DataFrameConstr

Jeffrey B.Arnold

March 17, 2013

Contents

DataFrameConst

This **R** package defines two S4 classes

- `HomogList`: a list in which all elements must be the same class
- `DataFrameConst`: a data frame with optional required columns and classes, or general constraints.

It also defines the most common methods `[<-`, `[[<-`, `$<-`, `c`, `cbind2`, `rbind2` for these classes so that the constraints are checked when data in the objects are updated.

Install

Install the latest version of the devtools **devtools** package, then

```
library("devtools")
install_github("ggthemes", "jrnold")
```

Classes

HomogList

Create a list in which all elements must be functions

```
library("DataFrameConstr")
foo <- HomogList(list(sum = sum, max = max, min = min),
    "function")
print(foo)
```

```
## List of "function" objects
## $sum
## function (..., na.rm = FALSE)  .Primitive("sum")
##
## $max
## function (..., na.rm = FALSE)  .Primitive("max")
##
## $min
## function (..., na.rm = FALSE)  .Primitive("min")
```

Since `HomogList` extends `list`, it can be used like any other list.

```
x <- 1:10
lapply(foo, function(f) f(x))
```

```
## $sum
## [1] 55
##
## $max
## [1] 10
##
## $min
## [1] 1
```

It can be updated,

```
foo[["mean"]] <- mean
print(foo)
```

```
## List of "function" objects
## $sum
## function (..., na.rm = FALSE)  .Primitive("sum")
```

```
##
## $max
## function (..., na.rm = FALSE)  .Primitive("max")
##
## $min
## function (..., na.rm = FALSE)  .Primitive("min")
##
## $mean
## function (x, ...)
## UseMethod("mean")
## <bytecode: 0x51d8278>
## <environment: namespace:base>
```

but the object will return an error if an element other than the specified class name is returned.

```
foo[["a"]] <- 1
```

```
## Error: invalid class "HomogList" object: Not all
## elements have class function
```

The methods `[<-`, `[[<-`, `$<-` and `c` are all defined to return `HomogList` objects, and by extension, check the class types of the elements in the new list.

The function `subclass_homog_list` can be used to create subclasses of `HomogList` for a specified class. The function `subclass_homog_list`, will create the class and all its associated methods, and return a function which creates new objects of that class.

For example, the following creates a new class “FunctionList”, in which all elements must be `function` objects.

```
FunctionList <- subclass_homog_list("FunctionList",
    "function")
```

```
## Error: package slot missing from signature for
## generic '[' and classes FunctionList, ANY, ANY
## cannot use with duplicate class names (the
## package may need to be re-installed)
```

Then a new object of class `FunctionList` can be created either by

```
FunctionList(list(sum = sum, mean = mean))
```

```
## Error: could not find function "FunctionList"
```

or, more verbosely,

```
new("FunctionList", list(sum = sum, mean = mean))
```

```
## An object of class "FunctionList"
## List of "function" objects
## $sum
## function (..., na.rm = FALSE)  .Primitive("sum")
##
## $mean
## function (x, ...)
## UseMethod("mean")
## <bytecode: 0x51d8278>
## <environment: namespace:base>
```

What is important about this class is that it will not accept any non-function elements either on creation,

```
FunctionList(list(a = 1))
```

```
## Error: could not find function "FunctionList"
```

or when updating an existing object,

```
foo <- FunctionList(list(sum = sum, mean = mean))
```

```
## Error: could not find function "FunctionList"
```

```
foo[["a"]] <- 1
```

```
## Error: invalid class "HomogList" object: Not all
## elements have class function
```

This makes classes extending `HomogList` particularly useful with S4 objects, either to define lists of S4 objects, or as the slot class for a class definition.

For example, in the **coda** package, `mcmc.list` is a S3 class consisting of a list of `mcmc` objects. An equivalent S4 class, which I'll call `NewMcmcList`, could be created with one function call, `r NewMcmcList <- subclass_homog_list("NewMcmcList", "mcmc.list")`

DataFrameConstr

The `DataFrameConstr` class extends the `data.frame` class, but allows for required columns with specified classes, and for general constraints on the `data.frame`.

