[S4 object system][http://adv-r.had.co.nz/S4.html]

R拥有三种面向对象系统(OO sytems)：S3/S4/R5

S4不同与S3：正式的类别定义，S4正式定义了每个类别的representation和inheritance；多重dispatch，generic函数能够通过任意数目的类别参数分配到一个method，而不是S3中的一个class类别

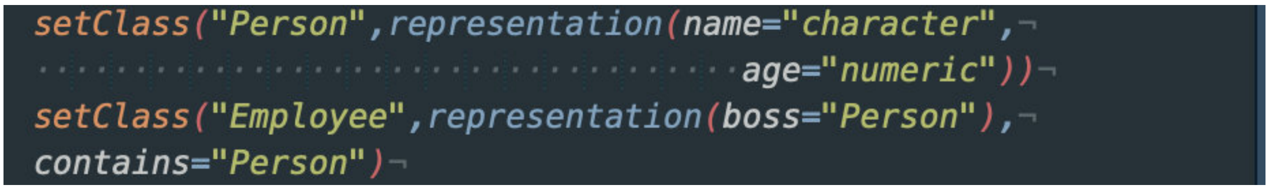
Classes and instances

在S3中可通过设置class属性来将任一对象的类别修改为其他类别，而S4更严格，必须使用`setClass`定义call的representation，同时instance(实例)唯一的构建方式就是通过构建函数new。

一个S4类别包含三个关键特征：

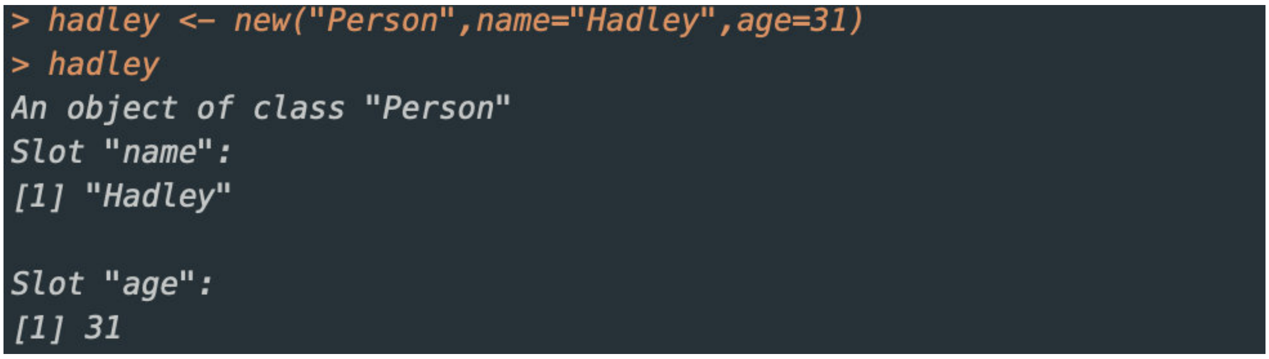
- 名称，字母字符串指明类别名称
- **Representation(代理)，一系列slots(attributes, 属性)，定义其名称和类别。例如，一个person类别可能通过字符名称和数字年龄来代理: representation(name="character",age="numeric")**
- **类别的字符串向量代表它所继承的形式，或称为contains。值得注意的是，S4支持多重继承。**

使用`setClass`构建类别(contains)：



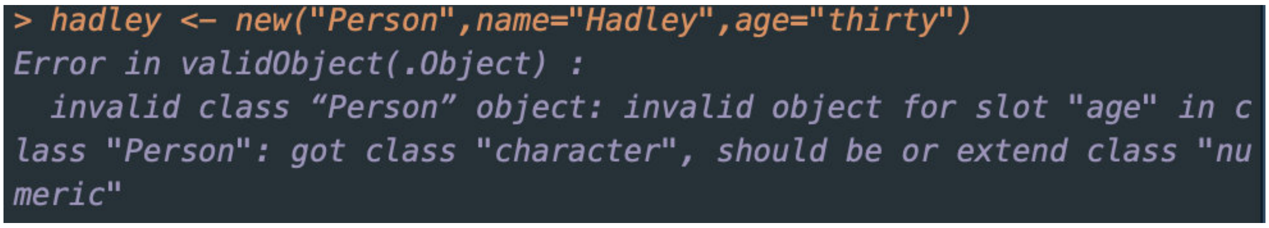
```
setClass("Person",representation(name="character",
                                 age="numeric"))
setClass("Employee",representation(boss="Person"),
contains="Person")
```

使用`new`构建该类别的实例：



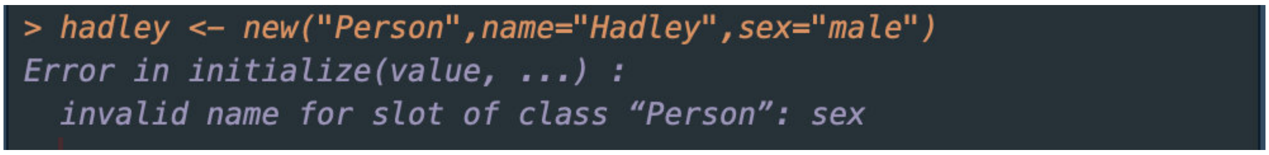
```
> hadley <- new("Person",name="Hadley",age=31)
> hadley
An object of class "Person"
Slot "name":
[1] "Hadley"

Slot "age":
[1] 31
```

不同于S3, S4在使用`new`构建实例时会检查所有的slots的类型是否正确：



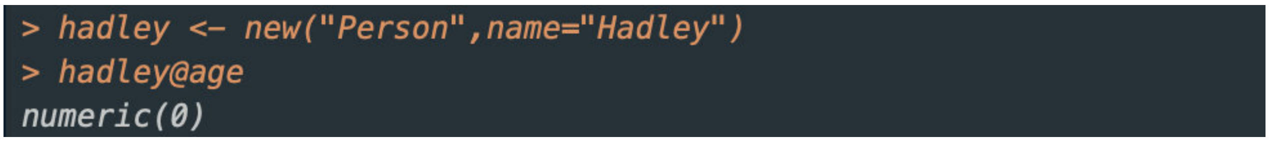
```
> hadley <- new("Person",name="Hadley",age="thirty")
Error in validObject(.Object) :
  invalid class “Person” object: invalid object for slot "age" in c
lass "Person": got class "character", should be or extend class "nu
meric"
```

