Une vie polyamoureuse entre R et Julia

Rémy Drouilhet*

Résumé

Le langage `julia` partage avec le langage `R` les caractéristiques comme l'indexation des tableaux commençant à 1, le (*multiple*) *dispatching*, la *metaprogramming* et son système unique de gestion des librairies (paquets). A la différence de `R`, `julia` proposant une compilation JIT (*Just In Time*) est intrinsèquement plus performant que le langage `R`, rétablissant au passage l'utilisation des boucles `for` comme c'est le cas pour les langages compilés. De par sa jeunesse (un peu plus de 10 ans), `julia` reste toutefois un langage en devenir surtout au niveau du développement de son écosystème de paquets. Pour toutes ces raisons, le langage `julia` peut être vu comme un digne successeur du langage `R`. Dans cette présentation, nous proposons le paquet `R`, nommé `jl4R`, dont l'objectif avoué est de téléguider depuis `R` des paquets `julia`. L'esprit du paquet est principalement d'imaginer le `julia` comme un remplacement de `Rcpp` et ainsi de proposer des paquets `R` de type *wrapper* de paquet `julia`.
Mots-clefs : Julia – *Multiple Dispatching* – Paquet `R` – *wrapper* de paquet `julia`

Démarrage rapide

Rien de mieux pour présenter le paquet `jl4R` [**1**] que de commencer par une utilisation basique.

```
> require(jl4R)
> ## conversion R vers julia
> v_jl <- jl(c(1,3,2))
> v_jl
3-element Vector{Float64}:
 1.0
 3.0
 2.0
> jltypeof(v_jl)
Vector{Float64} (alias for Array{Float64, 1})
> length(v_jl)
[1] 3
> v_jl[2]
3.0
> jltypeof(v_jl[2])
Float64
> R(v_jl) # ou toR(v_jl)
[1] 1 3 2

> ## exécution directe code julia
> v2_jl <- jl(‘[1,3,2]‘)
> v2_jl
3-element Vector{Int64}:
 1
```

*Laboratoire Jean Kuntzmann, `Remy.Drouilhet@univ-grenoble-alpes.fr`

```
 3
 2
> jltypeof(v2_jl)
Vector{Int64} (alias for Array{Int64, 1})
> jltypeof(v2_jl[2])
Int64
> toR(v2_jl)
[1] 1 3 2
```

Après avoir chargé le paquet, nous créons une instance `julia` obtenue par autoconversion d'un vecteur `R` dont la sortie est celle proposée par `julia`. La fonction `jltypeof()` retourne le type `julia`, ici un `Array{Float64}`. Après avoir obtenu sa longueur, on extrait son deuxième élément dont on fournit le type `julia`. La fonction `jl()` permet aussi d'éxecuter directement du code `julia` comme illustrée avec la variable `v2_jl`. Remarquons au passage que le type des éléments de `v2_jl` sont des `Int64`. La fonction `R()` (ou `toR()`) retourne le résultat d'une instance `julia` converti en objet `R` si possible.

En interne, une instance `julia` est représentée par un pointeur externe (`externalptr`) dont la classe est dans l'ordre le type `julia` puis `jlValue`, comme le montre la sortie ci-dessous.

```
> typeof(v_jl)
[1] "externalptr"
> class(v_jl)
[1] "Array"   "jlvalue"
```

Fonctionnalités principales du paquet

N'ayant pas la place nécessaire dans ce résumé d'illustrer par des exemples les fonctionnalités principales du paquet (cela fera plutôt l'objet de la présentation), nous en proposons une liste non exhaustive :

1. la fonction principale `jl()` permettant :
   (a) lorsque l'argument est une expression `julia` entre deux caractères ` (*backtick* en anglais) : d'exécuter du code `julia` et de créer une instance `julia`.
   (b) lorsque l'argument est un objet `R` dont la conversion est gérée par le paquet : créer l'instance `julia` correspondante.
2. un unique objet `jl` défini dans l'environnement global de `R` qui permet de créer et extraire des variables dans le module `Main` de `julia`
3. une conversion de `R` vers `julia` des vecteurs, matrices de données, facteurs.
4. une conversion de `julia` vers `R` des `Array`, `Tuple`, `DataFrame` et `CategoricalArray`.
5. synchronisation entre les ramasses-miettes (*garbage collectors*) de `julia` et `R`.

Commentaires

Le paquet `jl4R` est en phase de développement et est loin d'être considéré comme stable. Le paquet pourrait notamment dans l'avenir changer de nom. Comme le but avoué du paquet est de remplacer l'utilisation de `Rcpp`, il n'en dépend surtout pas. Un cas d'usage serait notamment le développement de *wrappers* de paquets `julia` et notamment un pour remplacer le paquet `VAM` (*Virtual Age Models*) qui repose actuellement sur `Rcpp`. Le nom du paquet `R` serait `VirtualAgeModels_jl` et permettrait de rendre accessible dans le système `R` le paquet `VirtualAgeModels.jl` en cours de développement et déjà installable dans l'environnement `julia` via le paquet `Pkg.jl`.

Notons aussi qu'il existe déjà un paquet `R` du même type `JuliaCall` [**2**] qui permet déjà de proposer des paquets `R` de type *wrapper* de paquet `julia`. L'exemple le plus notable est le paquet `diffeqr` [**3**] qui permet d'utiliser le paquet `DifferentialEquations.jl` directement depuis `R`. L'auteur de `diffeqr` [**3**] fournit dans le même esprit `diffeqpy` [**4**] un paquet `python` permettant d'utiliser le paquet `DifferentialEquations.jl` en `python`.

Cependant, à la différence de notre paquet, le paquet `JuliaCall` [**5**] :

1. dépend de `Rcpp` quand notre paquet repose uniquement sur l'API `C` de `R`
2. nécessite l'installation du paquet `RCall.jl`

Remerciements

Ce travail a été partiellement financé par l'Agence National Française pour la Recherche dans le cadre du programme France 2030 (ANR-15-IDEX-0002) et par le LabEx PERSYVAL-Lab (ANR-11-LABX-0025-01).

Références

[**1**] `jl4R`, paquet `R`, `https://github.com/rcqls/jl4R`
[**5**] `JuliaCall`, paquet `R`, `https://github.com/Non-Contradiction/JuliaCall`
[**3**] `diffeqr`, paquet `R`, `https://github.com/SciML/diffeqr`
[**4**] `diffeqpy`, paquet `python`, `https://github.com/SciML/diffeqpy`