예외처리 및 Debug

KRUG 4차 Meet up

(2012-05-17)

유충현
bdboy@r-project.kr

Agenda

- **Start-Up & Shutdown**
  - Start-Up
  - Shutdown
- **Messages**
  - Warning
  - Error
- **Exception**
  - Error Recovery
  - Function Exit Code
- **Debugging**
  - Trace Back
  - Debug
  - Browser Mode
  - Browse
  - Recover
- **Q&A**

Start-Up & Shutdown

Start-Up

Start-Up & Shutdown

- **.First function**
  - Hidden object
    - .으로 시작하는 R 객체는 hidden 객체임
    - like UNIX
    - ls() : 조회 불가능
    - ls(all.names=TRUE) : 조회
  - R이 구동되면서 자동적으로 실행되는 함수
    - 사용자 정의 함수임
    - R을 매번 구동할 때마다 반복적으로 수행하는 코드를 기술하면 좋음
    - like autoexec.bat : DOS (Windows)
    - like .profile : (UNIX)
- **Example**

```
.First <- function()
{
  print(paste("Hello R World!!!", date()))
  library(lattice)
}
```

Shutdown

- .Last function
  - R을 종료하면서 자동적으로 실행되는 함수
    - 사용자 정의 함수임
    - R을 수행 중에 설정한 설정 값을 원래의 값으로 초기화 할 때 유용하게 사용함
- Example

```
pi <- 3
.Last <- function()
{
  rm(pi, pos=1) # Not rm(pi)
}
```

Messages

Warning Messages

- **경고 메시지, 그러나 이후 Script 수행 진행**
  - 치명적인 오류는 아니지만 검토 해볼만한 사항을 알려줌
    - 메시지 길이 제한

```
> options()[grep("warning", names(options()))]
$warning.length
[1] 1000
> getOption("warning.length")
[1] 1000
```

    - 메시지 출력 여부 설정

```
> getOption("warn") # Max 50
[1] 0
> options(warn=-1)  # 출력 해제
```

- **warning()로 생성 : call. = TRUE**
  - Example

```
testit <- function() warning("testit")
testit()
```

- **warnings()로 조회**
  - Example

```
warnings()
last.warning
```

Error Messages

- **에러 메시지, Script 실행 종료**
  - 치명적인 오류로 실행 정지 및 원인을 알려줌
- **stop()으로 생성 : call. = T**
  - Example

    iter <- 12

    if(iter > 10)

      stop("too many iterations")
- **stopifnot()으로 조건부 생성**
  - Example

    stopifnot (iter <= 10)
- **geterrmessage()로 조회**
  - Example

    geterrmessage()

Exception

Error Recovery Exception

- **Try ~ Catch ~ Exception**
  - 예측 가능한 오류의 경우 오류를 인지하여 후 처리함
- **tryCatch() 이용**
  - Example

```
e <- simpleError("test error")
tryCatch(stop(e), error = function(e) e, finally=print("Hello"))
```

- **에러의 발생 가능성이 있는 루틴에 적용**
  - Example

```
getLog <- function(x) {
 tryCatch(log(x), error = function(e) e, finally={if (x<0)
print("must x>0")})
}
getLog(-1)
```

Function Exit Code Exception

- **함수 종료 시 반드시 실행할 코드가 있는 경우 사용**
  - 함수 종료 시 반드시 실행할 코드가 실행을 할 수 없는 경우 발생
    - 함수 안에서 코드 이전에 에러가 발생할 경우
    - 함수 안에서 에러가 발생할 경우에도 반드시 실행해야 할 경우
  - 함수 안에서 Global 환경변수를 수정할 경우에 유용하게 사용
- **on.exit() 이용**
  - Example

```
opar <- par(mar = c(5,5,5,5))
  on.exit(par(opar))
```

Debugging

Trace Back

- **에러가 발생한 시점부터 역순으로 추적함**
  - 여러 번의 중첩으로 함수 호출로 이루어진 함수에서 에러가 발생하였을 때, 어느 부분에서 에러가 발생하였는지 인지
    - 에러를 수정하기 위해서는 먼저 에러 발생 부분을 찾아야 한다.
  - 에러 발생 지점에서 호출 지점으로의 추적
  - strange error의 추적
  - 간단한 에러는 쉽게 찾을 수 있는 방법이나, 복잡한 에러는 찾기가 쉽지 않음
- **traceback() 이용**
  - 이전에 발생한 에러를 추적함
  - Example

```
foo <- function(x) { print(1); bar(2) }
bar <- function(x) { x + a.variable.which.does.not.exist }
foo(2)
```

