Advanced R - OO field guide

Anh Le

August 13, 2014

1 Base type

2 S3

2.1 Recognizing objects, generic functions, and methods

Most objects you encounter are S3 objects. Check by using `pryr::otype()`.

```
library(pryr)
df <- data.frame(x = 1:10, y = letters[1:10])
otype(df) # data frame is S3

## [1] "S3"

otype(df$x) # numeric vector is not

## [1] "base"

otype(df$y) # factor is S3

## [1] "S3"
```

In S3, methods belong to functions, called generic functions, or generics for short. S3 methods do not belong to objects or classes. This is different from most other programming languages, but is a legitimate OO style.

Given a class, the job of an S3 generic is to call the right S3 method. You can recognise S3 methods by their names, which look like `generic.class()`. For example, the Date method for the mean() generic is called `mean.Date()`, and the factor method for print() is called `print.factor()`.

2.2 Creating new methods and generics

```
f <- function(x) UseMethod("f") # creating generic
f.a <- function(x) "Class a" # creating method
```

```r
a <- structure(list(), class="a")
class(a)

## [1] "a"

f(a)

## [1] "Class a"

mean.a <- function(x) "a" # adding method to existing generic
mean(a)

## [1] "a"
```

2.3 Exercises

1. Read the source code for t() and t.test() and confirm that t.test() is an S3 generic and not an S3 method. What happens if you create an object with class test and call t() with it?

```r
t

## function (x)
## UseMethod("t")
## <bytecode: 0x2ce3250>
## <environment: namespace:base>

t.test

## function (x, ...)
## UseMethod("t.test")
## <bytecode: 0x3809870>
## <environment: namespace:stats>

obj_class_test <- structure(list(), class="test")
class(obj_class_test)

## [1] "test"

t(obj_class_test)

## Warning:  argument is not numeric or logical:  returning NA
## Error:  is.atomic(x) is not TRUE
```

It looks like `t(obj_class_test)` could not find `t.test` method so fall back to `t.default`.

2. `UseMethod()` calls methods in a special way. Predict what the following code will return, then run it and read the help for `UseMethod()` to figure out whats going on. Write down the rules in the simplest form possible.

```
y <- 1
g <- function(x) {
  y <- 2
  UseMethod("g")
}
g.numeric <- function(x) y
g(10) # 2

## [1] 2

h <- function(x) {
  x <- 10
  UseMethod("h")
}
h.character <- function(x) paste("char", x)
h.numeric <- function(x) paste("num", x)

h("a")

## [1] "char a"
```

3. Internal generics dont dispatch on the implicit class of base types. Carefully read `?"internal generic"` to determine why the length of f and g is different in the example below. What function helps distinguish between the behaviour of f and g?

```
f <- function() 1
g <- function() 2
class(g) <- "function"

class(f)

## [1] "function"

class(g)

## [1] "function"
```

```r
length.function <- function(x) "function"
length(f)

## [1] 1

length(g)

## [1] "function"
```

The difference comes from the fact that "For efficiency, internal dispatch only occurs on objects, that is those for which is.object returns true."

```r
is.object(f)

## [1] FALSE

is.object(g)

## [1] TRUE
```

Only `g` is an object. So only on `g` does `length` dispatch to `length.function`.