, [<-, [[<- methods)
.Words <- setClass("Words",
    representation(words="character", parts="character"))
words <- function(x) x@words
parts <- function(x) x@parts
setMethod("length", "Words", function(x) length(words(x)))
setMethod("[", c("Words", "ANY", "missing"), function(x, i, j, ...) {
    initialize(x, words=words(x)[i], parts=parts(x)[i], ...)
})

## validity
setValidity("Words", function(object) {
    if (length(words(object)) == length(parts(object)))
        NULL
    else
        "'words()' and 'parts()' are not the same length"
})
```

@nicola's suggestion that one have a list of words has been formalized in the IRanges package (actually, S4Vectors in the 'devel' / 3.0 branch of Bioconductor), where a 'SimpleList' takes the 'naive' approach of requiring all elements of the list to have the same class, whereas a 'CompressedList' has similar behavior but actually is implemented as a vector-like object (one with a length(), [, and [[methods) that is 'partitioned' (either by end or width) into groups.

```
library(IRanges)
.Sentences = setClass("Sentences",
    contains="CompressedList",
    prototype=c(elementType="Words"))
```

One would then write a more user-friendly constructor, but the basic functionality is

```
## 0 Sentences
.Sentences()
## 1 sentence of 0 words
.Sentences(unlistData=.Words(), partitioning=PartitioningByEnd(0))
## 3 sentences of 2, 0, and 3 words
s3 <- .Sentences(unlistData=.Words(words=letters[1:5], parts=LETTERS[1:5]),
    partitioning=PartitioningByEnd(c(2, 2, 5)))
```

leading to

```
> s3[[1]]
An object of class "Words"
Slot "word":
[1] "a" "b"

Slot "part":
[1] "A" "B"

> s3[[2]]
An object of class "Words"
Slot "word":
character(0)

Slot "part":
character(0)

> s3[[3]]
An object of class "Words"
Slot "word":
[1] "c" "d" "e"

Slot "part":
[1] "C" "D" "E"
```

Notice that some typical operations are fast because they can operate on the 'unlisted' elements without creating or destroying S4 instances, e.g., coercing all 'words' to upper case

```
setMethod(toupper, "Words", function(x) { x@word <- toupper(x@word); x })
setMethod(toupper, "Sentences", function(x) relist(toupper(unlist(x)), x))
```

This is 'fast' for large collections of sentences because unlist / relist is really on a slot access and creation of a single instance of 'Words'. Scalable Genomics with R and Bioconductor outlines this and other strategies.

In an answer @nicola says 'R is not perfectly suited for OO programming style' but it's probably more helpful to realize that R's S4 object oriented style differs from C++ and Java, just as R differs from C. In particular it's really valuable to continue thinking in terms of vectors when working with S4 -- Words rather than Word, People rather than Person...

Share

Improve this answer

Follow

edited Sep 16, 2014 at 13:27

answered Sep 15, 2014 at 21:58

Martin Morgan

46.8k ● 7 ● 88 ● 115

> Does everything I needed to do, and more than I expected was easily possible. Very clear and informative. Gives me lots of ideas on tackling my problem! – Will Beason Sep 16, 2014 at 2:48

4

I suggest just a work-around for this class of problems. Keep in mind that R is not perfectly suited for OO programming style and every solution will hardly show the solidity of other languages like Java or C++. However, you can declare your `Sentence` class with a `words` slot as a list. Then you define your constructor as such:

```
   Sentence<-function(words,stats) {
     #check for the components' class of words argument
     if (!is.list(words) || !all(sapply(words,function(x) class(x)=="Word")))
stop("Not valid words argument")
     #create the object
      new("Sentence", words=words, stats=stats)
   }
```

An example of such constructor can be find in the `sp` package for the `Polygons` class. You can see the body of that function.

If you want to avoid that user sets incorrectly the `words` slot, you can redefine the `@<-` operator such like:

```
    "@<-.Sentence"<-function(sentence,...) invisible(sentence)
```

I don't think that the last step is necessary. No matter what you do, user can always mess things up. For instance, he could directly call the `new` function bypassing your constructor. Or he could set the `Word` class to an arbitrary object and then pass it to `Sentence`. As I said, R is not perfect for this style of programming, so you should often adopt some kind of non-optimal solution.

Share Improve this answer Follow

answered Sep 15, 2014 at 21:27

nicola
24.5k 3 36 57