同时查看slots：



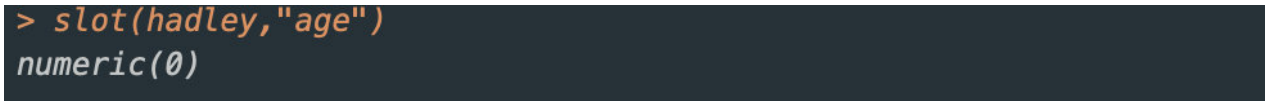
```
> hadley <- new("Person",name="Hadley",sex="male")
Error in initialize(value, ...) :
  invalid name for slot of class “Person”: sex
```

若省略一个`slot`，将会默认采用该类别的初始值，使用@访问S4对象的slots：



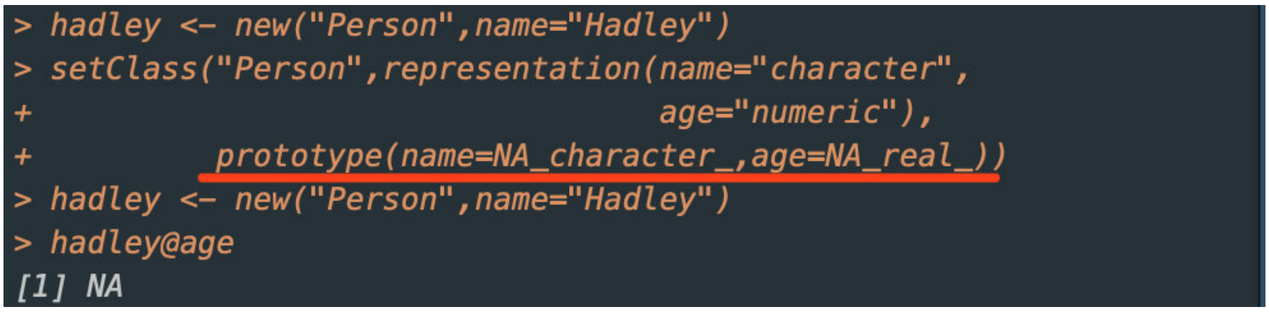
```
> hadley <- new("Person",name="Hadley")
> hadley@age
numeric(0)
```

或直接使用`slot`函数：



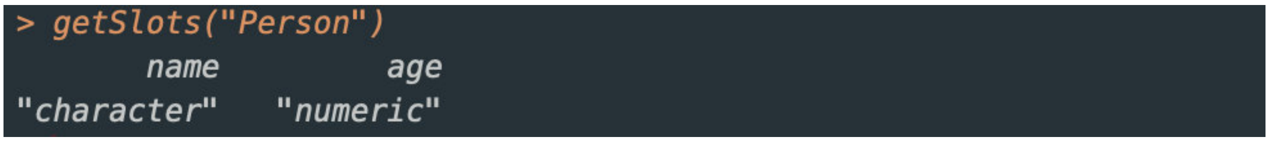
```
> slot(hadley,"age")
numeric(0)
```

同时避免空slots的默认值出现，可在class中设置默认的`prototype`：



```
> hadley <- new("Person",name="Hadley")
> setClass("Person",representation(name="character",
+                                  age="numeric"),
+          prototype(name=NA_character_,age=NA_real_))
> hadley <- new("Person",name="Hadley")
> hadley@age
[1] NA
```

函数`getSlots`返回一个类别的所有slots描述：

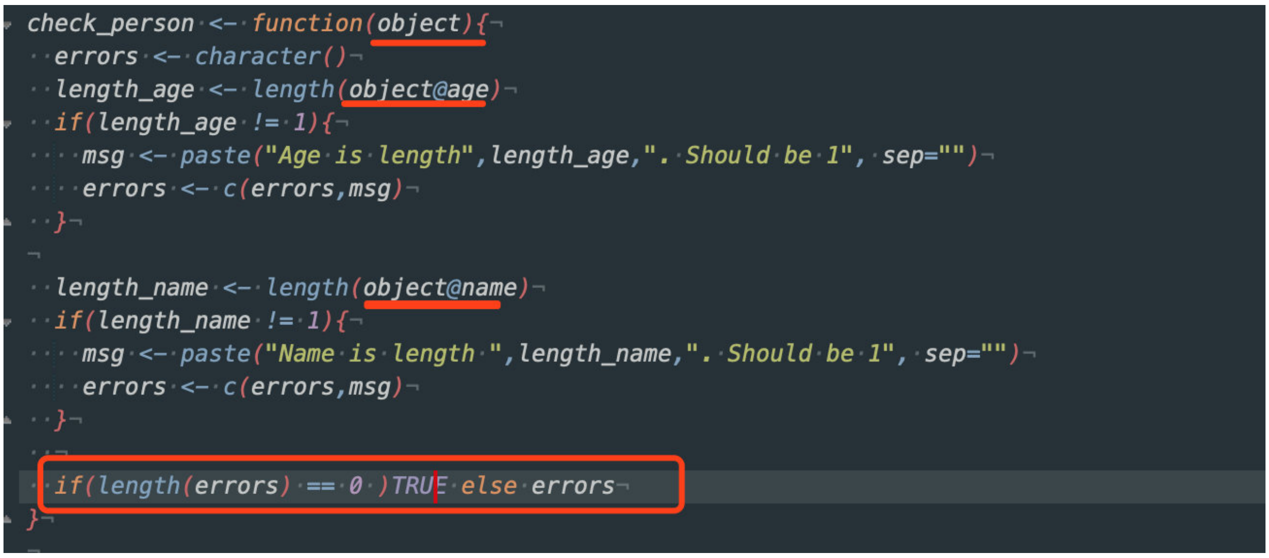


```
> getSlots("Person")
       name         age
"character"   "numeric"
```

Notes: In particular, note that the examples rely on the fact that multiple calss to `setClass` with the same class name will silently override the previous definition unless the first definition is sealed with `sealed=TRUE`.

