リリース 4: カテゴリ編集機能の実装—その 1

市東 亘

平成 30 年 1 月 9 日

1　概観

目 次

2　カテゴリ編集インターフェースの機能要件

必要な機能

- カテゴリの追加
- カテゴリの更新
- カテゴリの削除

外部仕様

(1) メインメニューから「カテゴリの編集」を選択すると，サブメニューとしてカテゴリ編集メニューが表示される.

(a) カテゴリの追加
カテゴリの一覧が表示され，新規追加カテゴリの入力プロンプトが表示される.

(b) カテゴリの更新
カテゴリ一覧から番号を指定して更新するカテゴリを選ぶと，更新用の入力プロンプトが表示される．

(c) カテゴリの削除
カテゴリ一覧から番号を指定して更新するカテゴリを選ぶと，確認用のメッセージが表示され，Yes を選択すると削除される．親カテゴリを削除する場合はその下に連なる子カテゴリも削除されるので確認を取るようにする．

3　メインメニューとカテゴリ編集メニューの実装

3.1　メインメニューの実装.

まず，メインメニューとして「新規カード」と「カテゴリの編集」を以下のように作成する．変更箇所は以下のとおり．

- 前回は doTask1 という名前だったが，機能が分かりやすいように，変数名も変更する．
- 各メニューのタイトルのスロット名を title から menu.title に変更．それに合わせて，run.app() 内部でメニュータイトルを抜き出すコードも x$menu.title に変更．
- 前回タスクを登録した箇所はメニューの登録に変わり，変数名も tasks から menus に変更．

```
# 新規カード（メインメニュー項目）
add.card.menu <- list(menu.title="新規カード",
                      fn=function() {
                          cat ("\n\n 処理: 新規カード作成中.....\n\n")
                      })
# カテゴリ編集（メインメニュー項目）
category.menus <- list(menu.title="カテゴリの編集",
                       fn=function() {
                           cat ("\n\n 処理: カテゴリの編集.....\n\n")
                       })

# アプリ起動関数
run.app <- function() {
    # メニューを登録
    menus <- list(add.card.menu, category.menus)
    # メインメニューの title だけ抜き出す.
    menu.items <- sapply(menus, function(x){ x$menu.title })

    while(TRUE) {
        choice <- menu(menu.items)
        if (choice == 0)
            return()
        menus[[choice]]$fn()
    }
}
```

3.2　カテゴリ編集サブメニューの追加

- 「カテゴリの編集」を選んだら，さらにサブメニューを表示したい．
  $\Longrightarrow$ アプリ起動関数内のメニュー表示と同じ!

```
# カテゴリ編集（サブメニュー項目）
add.category.menu <- list(menu.title="カテゴリの追加",
                          fn=function() {
                              cat ("\n\n 処理: カテゴリの追加中.....\n\n")
                          })
edit.category.menu <- list(menu.title="カテゴリの更新",
                           fn=function() {
                               cat ("\n\n 処理: カテゴリの更新中.....\n\n")
                           })
delete.category.menu <- list(menu.title="カテゴリの削除",
                           fn=function() {
                               cat ("\n\n 処理: カテゴリの削除中.....\n\n")
                           })
# カテゴリ編集（メインメニュー項目）
category.menus <- list(menu.title="カテゴリの編集",
                       fn=function() {
                           menus <- list(add.category.menu,
                                         edit.category.menu,
                                         delete.category.menu)
                           menu.items <- sapply(menus, function(x)x$menu.title)
                           while(TRUE) {
                               choice <- menu(menu.items)
                               if (choice == 0)
                                   return()
                               menus[[choice]]$fn()
                           }
                       })
```

`category.menus$fn` と `run.app()` 内のコードが重複している！
$\Longrightarrow$ Don't Repeat Youself! (DRY principle)

3.3 リファクタリングとは？

まずはコードを書き，そのあと，より良いコードに改善していく書き換え作業を「リファクタリング（refactoring)」と呼ぶ.
ここでは重複している部分を抽出する.

