Erlang 快速入门

原文：*Getting Started with Erlang*
翻译: @dontpanic
版本: 20220406 (f9f6630)
在线阅读：*https://tuesday.dontpanic.blog*

1 简介

本教程希望能够帮助您快速入门 Erlang。教程里的内容都是事实，然而只是部分事实——例如，我们只为您展现了一些最简单的语法形式，并没有涉及那些深奥晦涩的语法。对于我们省略的那些细节，在正文中会使用 “* manual *” 记号标记，您可以在 Erlang Book 或者 Erlang 参考手册中找到更详细的描述。

1.1 前置需求

本书读者需要具备以下知识:

- 计算机的基础知识
- 计算机编程的基础知识

1.2 超过本书讨论范围的话题

- 引用
- 局部错误处理（catche/throw)
- 单向连接（monitor）
- 二进制数据处理（二进制/比特位语法）
- 列表生成式（List comprehension）
- 与其他语言编写的软件进行通信（port），参见 Interoperability Tutorial
- Erlang 库（例如文件处理）
- OTP 以及 Mnesia 数据库
- 用于 Erlang 词项的哈希表
- 在正在运行的系统中修改代码

2 顺序编程

2.1 The Erlang Shell

大多数操作系统都内置了命令解释器，也叫做 Shell。UNIX 和 Linux 上有很多种 Shell，Windows 则内置了命令提示符。Erlang 也有它自己的 Shell，可以在里面直接编写 Erlang 代码并求值（参见参考手册中 shell(3) 里的 STDLIB 部分）。

在 UNIX 或者 Linux 系统中，可以在系统 Shell 中输入 `erl` 来打开一个 Erlang Shell。你会看到类似这样的输出:

```
% erl
Erlang R15B (erts-5.9.1) [source] [smp:8:8] [rq:8] [async-threads:0] [hipe] [kernel-
poll:false]

Eshell V5.9.1  (abort with ^G)
1>
```

试一下在 Shell 中输入 `2 + 5.` 并按一下回车。需要注意的是，每次输入完代码之后，需要一个句号 `.` 来需要告诉 Erlang Shell 语句已经结束。

```
1> 2 + 5.
7
2>
```

从这个例子可以看出，Erlang Shell 会在可以输入的行前面添加编号（比如 1> 2>），它也正确地计算出 2 + 5 等于 7。如果在 Shell 里打错字了，可以用退格键删除。Shell 还支持更多的编辑命令，可以参见 ERTS 用户手册中的 tty - A command line interface 一节。

（请注意，在接下来的教程中，会有很多编号是乱序的。这是由于本教程经过了多次的攥写与编排。）

下面让我们来尝试一下更复杂的计算：

```
2> (42 + 77) * 66 / 3.
2618.0
```

请注意括号的使用、乘法操作符 `*` 以及除法操作符`/`，与我们平时做的算数运算一致（详情请参阅 Expressions）

按下 Control-C 可以种植 Erlang 系统和 Erlang Shell，会出现以下提示：

```
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
a
%
```

输入 `a` 离开 Erlang 系统。

另一种关闭 Erlang 系统的方法是使用 `halt()`：

```
3> halt().
%
```

2.2 模块与函数

如果一门编程语言只支持在 shell 中运行代码的话，那它的实用性就大打折扣了。下面是一个简短的 Erlang 程序。请使用任意一种文本编辑器把这段代码保存为 `tut.erl` 文件。请确保文件名设定为 `tut.erl`，并把这个文件放在你打开 `erl` 时所在的目录。如果你的编辑器很智能的话，它可能会支持格式化 Erlang 代码，这样代码看起来会更加整洁（如果你在使用 Emacs 编辑器，可以参看用户手册中的 The Erlang mode for Emacs）。不过即便你的编辑器不提供 Erlang 语言的支持也没关系。这是你需要保存的代码：

```
-module(tut).
-export([double/1]).

double(X) ->
    2 * X.
```

不难猜到，这段程序是用来把一个数字翻倍的。代码前两行的含义我们会稍后加以说明。下面我们来编译这段程序。在 Erlang shell 中，可以使用下面的代码进行编译，其中 `c` 就代表了

编译:

```
3> c(tut).
{ok,tut}
```

{ok,tut} 表示编译成功。如果它显示了 error，那就说明你保存的代码有问题。同时它还会输出一些额外的错误信息，你可以根据这些错误信息对代码进行修改，然后尝试重新编译。

下面来运行一下:

```
4> tut:double(10).
20
```

准确无误，10 翻倍后是 20。

现在我们来解释一下代码的头两行。Erlang 程序以文件的形式存在，每个文件包含了一个 Erlang **模块**。模块的第一行是模块的名字（参见 Modules）

```
-module(tut).
```

因此，这个模块叫做 tut。请特别注意在行尾有一个句号 .。文件名必须与模块名相同，同时需要以 erl 作为文件后缀。当需要调用其他模块中的函数时，需要使用模块名:函数名 (参数) 这样的格式。因此下面的代码表示调用 tut 模块中的 double 函数，并传入参数 10。

```
4> tut:double(10).
```

第二行表示 tut 模块包含了 double 函数，它接受一个参数（在这个例子中参数名为 X ）:

```
-export([double/1]).
```

同时这一行也表示这个函数允许从 tut 模块外部调用。下文会进行详细的讨论。请再次主义行尾的 .。

现在我们来看一个更复杂的例子：求一个数的阶乘。例如，4 的阶乘是 432*1 = 24。

在一个新文件中输入下面的代码，并保存为 tut1.erl:

```
-module(tut1).
-export([fac/1]).

fac(1) ->
    1;
fac(N) ->
    N * fac(N - 1).
```

回顾一下刚才的知识：这是一个叫做 tut1 的模块，它包含 fac 函数，这个函数接受一个参数，用 N 表示。

代码的前半段表示 1 的阶乘是 1:

```
fac(1) ->
    1;
```

未完待续