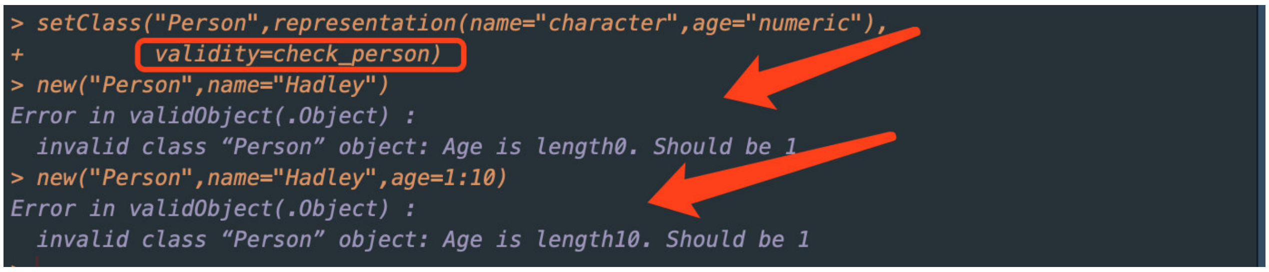
Checking validity

提供可选function增加限制，该函数应含一个参数object，同时有效情况下返回TRUE，无效时返回字符向量给出错误原因：



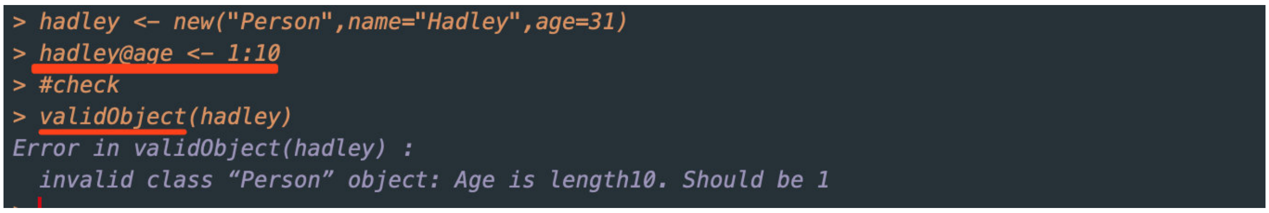
```r
check_person <- function(object){
  errors <- character()
  length_age <- length(object@age)
  if(length_age != 1){
    msg <- paste("Age is length",length_age,". Should be 1", sep="")
    errors <- c(errors,msg)
  }

  length_name <- length(object@name)
  if(length_name != 1){
    msg <- paste("Name is length ",length_name,". Should be 1", sep="")
    errors <- c(errors,msg)
  }

  if(length(errors) == 0 )TRUE else errors
}
```

例如：



```
> setClass("Person",representation(name="character",age="numeric"),
+          validity=check_person)
> new("Person",name="Hadley")
Error in validObject(.Object) :
  invalid class “Person” object: Age is length0. Should be 1
> new("Person",name="Hadley",age=1:10)
Error in validObject(.Object) :
  invalid class “Person” object: Age is length10. Should be 1
```

同时当直接修改slot值时，该判断不会自动进行，调用validObject函数：



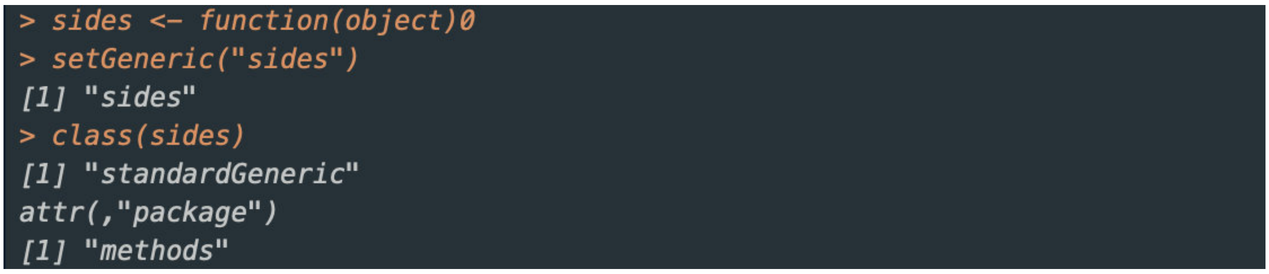
```
> hadley <- new("Person",name="Hadley",age=31)
> hadley@age <- 1:10
> #check
> validObject(hadley)
Error in validObject(hadley) :
  invalid class “Person” object: Age is length10. Should be 1
```

Generic functions and methods

generic functions and methods work similarly to S3, but dispath is based on the class of all arguments, and there is a special syntax for creating both generic functions and new methods.

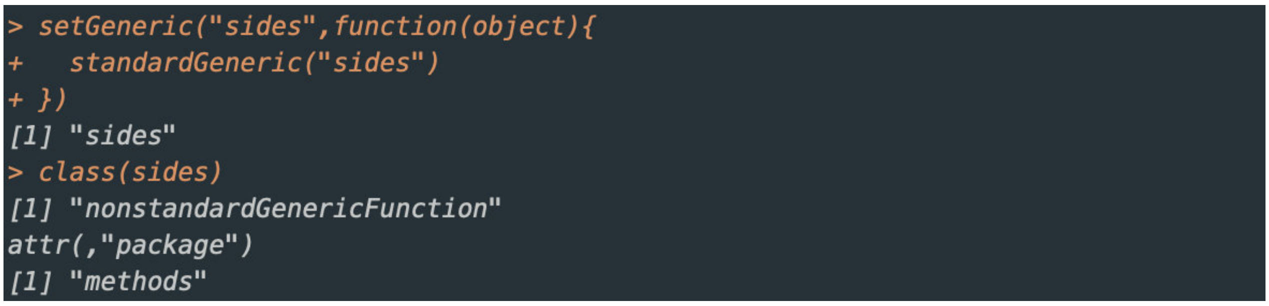
The `setGeneric` function is called to initialize a generic function as preparation for defining some methods for that function.