考察

- 共通部分は何か？
  $\Longrightarrow$ `run.app()` と `category.menus$fn()` 関数のロジック.
- 異なる部分は？
  $\Longrightarrow$ 登録するメニュー項目（`menus`）

```
# 異なる部分を引数にして外部から与える
show.menus <- function(menus) {
    menu.items <- sapply(menus, function(x)x$menu.title)
    while(TRUE) {
        choice <- menu(menu.items)
        if (choice == 0)
            return()
        menus[[choice]]$fn()
    }
}

# カテゴリ編集（メインメニュー項目）
category.menus <- list(menu.title="カテゴリの編集",
```

```
                        fn=show.menus) # 関数を代入

# アプリ起動関数
run.app <- show.menus # 関数を代入

run.app(list(add.card.menu, category.menus))
```

上のコード内で show.menus() 関数を，データのように扱っている．$\Longrightarrow$ 第 1 級関数オブジェクト！

考察. 上のコードはうまく動くか？(実行せずに問題点を見つけられるか？)

- show.menus() にはメニュー項目を引数として与えなければならない！
- いつ与える？ 実行時 or 代入時？
  どちらもうまくいかない！やってみよ．

4 クロージャ（関数閉包）

クロージャ (Closure) は，関数内で使う変数を，実行時に引数として与えるのではなく，関数定義時に関数に組み込むテクニック．

```
> # 「関数」を返す関数を定義
> closure <- function() {
+     outer.var <- "外側の関数のローカル変数"
+     # closure は以下の関数を返す
+     function () {
+         inner.var <- "インナー関数のローカル変数"
+         cat("inner.var =", inner.var, "\n")
+         cat("outer.var =", outer.var, "\n") # outer.var にアクセス
+     }
+ }
> test.function <- closure()  # (1) インナー関数を取り出し test.function に代入
> test.function()             # (2) インナー関数を起動

inner.var = インナー関数のローカル変数
outer.var = 外側の関数のローカル変数
```

- (1) で closure() を実行し終えると，closure() 関数のローカル変数である outer.var はもう役目を終えている．
- それにも関わらず (2) で closure() 関数のスコープ外からインナー関数を起動しても outer.var の値にアクセスできている！
- 関数を定義した時に，あたかも outer.var をインナー関数に閉じ込めてしまった様だ．
  $\Longrightarrow$ 関数閉包（クロージャ）
- 閉じ込めた変数のスコープを抜けても，変数は生き続ける？
  $\Longrightarrow$ オブジェクトは他から参照されている限り生き続ける．
  $\Longrightarrow$ 全てのオブジェクトは誰からも参照されなくなるとガーベッジコレクトされ，占有しているメモリが解放される．
- なぜ外側の変数にアクセスできる？
  $\Longrightarrow$ 関数の外側の environment は親 environment だからアクセスできる．

- closure() 関数は,「関数」を返す関数. $\Longrightarrow$ 高階関数！

outer.var を変更すれば，インナー関数に引数を渡さずに異なる値を渡すことができる！

```
> # 「関数」を返す関数に引数を用意
> closure <- function(x) {
+     outer.var <- x    # 引数を outer.var に設定
+     # closure は以下の関数を返す
+     function () {
+         inner.var <- "インナー関数のローカル変数"
+         cat("inner.var =", inner.var, "\n")
+         cat("outer.var =", outer.var, "\n") # outer.var にアクセス
+     }
+ }
> test1 <- closure("1 回目に作成")  # インナー関数を取り出し.
> test2 <- closure("2 回目に作成")  # インナー関数を取り出し.
> test1()

inner.var = インナー関数のローカル変数
outer.var = 1 回目に作成

> test2()

inner.var = インナー関数のローカル変数
outer.var = 2 回目に作成
```

インナー関数の振る舞いを変えることに成功！

外側の関数の「引数」すらも閉じ込められる！

```
> # 引数 x をクロージャに閉じ込める
> closure <- function(x) {
+     function () {
+         inner.var <- "インナー関数のローカル変数"
+         cat("inner.var =", inner.var, "\n")
+         cat("outer.var =", x, "\n")     # <-- 引数を直接参照！
+     }
+ }
> test1 <- closure("1 回目引数")  # インナー関数を取り出し.
> test2 <- closure("2 回目引数")  # インナー関数を取り出し.
> test1()

inner.var = インナー関数のローカル変数
outer.var = 1 回目引数

> test2()

inner.var = インナー関数のローカル変数
outer.var = 2 回目引数
```

5　リファクタリング：共通部分の抽出

高階関数（関数を返す関数）とクロージャを用いて，メニュー表示に関する共通部分を抽出するリファクタリングを行う．

```
## 共通部分を抽出
#  「メニューを表示する関数」を返す高階関数を定義
make.menu.func <- function(menus) {

    # 引数 menus をクロージャにして渡す
    function () {
        menu.items <- sapply(menus, function(x)x$menu.title)
        while(TRUE) {
            choice <- menu(menu.items)
            if (choice == 0)
                return()
            menus[[choice]]$fn()
        }
    }
}

# カテゴリ編集（メインメニュー項目）
category.menus <- list(menu.title="カテゴリの編集",
                       fn=make.menu.func(
                           list(add.category.menu,
                                edit.category.menu,
                                delete.category.menu)))

# アプリ起動関数
run.app <- make.menu.func(list(add.card.menu, category.menus))
```