For example, let's create a data frame which must have an `numeric` column named `"a"`, a column named `"b"`, which can be of any class, and a `"factor` column named `"c"`. Additionally, require that all values of `"a"` are positive.

```
foo <- DataFrameConstr(data.frame(a = runif(3), b = runif(3),
    c = letters[1:3]), columns = c(a = "numeric", b = "ANY",
    c = "factor"), constraints = list(function(x) {
    x$a > 0
}))
```

The new object `foo` acts just like any other `data.frame`,

```
print(foo)
```

```
##        a       b c
## 1 0.6748 0.47035 a
## 2 0.1708 0.79275 b
## 3 0.2248 0.02178 c
```

```
summary(foo)
```

```
##        a                b           c
##  Min.   :0.171   Min.   :0.0218   a:1
##  1st Qu.:0.198   1st Qu.:0.2461   b:1
##  Median :0.225   Median :0.4703   c:1
##  Mean   :0.357   Mean   :0.4283
##  3rd Qu.:0.450   3rd Qu.:0.6315
##  Max.   :0.675   Max.   :0.7927
```

However, it will validate updates to ensure that the data meets the specified constraints, This will return an error because `a` was defined as `numeric`,

```
foo$a <- as.character(foo$a)
```

```
## Error: invalid class "DataFrameConstr" object:
## column a does not inherit from numeric
```

This returns an error because `a` is constrained to be strictly positive,

```
foo["a", 1] <- -1
```

```
## Error: invalid class "DataFrameConstr" object:
## Constraint failed: function (x) { x$a > 0 }
```

```
# Unfortunately, this syntax, does not work, and
# alters foo foo[['a']][1] <- -1 I can't figure
# out how to avoid that, so if anyone knows, can
# you let me know?
```

This will not cause an error because the column `b` is allowed to have any class (more formally, it is of class "ANY"),

```
foo$b <- as.character(foo$b)
```

Since `foo` was created with `exclusive=FALSE` (by default) then the data frame can contain more rows than `a`, `b`, and `c`. The following is valid,

```
foo$d <- runif(3)
```

However, `foo` is guaranteed to always contain columns `a`, `b`, and `c`, and thus these columns cannot be deleted. This will return an error,

```
foo$a <- NULL
```

```
## Error: invalid class "DataFrameConstr" object:
## column a not in 'object'
```

The methods `[<-`, `[[<-`, `$<-`, `cbind2` (use instead of `cbind`), and `rbind2` (use instead of `rbind`), are defined so that they return `DataFrameConstr` objects, and by extension check the column classes, and constraints of the new object.

The function `constrained_data_frame` can be used to create subclasses of `DataFrameConstr`. For example, to create a class, which I'll call `"Foo"`, which has the same columns and constraints as the `foo` object previously created,

```
Foo <- constrained_data_frame("Foo", columns = c(a = "numeric",
    b = "ANY", c = "factor"), constraints = list(function(x) {
    x$a > 0
}))
```

```
## Error: trying to get slot "target" from an object
## of a basic class ("environment") with no slots
```

Now there is a new class, "Foo", which inherits from DataFrameConstr,

```
showClass("Foo")
```

```
## Class "Foo" [in ".GlobalEnv"]
##
## Slots:
##
## Name:                 .Data            columns
## Class:                 list          character
##
## Name:             exclusive        constraints
## Class:              logical       FunctionList
##
## Name:                 names          row.names
## Class:            character data.frameRowLabels
##
## Name:              .S3Class
## Class:            character
##
## Extends:
## Class "DataFrameConstr", directly
## Class "data.frame", by class "DataFrameConstr", distance 2
## Class "list", by class "DataFrameConstr", distance 3
## Class "oldClass", by class "DataFrameConstr", distance 3
## Class "vector", by class "DataFrameConstr", distance 4
```

Then create a new object, bar of class Foo,

```
bar <- Foo(data.frame(a = runif(3), b = runif(3), c = letters[1:3]))
```

This new object will validate any new data, so the following will produce errors,

```
bar[["a"]] <- as.character(bar[["a"]])
```

```
## Error: invalid class "DataFrameConstr" object:
## column a does not inherit from numeric
```

```
bar[["a"]] <- -1
```

```
## Error: invalid class "DataFrameConstr" object:
## Constraint failed: function (x) { x$a > 0 }
```

```
bar[["a"]] <- NULL
```

```
## Error: invalid class "DataFrameConstr" object:
## column a not in 'object'
```

This will validate the object on creation, so the following will return an error, because it does not contain the columns `b` or `c`,

```
Foo(data.frame(a = runif(3)))
```

```
## Error: invalid class "Foo" object: column b not
## in 'object'
```

The additional capabilities that `DataFrameConstr` adds to `data.frames` make it useful for the following,

- slot class types within S4 objects
- data validation
- creating an **R** ORM to databases