函数`setGeneric`提供两种方式创建**新的generic**：



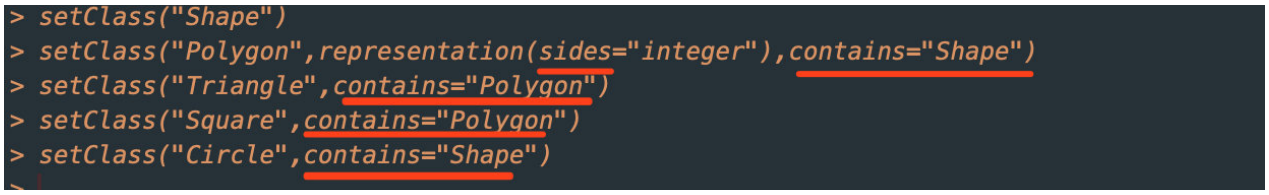
```
> sides <- function(object)0
> setGeneric("sides")
[1] "sides"
> class(sides)
[1] "standardGeneric"
attr(,"package")
[1] "methods"
```

若直接构建，第二个参数应该为一函数定义所有的参数：



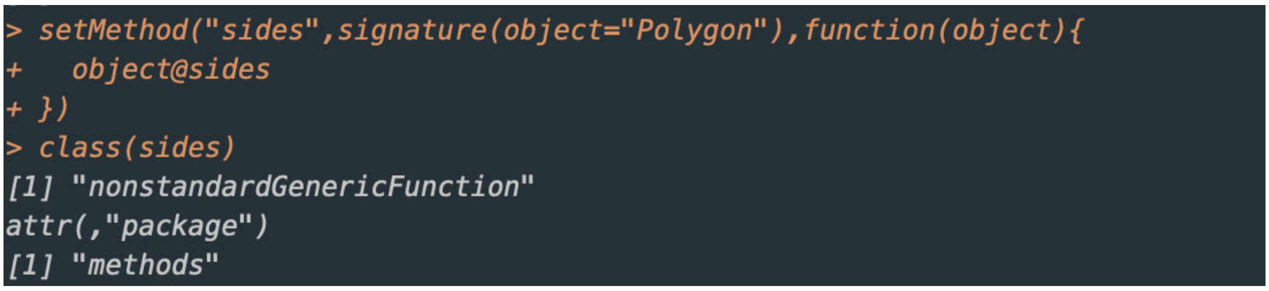
```
> setGeneric("sides",function(object){
+   standardGeneric("sides")
+ })
[1] "sides"
> class(sides)
[1] "nonstandardGenericFunction"
attr(,"package")
[1] "methods"
```

继承sides function：



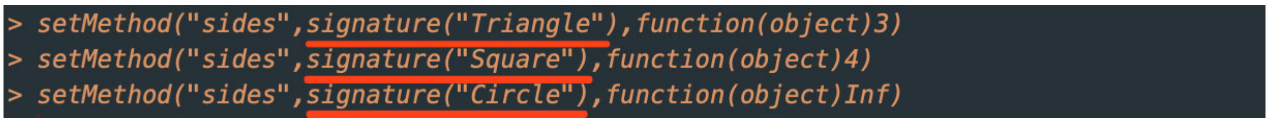
```
> setClass("Shape")
> setClass("Polygon",representation(sides="integer"),contains="Shape")
> setClass("Triangle",contains="Polygon")
> setClass("Square",contains="Polygon")
> setClass("Circle",contains="Shape")
```

使用sides slot直接定义polygon的method，`setMethod`函数拥有3个参数：genric function的名称，匹配给method的signature，和计算结果的函数



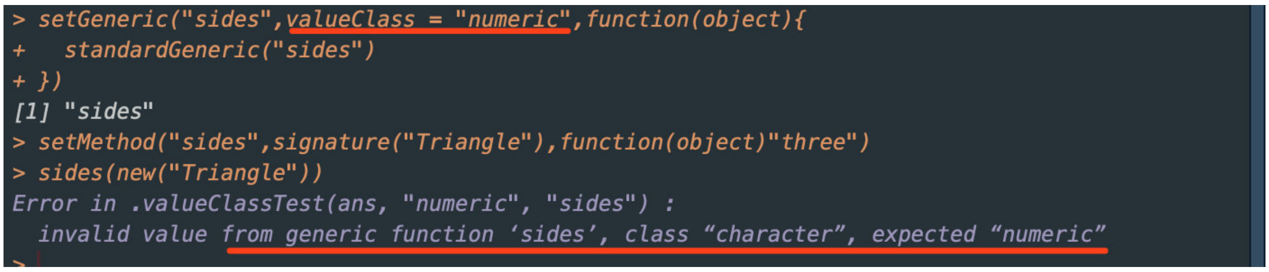
```
> setMethod("sides",signature(object="Polygon"),function(object){
+   object@sides
+ })
> class(sides)
[1] "nonstandardGenericFunction"
attr(,"package")
[1] "methods"
```

