L06 – Condition Handling

17. Mai 2021

Contents

Conditions (auch *exceptions* oder *Ausnahmen*) sind Botschaften an Nutzer einer Funktion, die über bestimmte Programmzustände informieren.

Bemerkung: Das Wort *Nutzer* schließt hier Personen jeglichen Geschlechts sowie andere Programme und Funktionen mit ein.

Zu den Conditions zählen Fehler (*Errors*), Warnungen (*Warnings*) und Nachrichten (*Messages*).

```
message("Hi!") # Message
## Hi!
log(-1) # Warning
## Warning in log(-1): NaNs produced
## [1] NaN
1 + "a" # Error
## Error in 1 + "a": non-numeric argument to binary operator
```

Condition handling ermöglicht es, auf Conditions im Programmcode zu reagieren.

Signalling

Errors

Fehlermeldungen werden mit der Funktion `stop()` gesendet. Sie führen zum Abbruch des Programmes.

```
g <- function() {
  cat("g start\n")
  h()
  cat("g end\n")
}
h <- function() {
  cat("h start\n")
  stop("This is an error!")
  cat("h end\n")
}
```

```
g()
## g start
## h start
## Error in h(): This is an error!
```

Warnings

Warnungen werden durch `warning()` gesendet. Die Funktionsausführung wird dabei nicht abgebrochen. Eine Funktion kann mehrere Warnungen senden.

```
f <- function() {
  cat("cat 1\n")
  warning("warning 1")
  cat("cat 2\n")
  warning("warning 2")
}
f()
## cat 1
## cat 2
## Warning messages:
## 1: In f() : warning 1
## 2: In f() : warning 2
```

Erzeugt eine Funktion sehr viele Warnungen, werden nicht alle ausgegeben.

```
for (i in 1:100) warning(paste("Pass auf!", i))
## There were 50 or more warnings (use warnings() to see the first 50)
```

Um die letzten Warnungen abzurufen, nutze `warnings()`. Dies gibt die Namen der benannten Liste `last.warning` (ein Objekt im Paket `base`) aus.

```
warnings()
## Warning messages:
## 1: Pass auf! 1
## 2: Pass auf! 2
## 3: Pass auf! 3
### ...
## 49: Pass auf!  49
## 50: Pass auf!  50
```

Messages

Einfache Nachrichten an den Nutzer einer Funktion werden mit `message()` gesandt.

Sie werden sofort ausgegeben.

```
fm <- function() {
  cat("1\n")
  message("M1")
  cat("2\n")
  message("M2")
}
fm()
## 1
## M1
## 2
```

```
## M2
```

Es gibt zwei wichtige Unterschiede zwischen `cat()` und `message()`:

1) `message()` schreibt Nachrichten nach `stderr` wie auch `warning()` und `stop()`. `cat()` schreibt nach `stdout`.

`stderr`, `stdout` sind zwei unterschiedliche Kanäle, mit denen ein Programm mit dem Nutzer kommunizieren kann.

Bei der Nutzung von R in RStudio wird `stderr` und `stdout` auf der Konsole ausgegeben. Dabei wird `stderr` farblich hervorgehoben.

2) Die Botschaften einer `message()` werden über das Condition-System versendet. Dies ermöglicht es uns die Nachrichten abzufangen und im Programmcode zu verarbeiten.

Insbesondere können Nachrichten einfach unterdrückt werden. Das ist für `cat()`-Ausgaben schwieriger.

Die Behandlung von Conditions wird in den nächsten Abschnitten beschrieben.

Aufgabe:

Schreibe eine Funktion `clear_warnings()`, sodass `warnings()` direkt nach dem Aufruf von `clear_warnings()` keine Ausgabe macht.

```
warning("warn")
## Warning message:
## warn
warnings()
## Warning message:
## warn
clear_warnings()
warnings()
##
```

Ignoring

Verhindere einen Funktionsabbruch bei Fehler in einem Ausdruck `expr` mit `try(expr)`. `try()` wandelt einen *Fehler* in eine *Nachricht* um.

```
f <- function() {
  print("1")
  stop("1.5")
  print("2")
}
g1 <- function() {
  f()
  print("Ende")
}
g1() # Abbruch wegen Fehler
## [1] "1"
## Error in f() : 1.5
g2 <- function() {
  try(f())
  print("Ende")
}
g2() # Weiter ausführen trotz Fehler
## [1] "1"
```

```
## Error in f() : 1.5
## [1] "Ende"
```

Verhindere zusätzlich die Ausgabe der Fehlermeldung mit `try(expr, silent=TRUE)`.

```
g3 <- function() {
  try(f(), silent=TRUE)
  print("Ende")
}
g3()
## [1] "1"
## [1] "Ende"
```

Verhindere die Ausgabe von Warnungen mittels `suppressWarnings(expr)`.

```
f <- function() {
  message("1: message")
  print("2: print")
  warning("3: warning")
}
f()
## 1: message
## [1] "2: print"
## Warning in f(): 3: warning
suppressWarnings(f())
## 1: message
## [1] "2: print"
```

Verhindere die Ausgabe von Nachrichten mittels `suppressMessages(expr)`.

```
suppressMessages(f())
## [1] "2: print"
## Warning in f(): 3: warning
```

Aufgabe:

Schreibe eine Funktion `careless(f)`, der eine Funktion `f` übergeben wird. Die Rückgabe ist ein Funktionsobjekt, welches die gleiche Funktion wie `f` ausführt, allerdings keine Warnmeldungen mehr von sich gibt.

```
log(-1)
## [1] NaN
## Warning message:
## In log(-1) : NaNs produced
careless_log <- careless(log)
careless_log(-1)
## [1] NaN
```

Handling

Jede Condition hat eine Standardbehandlung: Fehler halten die Funktionsausführung an, Warnungen werden gespeichert und als Liste nach dem Funktionsaufruf ausgegeben und Nachrichten werden sofort ausgegeben.

