Formats

Brandon Moretz

2021-12-27

Format Objects

```
library(dyn.log)
```

Overview

Format objects are the driver of customization in log rendering. Log layouts were developed with the composition design pattern in mind; a log layout is simply a series of formats that get evaluated with their associated context to form a log message.

All formats derive from the **fmt_layout** base type and have a couple of generics associated with them, specifically: **style** and **value**. The **fmt_layout** is meant to be an abstract base type; by driving from it the logging framework can make some assumptions about how to treat a format object.

Format Types

There are five main categories of log format objects:

- Core
  - **fmt_level_info:** the log level information.
  - **fmt_log_msg:** the log message, evaluated with standard glue format.
- System Context
  - **fmt_metric:** a 'system' context value; see below for more detail.
  - **fmt_timestamp:** the current system time with a customizable format.
- Execution Scope
  - **fmt_exec_scope:** an 'execution' context value; see below for more detail.
- Class Fields
  - **fmt_cls_field:** a field value in the encompassing **R6** class; see below for more detail.
- Literals & New Lines
  - **fmt_literal:** a literal value, which is useful for tweaking exact format specifications.
  - **fmt_newline:** a new line feed in the log message, which is useful for multi-line log messages that have a lot of contextual information in the log output.

System Context

The values available for a **fmt_metric** type can be accessed via *sys_context*:

```
sys_context()
$sysname
[1] "Linux"

$release
[1] "5.10.16.3-microsoft-standard-WSL2"

$version
[1] "#1 SMP Fri Apr 2 22:23:49 UTC 2021"

$nodename
[1] "WORKSTATION"

$machine
[1] "x86_64"

$login
[1] "unknown"

$user
[1] "bmoretz"

$effective_user
[1] "bmoretz"

$r_ver
[1] "4.1.2"

attr(,"class")
[1] "sys_context" "context"
```

System Context Example

```
new_log_layout(
  format = list(
    new_fmt_log_level(),
    new_fmt_metric(crayon::green$bold, "sysname"),
    new_fmt_metric(crayon::yellow$bold, "release"),
    new_fmt_timestamp(crayon::silver$italic, "[%x %H:%M:%S]"),
    new_fmt_log_msg()
  ),
  seperator = '-',
  association = "ex-sys-layout"
)

var1 <- "abc"; var2 <- 123; var3 <- round(runif(1), digits = 6)
```

```
Logger$debug("my log message - var1: {var1}, var2: {var2}, var3: {var3}",
             layout = "ex-sys-layout")
```

```
DEBUG-Linux-5.10.16.3-microsoft-standard-WSL2-[12/27/21 01:06:25]-my log message - var
```

As you can see, the log message has a great deal of detail, but difficult to interpret due to the amount of information jammed into one line. This is where literals and new lines come into play.

Literals & New Lines

Literals and new lines are simple formatting objects that help you tweak the layout of a log message to something that is both informative and easy to consume. Taking the previous example, and tweaking the format slightly incorporating literals & new lines, we can produce a log message like this:

```
new_log_layout(
  format = list(
    new_fmt_metric(crayon::green$bold, "sysname"),
    new_fmt_literal(crayon::magenta, "["),
    new_fmt_metric(crayon::blue$bold, "release"),
    new_fmt_literal(crayon::magenta, "]"),
    new_fmt_line_break(),
    new_fmt_log_level(),
    new_fmt_timestamp(crayon::silver$italic, "[%x %H:%M:%S]"),
    new_fmt_log_msg()
  ),
  seperator = ' ',
  association = "ex-syslit-layout"
)

var1 <- "abc"; var2 <- 123; var3 <- round(runif(1), digits = 6)

Logger$debug("my log message - var1: {var1}, var2: {var2}, var3: {var3}",
             layout = "ex-syslit-layout")
```

```
Linux [ 5.10.16.3-microsoft-standard-WSL2 ]
DEBUG [12/27/21 01:06:25] my log message - var1: abc, var2: 123, var3: 0.957485
```

Which has the same information as the previous example, but much easier to consume.

Execution Scope

Execution scope formats give you the ability to log the context around the invocation of the logger, and is a context object, much like *sys_context*, called **exec_context**:

```
test <- function(a, b, c) {
  wrapper <- function(x, y, z) {
    outer <- function(d, e, f) {
      inner <- function(g, h, i) {
        # call_subset is used here to skip past knitr execution calls
        exec_context(max_calls = 30, call_subset = c(knitter_offset, -1))
      }

      inner(d, e, f)
    }

    outer(x, y, z)
  }
  wrapper(a, b, c)
}

exec_context <- test(1,2,3)
```

```
exec_context
$call_stack
        call_1          call_2          call_3          call_4
"global::test"      "wrapper"        "outer"        "inner"
attr(,"class")
[1] "call_stack" "stack"

$calling_fn
[1] "inner"

$ncalls
[1] 4

attr(,"class")
[1] "exec_context" "context"
```

The evaluated exec_context gives you a structure with these 3 fields:

- **call_stack:** a named vector of calls
  - *call_1* : “global::test” - top level call
  - *call_2* : “wrapper” - …
  - *call_3* : “outer” - …
  - *call_4* : “inner” - inner most fn call
- **calling_fn:** name of the function enclosing the logger call.
  - *calling_fn:* inner
- **ncalls:** number of calls in the stack.
  - *ncalls:* 4

The execution scope can be accessed via the **new_fmt_exec_scope** format object, e.g.:

```
new_log_layout(
  format = list(
    new_fmt_metric(crayon::green$bold, 'sysname'),
    new_fmt_metric(crayon::blue$yellow, 'release'),
    new_fmt_line_break(),
    new_fmt_log_level(),
    new_fmt_timestamp(crayon::silver$italic, '[%x %H:%M:%S]'),
    new_fmt_literal(crayon::magenta$bold, 'fn('),
    new_fmt_exec_scope(crayon::magenta$bold, 'calling_fn'),
    new_fmt_literal(crayon::magenta$bold, ')'),
    new_fmt_log_msg(),
    new_fmt_line_break(),
    new_fmt_exec_scope(crayon::bgYellow$blue$bold, 'call_stack')
  ),
  seperator = '-',
  association = 'ex-sysexec-cs-layout'
)

local_fn <- function() {
  outer <- function() {
    inner <- function() {
      var1 <- "abc"; var2 <- 123; var3 <- round(runif(1), digits = 6)

      Logger$debug("my log message - var1: '{var1}', var2: '{var2}', var3:
        '{var3}'",
                   layout = 'ex-sysexec-cs-layout')
    }
    inner()
  }
  outer()
}

local_fn()
```

```
Linux-5.10.16.3-microsoft-standard-WSL2
DEBUG-[12/27/21 01:06:25]-fn(-inner-)-my log message - var1: 'abc', var2: '123', var3:
global::local_fn-outer-inner
```