针对含有少量参数的generic可直接给定`signature`参数的值，这将省略空间同时位置要对应参数值：



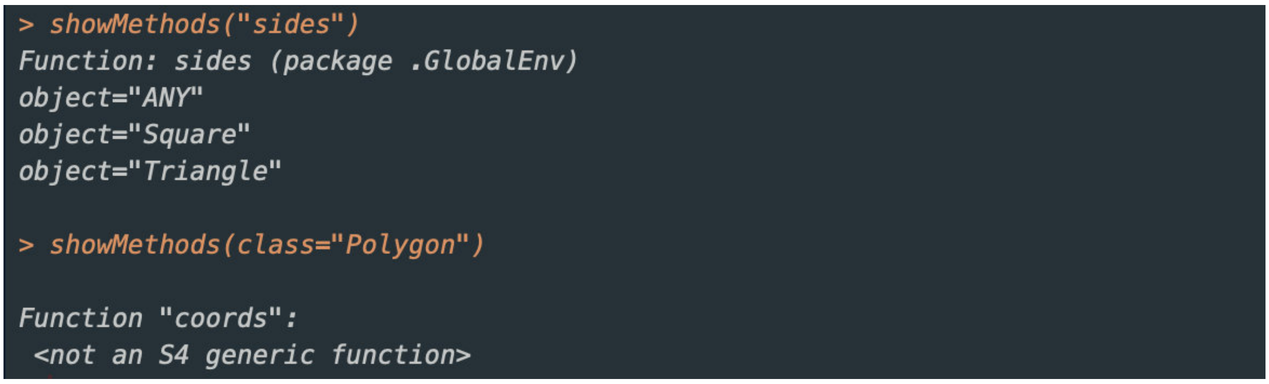
```
> setMethod("sides",signature("Triangle"),function(object)3)
> setMethod("sides",signature("Square"),function(object)4)
> setMethod("sides",signature("Circle"),function(object)Inf)
```

可通过指定`valueClass`来定义generic的输出：



```
> setGeneric("sides",valueClass = "numeric",function(object){
+   standardGeneric("sides")
+ })
[1] "sides"
> setMethod("sides",signature("Triangle"),function(object)"three")
> sides(new("Triangle"))
Error in .valueClassTest(ans, "numeric", "sides") :
  invalid value from generic function ‘sides’, class “character”, expected “numeric”
```

使用函数`showMethods`查看定义了的generic function的methods：

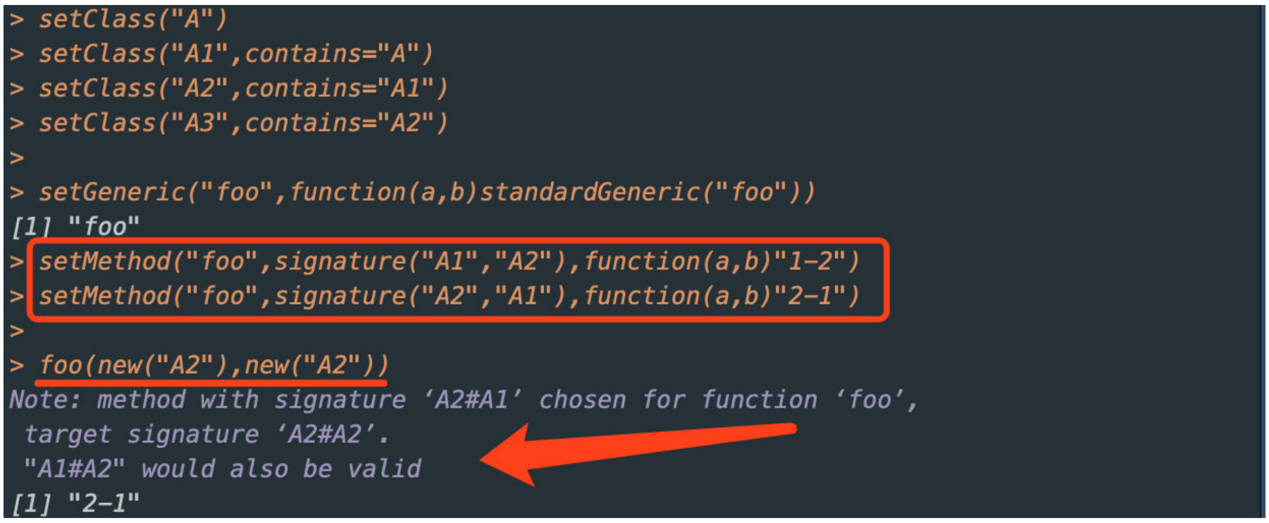


```
> showMethods("sides")
Function: sides (package .GlobalEnv)
object="ANY"
object="Square"
object="Triangle"

> showMethods(class="Polygon")

Function "coords":
 <not an S4 generic function>
```

Method dispatch

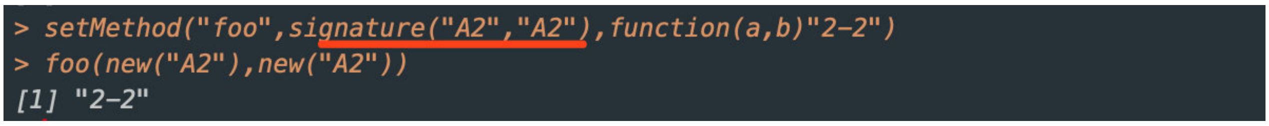
调用(call)generic function的正确method，若在调用(call)中的objects的class之间，和method的signature之间存在精确匹配，那么就调用该methods。否则：

- 对于调用function的每一个参数，计算class之间，和signature中的class的距离。若相同，距离为0，若class的signature为调用class的parent，距离为1；若为grandparent，距离为2，依次类推。
- 计算每一个method的距离，若存在唯一最小距离的method，直接采用。否则，给出警告信息同时采用匹配methods中的一个(按照字母顺序)。



```
> setClass("A")
> setClass("A1",contains="A")
> setClass("A2",contains="A1")
> setClass("A3",contains="A2")
>
> setGeneric("foo",function(a,b)standardGeneric("foo"))
[1] "foo"
> setMethod("foo",signature("A1","A2"),function(a,b)"1-2")
> setMethod("foo",signature("A2","A1"),function(a,b)"2-1")
>
> foo(new("A2"),new("A2"))
Note: method with signature ‘A2#A1’ chosen for function ‘foo’,
 target signature ‘A2#A2’.
 "A1#A2" would also be valid
[1] "2-1"
```