Debug Debugging

- **R에서 제공하는 Debug 도구를 이용한 오류 찾기**
  - 구문 에러는 쉽게 찾을 수 있으나 논리적인 오류는 쉽게 찾지 못함
    - 에러가 양성 종양이라면 오류는 악성 종양
    - Debug는 악성 종양을 찾아내서 수술하는 일련의 방법이다.
- **debug() 이용**
  - debug 함수에 debug하려는 함수를 인자로 사용하면 그 함수가 호출될 때마다 R의 Debug Mode로 이동한다.
  - 매번 호출될 때마다 Debug Mode로 이동한다.
  - 해제를 하기 위해서는
    - undebug() 로 toggle off
    - debugonce()를 이용하는 방법도 있음
  - Example

```
debug(my_func)
undebug(my_func)
debugonce(my_func)
#debugonce(package:::hid_func)
```

Browser Mode Debugging

- **Browser Mode Command**
  - n
    - 현 단계를 수행 후 다음 단계로 이동 (엔터 키도 동일)
    - **객체 n은 get("n")으로**
  - c 혹은 cont
    - 현 단계의 마지막으로 이동
    - 루프 내에서는 루프의 끝으로, 루프가 아닌 곳에서는 함수의 끝으로 이동하여 종료됨
  - where
    - 호출로 활성화된 함수의 traceback 목록
  - Q
    - Exit Browser Mode
- **isdebugged() – 현재 Debug Mode에 있는가?**
  - 가끔 Command Line인지 Browser Mode 인지 헷갈릴 때 이용
- **Example**

```
debugonce(mean.default)
mean(1:10)
```

Browse

- **Browse**
  - 명시적으로 Browser Mode를 호출
  - browser()
  - Example

```
my.sd.1 <- function(x)                    # (1) 사용자 정의 함수
{
    mu <- mean(x)
    sumsq <- sum((x - mu)^2)
    browser()                        # (2) browser 함수
    n <- length(x)
    sqrt(sumsq / (n-1))
}
> my.sd.1(1:10)
Called from: my.sd.1(1:10)
Browse[1]>
```

Trace

Debugging

- **Trace**
  - 대상 함수에 디버깅 코드를 삽입하여 거의 실시간으로 디버그
  - trace() : 설정
  - untrace() : 해제
  - Example
    - 삽입 코드

```
if(any(is.nan(tmp))) {
  browser()
}
```

    - Usage

```
as.list(body(my.power))
trace("my.power", quote(if(any(is.nan(tmp))) { browser() }), at=3,
print=F)
my.power
body(my.power)
my.power(2, 3)
untrace(my.power)
```

Recover

Debugging

- Recover
  - 에러가 발생할 경우에만 Browser Mode 호출
  - recover() : 설정, Browse Mode 탈출 시 0을 입력
  - untrace() : 해제
  - Example
    - Usage

```
my.power <- function(x, y)
{
    exp(y * func.1(x))
}

func.1 <- function(x)
{
    tmp <- log(x)

    if (tmp > 709) Inf else tmp
}
trace("my.power", quote(if(any(is.nan(tmp))) { recover() }), at=3, print=F)
my.power(3, 2)
my.power(-2, 2)
untrace("my.power")
```

- options(error=recover) : 모든 발생할 경우에만 Browser Mode 호출
- options(error=NULL) : 해제

Appendix

Etc

Appendix

- **암묵적인 print**
  - command line에서 object name을 입력하면 암묵적으로 print 함수가 수행됨
    - methods(print) 로 객체의 종류 확인
      - *로 끝나는 함수는 Hidden 함수
  - ls를 입력하면 ls 함수의 내용이 출력되고, 1+4를 입력하면 5가 출력되는 현상들...
- **Package 안에서의 Hidden Objects**
  - ::: 연산자로 참조 가능
  - Example

    print.aov # not found

    stats:::print.aov # found
- **연산자도 함수다**
  - R에서는 연산자도 함수다
  - “+”(1+5)

Q & A