Dieses Verhalten kann mit sogenannten *handlers* vorübergehend überschrieben werden.

Handlers sind Funktionen, die als Argument ein *Condition-Objekt* entgegennehmen.

Condition-Objekte sind S3-Objekte, die von der Klasse `condition` abstammen und auf Listen aufbauen.

Ein Condition-Objekt `cond` enthält den Text der Condition in `cond$message` und den Funktionsaufruf, der die Condition ausgelöst hat, unter `cond$call`.

Bei der Registrierung von Handlern unterscheiden wir *exiting Handlers* (registrieren via `tryCatch()`) und *calling Handlers* (registrieren via `withCallingHandlers()`).

Exiting Handlers

`tryCatch()` registriert **exiting Handlers**.

Für jede Art von Condition kann ein eigener Handler registriert werden. Sie werden `trycatch()` als Attribute mit den Namen `message`, `warning`, bzw `error` übergeben und überschreiben die Standardbehandlung.

Der Code, für den die übergebenen Handlers gelten sollen, wird als Argument `expr` angegeben (oder als unbenanntes Argument).

```
tryCatch(
  message = my_message_handler,
  warning = my_warning_handler,
  error = my_erro_handler,
  expr={
    ...
  }
)
```

Exiting Handlers werden aufgerufen, sobald eine Condition signalisiert wird. Dabei wird die Funktion, in der die Condition entsteht, verlassen (*exit*) und es wird nicht mehr dorthin zurückgekehrt.

```
f <- function() {
  print(1)
  message(2)
  print(3)
  stop(4)
}
g <- function() {
  tryCatch(
    message =function(cond) print("Eine Nachricht!"),
    error = function(cond) print("Ein Fehler!"),
    f()
  )
  print(5)
}
g()
## [1] 1
## [1] "Eine Nachricht!"
## [1] 5
```

Aufgabe:

A

Schreibe eine Funktion `fail_with(expr, value)`, die `expr` auswertet (`function(expr) expr` wäre eine Funktion, die `expr` auswertet). Falls `expr` einen Fehler erzeugt, soll dieser jedoch abgefangen und `value` zurückgegeben werden, anstatt abzubrechen.

```
log(10)
## [1] 2.302585
log("x")
## Error in log("x"): non-numeric argument to mathematical function
```

```r
fail_with(log(10), NA_real_)
## [1] 2.302585
fail_with(log("x"), NA_real_)
## [1] NA
```

B

Schreibe eine Funktion `does_error(expr)`, die zurückgibt, ob `expr` einen Fehler erzeugt.

```r
does_error(log(10))
## [1] FALSE
does_error({
  x <- "asdf"
  log(x)
})
## [1] TRUE
```

Hinweis: Der obige Umgang mit einem Ausdruck `expr` als Argument ist wegen *Lazy Evaluation* möglich: Das Argument `expr` wird erst dann ausgewertet (und kann erst dann einen Fehler erzeugen), wenn wir es im Inneren der Funktion benutzen.

Calling Handlers

`withCallingHandlers()` registriert **calling Handlers**.

Die Syntax ist analog zu `tryCatch()`.

Nach der Ausführung eines Calling-Handlers wird mit der ursprünglichen Funktionsausführung fortgefahren.

```r
f <- function() {
  print(1)
  message(2)
  print(3)
  stop(4)
}
g <- function() {
  withCallingHandlers(
    message = function(cond) print("Eine Nachricht!"),
    error = function(cond) print("Ein Fehler!"),
    f()
  )
  print(5)
}
g()
## [1] 1
## [1] "Eine Nachricht!"
## 2
## [1] 3
## [1] "Ein Fehler!"
## Error in f(): 4
```

Beachte, dass hierbei nicht nur unsere selbst definierten Handlers ausgeführt wurden, sondern auch die Standard-Handlers.

`withCallingHandlers()` gibt die Condition also nach der Behandlung weiter im Gegensatz zu `tryCatch()`.

```r
cat_m <- function(cond) cat("m\n")
withCallingHandlers(
```

```
  message = cat_m,
  withCallingHandlers(message = cat_m, message("bla"))
)
## m
## m
## bla
tryCatch(
  message = cat_m,
  tryCatch(message = cat_m, message("blub"))
)
## m
```

Die Weitergabe kann mit `rlang::cnd_muffle()` verhindert werden.

```
cat_m <- function(cond) {
  cat("m\n")
  rlang::cnd_muffle(cond)
}
withCallingHandlers(
  message = cat_m,
  withCallingHandlers(message = cat_m, message("blub"))
)
## m
```

Aufgabe:

A

Schriebe eine Funktion `count_warnings(expr)`, die `expr` auswertet und eine dabei die Anzahl der Warnungen zählt.

```
count_warnings({
  for (i in 1:100) warning(paste("Pass auf!", i))
  42
})
## There were  100  warnings!
## [1] 42
## There were 50 or more warnings (use warnings() to see the first 50)
```

B

Schreibe eine Funktion `record(expr)`, die die Texte aller Warnungen und Nachrichten, die in `expr` entstehen, in einem `character`-Vektor zurückgibt.

```
record({
  message("a")
  warning("b")
  cat("c\n")
  message("d")
})
## c
## [1] "a\n" "b"   "d\n"
```