R S4 Class

R S4 Class

In this article, you'll learn everything about S4 classes in R; how to define them, create them, access their slots, and use them efficiently in your program.

Unlike

S3 classes

and

objects

which lacks formal definition, we look at S4 class which is stricter in the sense that it has a formal definition and a uniform way to create objects.

This adds safety to our code and prevents us from accidentally making naive mistakes.

How to define S4 Class?

S4 class is defined using the `setClass()` function.

In R terminology, member variables are called slots. While defining a class, we need to set the name and the slots (along with class of the slot) it is going to have.

Example 1: Definition of S4 class

```
setClass("student", slots=list(name="character", age="numeric", GPA="numeric"))
```

In the above example, we defined a new class called `student` along with three slots it's going to have `name`, `age` and `GPA`.

There are other optional arguments of `setClass()` which you can explore in the help section with `?setClass`.

How to create S4 objects?

S4 objects are created using the `new()` function.

Example 2: Creation of S4 object

```
> # create an object using new()
> # provide the class name and value for slots
> s <- new("student",name="John", age=21, GPA=3.5)
> s
An object of class "student"
Slot "name":
[1] "John"
Slot "age":
[1] 21
Slot "GPA":
[1] 3.5
```

We can check if an object is an S4 object through the function `isS4()`.

```
> isS4(s)
[1] TRUE
```

The function `setClass()` returns a generator function.

This generator function (usually having same name as the class) can be used to create new objects. It acts as a constructor.

```
> student <- setClass("student", slots=list(name="character", age="numeric", GPA="numeric"))
> student
class generator function for class “student” from package ‘.GlobalEnv’
function (...)
new("student", ...)
```

Now we can use this constructor function to create new objects.

Note above that our constructor in turn uses the `new()` function to create objects. It is just a wrap around.

Example 3: Creation of S4 objects using generator function

```
> student(name="John", age=21, GPA=3.5)
An object of class "student"
Slot "name":
[1] "John"
Slot "age":
[1] 21
Slot "GPA":
[1] 3.5
```

How to access and modify slot?

Just as components of a list are accessed using `$`, slot of an object are accessed using `@`.

Accessing slot

```
> s@name
[1] "John"
> s@GPA
[1] 3.5
> s@age
[1] 21
```

Modifying slot directly

A slot can be modified through reassignment.

```
> # modify GPA
> s@GPA <- 3.7
> s
An object of class "student"
Slot "name":
[1] "John"
Slot "age":
[1] 21
Slot "GPA":
[1] 3.7
```

Modifying slots using slot() function

Similarly, slots can be access or modified using the `slot()` function.

```
> slot(s,"name")
[1] "John"
> slot(s,"name") <- "Paul"
> s
An object of class "student"
Slot "name":
[1] "Paul"
Slot "age":
[1] 21
Slot "GPA":
[1] 3.7
```

Methods and Generic Functions

As in the case of S3 class, methods for S4 class also belong to generic functions rather than the class itself. Working with S4 generics is pretty much similar to S3 generics.

You can list all the S4 generic functions and methods available, using the function `showMethods()`.

Example 4: List all generic functions

```
> showMethods()
Function: - (package base)
Function: != (package base)
...
Function: trigamma (package base)
Function: trunc (package base)
```

Writing the name of the object in interactive mode prints it. This is done using the S4 generic function `show()`.

You can see this function in the above list. This function is the S4 analogy of the S3 `print()` function.

Example 5: Check if a function is a generic function

```
> isS4(print)
[1] FALSE
> isS4(show)
[1] TRUE
```

We can list all the methods of show generic function using `showMethods(show)`.

Example 6: List all methods of a generic function

```
> showMethods(show)
Function: show (package methods)
object="ANY"
object="classGeneratorFunction"
...
object="standardGeneric"
(inherited from: object="genericFunction")
object="traceable"
```

How to write your own method?

We can write our own method using `setMethod()` helper function.

For example, we can implement our class method for the `show()` generic as follows.

```
setMethod("show",
"student",
function(object) {
cat(object@name, "\n")
cat(object@age, "years old\n")
cat("GPA:", object@GPA, "\n")
}
)
```

Now, if we write out the name of the object in interactive mode as before, the above code is executed.

```
> s <- new("student",name="John", age=21, GPA=3.5)
> s    # this is same as show(s)
John
21 years old
GPA: 3.5
```

In this way we can write our own S4 class methods for generic functions.