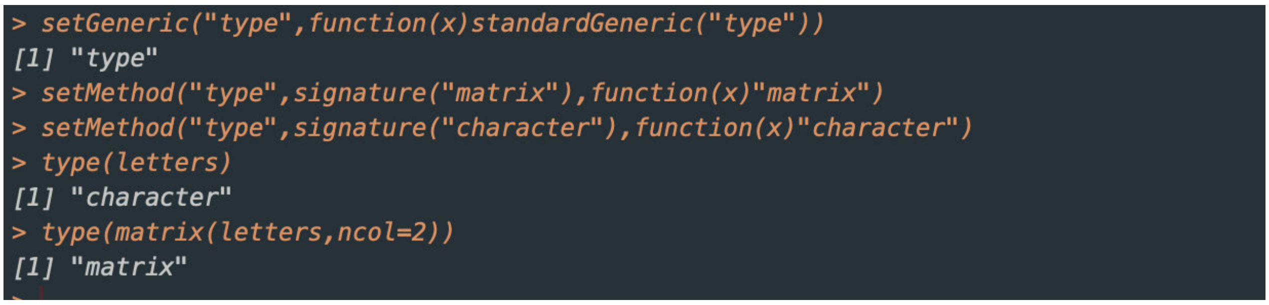
通常，为避免这类问题，需提供更特定的method：



```
> setMethod("foo",signature("A2","A2"),function(a,b)"2-2")
> foo(new("A2"),new("A2"))
[1] "2-2"
```

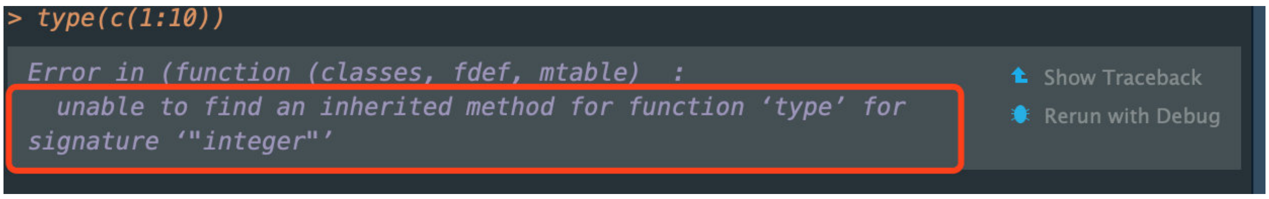
有两个特殊的class可以用于`signature`:`missing`和`ANY`。`missing`匹配未提供参数的情况(argument is not supplied)，`ANY`用于设定默认methods，同时`ANY`拥有最低的顺序用于method匹配(lowest possible precedence in method matching)。

同样可以使用basic classes，例如`numeric`,`character`和`matrix`。A matrix of characters will have class `matrix`。



```
> setGeneric("type",function(x)standardGeneric("type"))
[1] "type"
> setMethod("type",signature("matrix"),function(x)"matrix")
> setMethod("type",signature("character"),function(x)"character")
> type(letters)
[1] "character"
> type(matrix(letters,ncol=2))
[1] "matrix"
```

另外：



```
> type(c(1:10))

 Error in (function (classes, fdef, mtable)  :
   unable to find an inherited method for function ‘type’ for
 signature ‘"integer"’
```

也可以在S4中dispatch S3 classes，通过calling setOldClass：

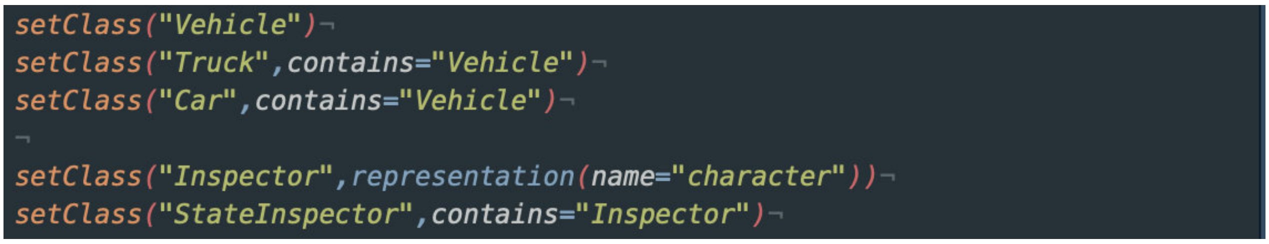
```
foo <- structure(list(x = 1), class = "foo")
type(foo)

setOldClass("foo")
setMethod("type", signature("foo"), function(x) "foo")

type(foo)

setMethod("+", signature(e1 = "foo", e2 = "numeric"), function(e1, e2) {
  structure(list(x = e1$x + e2), class = "foo")
})
foo + 3
```

Inheritance

构建简单的vehicle inspections model，该model根据vehicle(car/truck)的类型不同和inspector(normal/state)不同调用不同的输出。

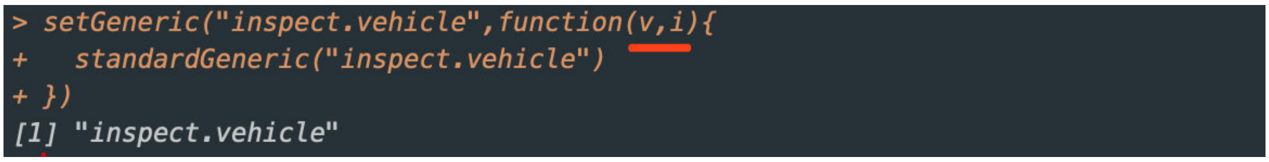
在S4中，`callNextMethod`函数用于调用下一个method，通过假设当前method不存在，寻找最近的下一个匹配的method