重複がなくなりすっきりした！
$\Longrightarrow$ 重複箇所を 1 箇所にまとめれば，将来の変更や修正も箇所で済む．

6　宿題

次回以降で必要になる `make.category.list()` 関数を `flashcard.R` に追加し，コードを読み解いてくる．ヒント: 内部の `get.children()` 関数は再帰処理をしています．

```
# カテゴリ一覧表示用リスト生成関数
#    cat.db を親子関係順にソート & 字下げ調整済みカテゴリ名をベクトル化
#    結果はリスト（data と names）にして返す．
make.category.list <- function (cat.db) {
    get.children <- function(id, depth) {
        data <- cat.db[cat.db$ID==id, ]
        # カテゴリ名の前に字下げ用の空白を depth 分だけ挿入
        names <- paste(paste(rep("   ", depth), collapse="", sep=""),
                       data$カテゴリ名, sep="")
        for (i in which(cat.db$親カテゴリ==id)) {
            children <- get.children(cat.db[i, "ID"], depth+1)
            data <- rbind(data, children$data)
            names <- c(names, children$names)
        }
        list(data=data, names=names)
    }
    df <- NULL  # 親子関係ソート済み category のデータフレーム格納用
    nm <- NULL  # 字下げしたカテゴリ名格納用ベクトル
    for (i in which(cat.db$親カテゴリ == 0)) {
        res <- get.children(cat.db[i, "ID"], 0)
        df <- rbind(df, res$data)
```

```
        nm <- c(nm, res$names)
    }
    list(data=df, names=nm)
}
```

タイプミスがなければ，以下の実行結果を得られる．ただし，create.category.db() と add.category() の定義は事前に読み込んでおくこと．下のコードは動作確認用なのでファイルに保存したければ，flashcard-test.R の方に記述すること．

```
# 動作確認用カテゴリデータベースを作成
get.id <- function (cat.db, name) {
    cat.db[cat.db$カテゴリ名 == name, "ID"]
}
cat.db <- create.category.db()
cat.db <- add.category(cat.db, "英単語", 0)$db
cat.db <- add.category(cat.db, "授業", 0)$db
cat.db <- add.category(cat.db, "期末テスト", get.id(cat.db, "英単語"))$db
cat.db <- add.category(cat.db, "英会話", get.id(cat.db, "授業"))$db
cat.db <- add.category(cat.db, "基礎", get.id(cat.db, "英会話"))$db
cat.db <- add.category(cat.db, "ゼミ", get.id(cat.db, "授業"))$db
cat.db <- add.category(cat.db, "2 年", get.id(cat.db, "ゼミ"))$db
cat.db <- add.category(cat.db, "3 年", get.id(cat.db, "ゼミ"))$db
cat.db <- add.category(cat.db, "4 年", get.id(cat.db, "ゼミ"))$db
cat.db <- add.category(cat.db, "外書講読", get.id(cat.db, "2 年"))$db
cat.db <- add.category(cat.db, "専門書", get.id(cat.db, "4 年"))$db
cat.db <- add.category(cat.db, "卒論", get.id(cat.db, "4 年"))$db

# 動作確認
make.category.list(cat.db)

## $data
##             ID カテゴリ名 親カテゴリ
## 1  1515501725     英単語          0
## 3  1515501726 期末テスト 1515501725
## 2  1515501726       授業          0
## 4  1515501726     英会話 1515501726
## 5  1515501725       基礎 1515501726
## 6  1515501725       ゼミ 1515501726
## 7  1515501726        2 年 1515501725
## 10 1515501725   外書講読 1515501726
## 8  1515501726        3 年 1515501725
## 9  1515501725        4 年 1515501725
## 11 1515501726     専門書 1515501725
## 12 1515501725       卒論 1515501725
##
## $names
##  [1] "英単語"             "   期末テスト"      "授業"
##  [4] "   英会話"          "      基礎"         "   ゼミ"
##  [7] "      2 年"          "         外書講読" "      3 年"
## [10] "      4 年"          "         専門書"   "         卒論"

cat(make.category.list(cat.db)$names, sep="\n")

## 英単語
##    期末テスト
## 授業
##    英会話
##       基礎
##    ゼミ
##       2 年
##          外書講読
##       3 年
##       4 年
##          専門書
##          卒論
```