gestion chemin here/pathological

laude

05/05/2019

Gestion des chemins

Constat

La gestion des chemins dans les applications est souvent peu sécurisée.

Positionnement général de ce type de problème

- Les applications doivent pouvoir se transporter facilement d'un environnement à un autre sans être gênées par l'organisation de leur "folder".
- La structure de leur "folder" doit pouvoir évoluer facilement en fonction des contraintes d'architecture rencontrées et des différences d'implémentation.
- Il doit être "facile" d'identifier programmatiquement dans quel "folder" on se trouve lors d'une opération de lecture/écriture.
- L'accès à un "folder" ne doit pas générer d'effet de bord non maîtrisé, le cas le pire étant que différents parcours du code ne débouchent sur des "paths" différents.
- Une définition de chemin inconsistante rend plus difficile l'intégration de plusieurs codes et/ou le refactoring.
- Certaines façon de coder et les caractéristiques des différents environnements R/Rstudio/git peuvent déboucher sur des définitions de "root" variables.
- la gestion des fichiers temporaires doit assurer que ceux-ci soient gérés au travers du système afin de ne pas créer de brèches spécifiques de sécurité qui ne soient pas adressables par des techniques gérées par les équipes dédiées à la sécurité.

Solutions envisageables (non exhaustives, non exclusives les unes les autres)

- Paramétrer la définition des chemins (solution générique)
- Créer des chemins relatifs,
- Centraliser la définition des chemins et interdire qu'une définition de chemin soit dupliquée dans le code pour éviter la création accidentelle d'incohérences lors des évolutions de l'application,
- S'affranchir des codages de chemins spécifiques à un OS,
- Créer des variables environnement pour les "path",
- Rassembler les définitions de chemin en un "lieu" et/ou une fonction unique
- Identifier automatiquement le "root" de l'application,

Solution proposée

1. Utilisation appropriée du package *here*.
2. Compléments fonctionnels au package *here* via le package *pathological*, dont la gestion des fichiers temporaires.

Codes, éléments pédagogiques , références

here

La fonction **here()** du package **“here”** s’appuie sur différents indices pour déterminer la racine du projet, comme la présence d’un fichier **.git**, d’un fichier projet ou d’un fichier **.here** (que l’on peut créer par une fonction du package). En changeant d’environnement (avec ou sans *git*, sans *Rstudio* …), il est donc possible de déterminer la racine (ou de la forcer en créant un .here) sans changer le comportement. Un code similaire à celui présenté ici est donc transposable sans risque dans divers contextes et permet de générer un *path* adapté pour vos fichiers et montre comment l’utiliser.

```
# not run
path_relatif_data <- "data"
path_data <- here::here(path_relatif_data)
if (!dir.exists(path_data)) {
    dir.create(path_data)
}

here::here(path_relatif_data, "mon_fichier_data.ext")
```

```
[1] "C:/Users/boss-hp/Desktop/rstudio-export 2020 10 08 05h47/data/mon_fichier_data.ext"
```

pathological

Le package permet d’utiliser la directory temporaire de la session pour y loger ses propres fichiers.

Il est aisé de connaître la directory temporaire de la session.

```
# accéder et lister le contenu de la directory temporaire par defaut
d <- pathological::temp_dir()
d
```

```
[1] "C:/Users/boss-hp/AppData/Local/Temp/RtmpQxAkon"
```

```
list.dirs(d)
```

```
[1] "C:/Users/boss-hp/AppData/Local/Temp/RtmpQxAkon"
```

Il est alors commode d’y inclure ses propres directories temporaires.

```
# créer plusieurs dir temporaires
mes_directories_tmp <- pathological::temp_dir(c("dir_tmp1", "dir_tmp2"))
pathological::create_dirs(mes_directories_tmp)
```

```
C:/Users/boss-hp/AppData/Local/Temp/RtmpQxAkon/dir_tmp1
                                                   TRUE
C:/Users/boss-hp/AppData/Local/Temp/RtmpQxAkon/dir_tmp2
                                                   TRUE
```

On peut les utiliser normalement, le package propose même une fonction de copie récursive de directories.

```
# copier une arborescence de fichiers, ici dans une de nos directories
# temporaires
pathological::copy_dir(here::here(), mes_directories_tmp[1], recursive = TRUE)
```

Que l'on peut lister pour vérifier

```
# lister cette directory temporaire avec une fonction r de base liste
# récursive par défaut (contient l'arborescence complète) ici pour écourter,
# on liste seulement la ligne 5 de la liste
list.dirs(mes_directories_tmp[1])[5]
```

Le package dispose de nombreuses fonctionnalités permettant de gérer et modifier les chemins de fichier et leurs extensions dans de bonnes conditions de compatibilité *Windows/Linux* et en respectant certains usages : par exemple le fait d'éviter de traiter les fichiers préfixés par un *"."* comme si ces fichiers comportaient une extension de fait. Pour autant, il supporte le parsing de noms de fichiers comportant un ou plusieurs points avant leur extension.

On voit ici que les chemins de fichier sont maîtrisables sans connaître l'OS.

```
# décomposer un chemin de FICHIER
chemin <- here::here("exemple_utilisation_pathological.R")
d <- pathological::decompose_path(chemin)
d$dirname
```

```
[1] "C:/Users/boss-hp/Desktop/rstudio-export 2020 10 08 05h47"
```

```
d$filename
```

```
[1] "exemple_utilisation_pathological"
```

```
d$extension
```

```
[1] "R"
```

Par ailleurs les usages sont respectés.

```
# Attention ne traite pas comme comportant une extension les fichiers dont
# le nom commence par un '.'
chemin <- here::here(".here")
d <- pathological::decompose_path(chemin)
d$dirname
```

```
[1] "C:/Users/boss-hp/Desktop/rstudio-export 2020 10 08 05h47"
```

```
d$filename
```

```
[1] NA
```

```
d$extension
```

```
[1] NA
```

A l'inverse, si l'on sait que l'on est sous Windows on peut accéder à des éléments caractéristiques de cet OS.

```
# uniquement pour Windows !!!  déterminer le drive où l'on est
d <- pathological::get_windows_drive()
d
```

```
C:/Users/boss-hp/Desktop/rstudio-export 2020 10 08 05h47
                                                    "C:"
```

Discussion

Dans l'exemple concernant *here* on constate que l'on a pas eu besoin d'exhiber le chemin correspondant à la racine du projet, ce qui évite les erreurs associées à la détermination de ce chemin de référence. On s'interdit le genre d'expressions suivantes :

```
chemin <- paste0(getwd(), "/")
```

Cette expression est en effet doublement dangereuse, le résultat du *getwd()* pouvant varier pendant la session, et le *"/"* pouvant être inefficace dans certains contextes non Linux. Evidemment une telle expression s'avère souvent moins dangereuse qu'un chemin "en dur", qui rend l'application peu portable d'un OS à l'autre, c'est pourquoi on la retrouve souvent. Mais on peut mieux faire avec *here*.