首先设置classes，2 vehicle类型和2 inspect类型：



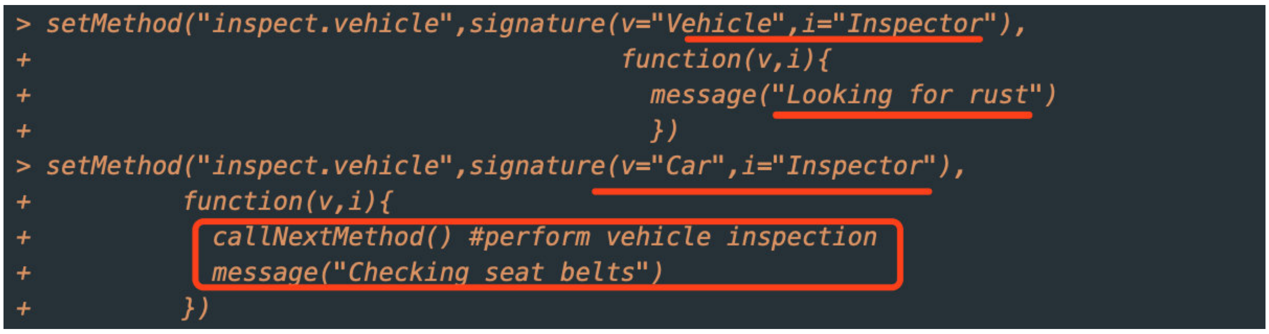
```
setClass("Vehicle")¬
setClass("Truck",contains="Vehicle")¬
setClass("Car",contains="Vehicle")¬
¬
setClass("Inspector",representation(name="character"))¬
setClass("StateInspector",contains="Inspector")¬
```

接着，定义generic function用于检查vehicle，拥有两个参数，被检查的vehicle和检查的人:



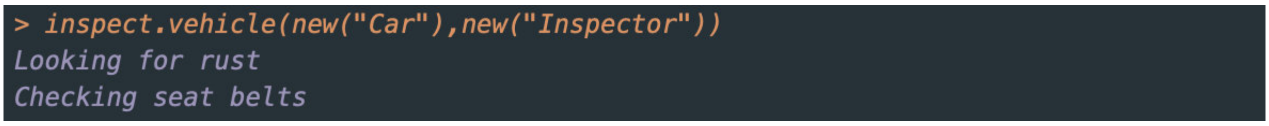
```
> setGeneric("inspect.vehicle",function(v,i){
+   standardGeneric("inspect.vehicle")
+ })
[1] "inspect.vehicle"
```

所有的vehicle都要被所有检查人查看生锈情况，同时Cars需要检查其安全带:



```
> setMethod("inspect.vehicle",signature(v="Vehicle",i="Inspector"),
+                                        function(v,i){
+                                          message("Looking for rust")
+                                          })
> setMethod("inspect.vehicle",signature(v="Car",i="Inspector"),
+           function(v,i){
+             callNextMethod() #perform vehicle inspection
+             message("Checking seat belts")
+           })
```

执行Cars/Inspector:



```
> inspect.vehicle(new("Car"),new("Inspector"))
Looking for rust
Checking seat belts
```

怎加针对trucks的检查，同时指定state inspector查看Cars的保险:

```
> setMethod("inspect.vehicle",
+           signature(v="Truck",i="Inspector"),
+           function(v,i){
+             callNextMethod()
+             message("Checking cargo attachments")
+           })
> inspect.vehicle(new("Truck"),new("Inspector"))
Looking for rust
Checking cargo attachments
```

以及:

```
> setMethod("inspect.vehicle",
+           signature(v = "Car", i = "StateInspector"),
+           function(v, i) {
+             callNextMethod()
+             message("Checking insurance")
+           })
> inspect.vehicle(new("Car"),new("StateInspector"))
Looking for rust
Checking seat belts
Checking insurance
```

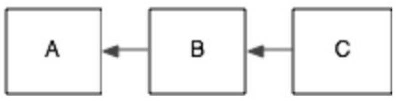
```
> inspect.vehicle(new("Truck"),new("StateInspector"))
Looking for rust
Checking cargo attachments
```

Method dispatch2

构建一个简单的class结构，有3个classes，class C继承一个字符向量，B继承C，A继承B。

```
> setClass("C",contains="character")
> setClass("B",contains="C")
> setClass("A",contains="B")
> a <- new("A","a")
> b <- new("B","b")
> c <- new("C","c")
```

构建的class关系如下:


$$\begin{tikzcd} \boxed{A} & \boxed{B} \arrow[l] & \boxed{C} \arrow[l] \end{tikzcd}$$

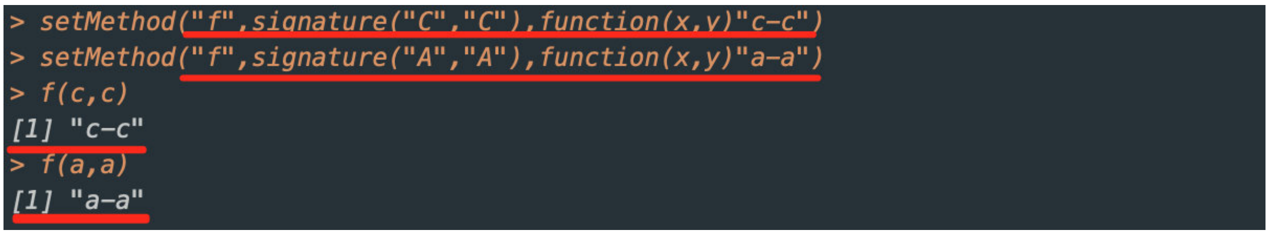
构建generic f，同时dispatch两个参数: x和y

```
> setGeneric("f",function(x,y)standardGeneric("f"))
[1] "f"
```

为预测generic将会使用的method，需要知道:

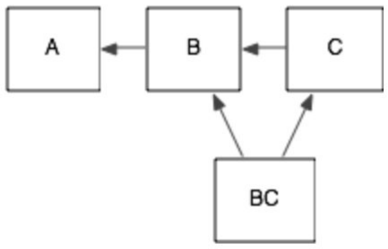
- the name and arguments to the generic
- the signatures of the methods
- the class of arguments supplied to the generic

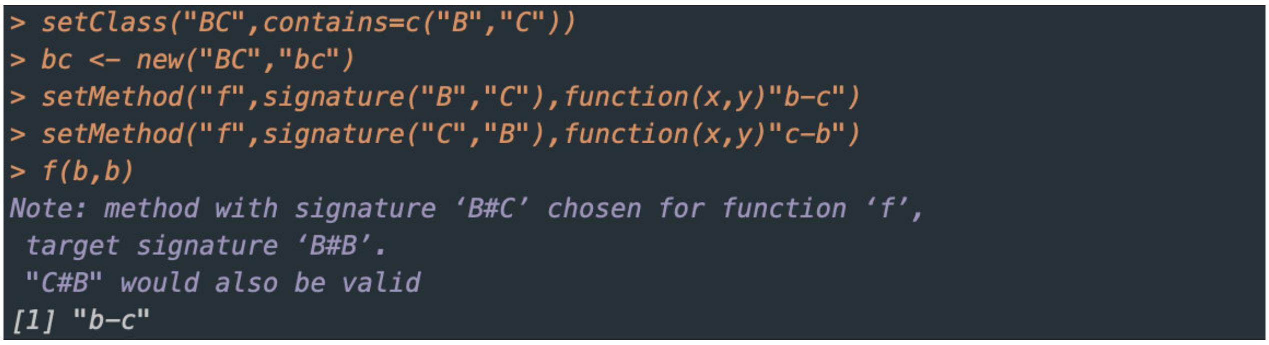
最简单method dispath是准确的匹配: exact match between the class of arguments(arg-classes) and the signature of (sig-classes):



```
> setMethod("f",signature("C","C"),function(x,y)"c-c")
> setMethod("f",signature("A","A"),function(x,y)"a-a")
> f(c,c)
[1] "c-c"
> f(a,a)
[1] "a-a"
```

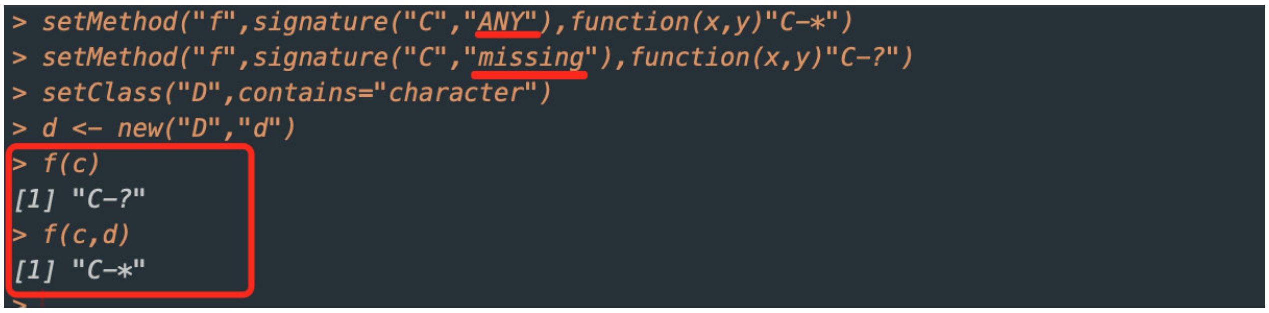
如果我们增加另一个class, BC, that inherited from both B and C, then this class would have distance one to both B and C, and distance two to A:





```
> setClass("BC",contains=c("B","C"))
> bc <- new("BC","bc")
> setMethod("f",signature("B","C"),function(x,y)"b-c")
> setMethod("f",signature("C","B"),function(x,y)"c-b")
> f(b,b)
Note: method with signature ‘B#C’ chosen for function ‘f’,
 target signature ‘B#B’.
 "C#B" would also be valid
[1] "b-c"
```

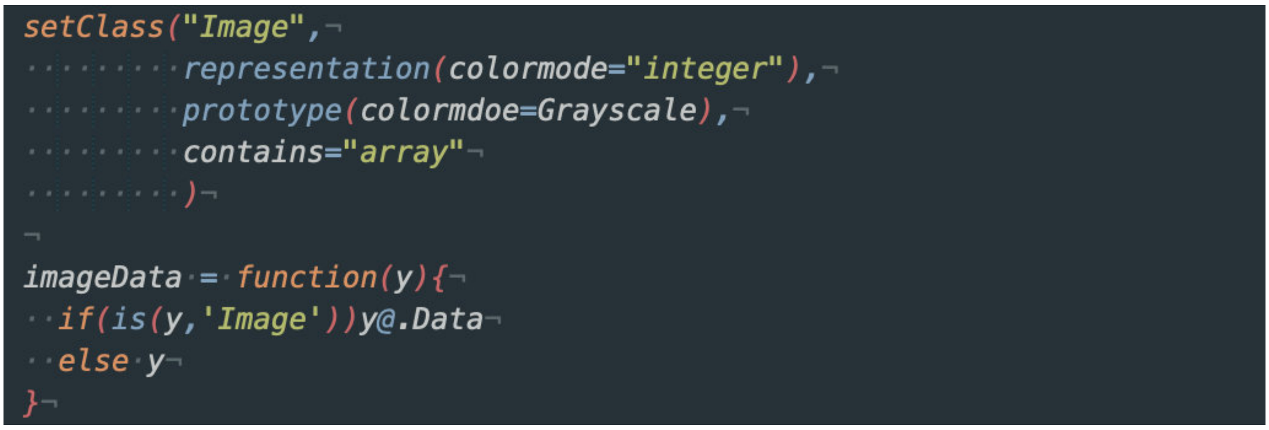
两个特殊的classes可用于signature: missing和ANY。missing匹配未提供的参数，ANY用于设定默认methods，同时ANY拥有最小的优先级用于method匹配:



```
> setMethod("f",signature("C","ANY"),function(x,y)"C-*")
> setMethod("f",signature("C","missing"),function(x,y)"C-?")
> setClass("D",contains="character")
> d <- new("D","d")
> f(c)
[1] "C-?"
> f(c,d)
[1] "C-*"
>
```

In the wild

To conclude, lets look at some S4 code in practice. The Bioconductor EBImage, it has only one class, an Image, which represents an image as an array of pixel values.



```r
setClass("Image",
         representation(colormode="integer"),
         prototype(colormdoe=Grayscale),
         contains="array"
          )

imageData = function(y){
  if(is(y,'Image'))y@.Data
  else y
}
```

略！！！