维基百科

Erlang

维基百科，自由的百科全书

Erlang（/ˈɜːrlæŋ/）是一种通用的并发函数式程序设计语言。Erlang也可以指Erlang/OTP的通称，开源电信平台（OTP）是Erlang的常用执行环境及一系列标准组件。

Erlang 执行环境为专有以下要求的系统设计：

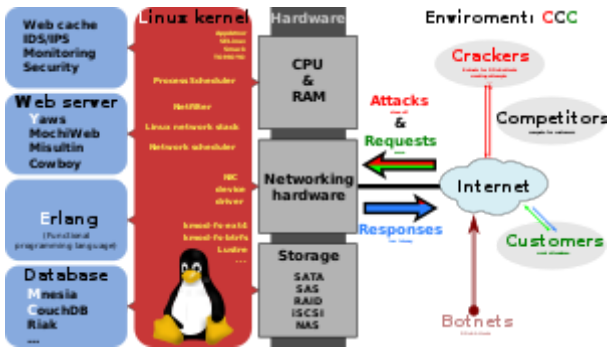
- 分布式运算
- 高故障容许度
- 软性即时运算
- 高可用性、不停运作的应用
- 热插拔，可以修改程序而不必停机

Erlang是运作于虚拟机的解释型语言，但是现在也包含有乌普萨拉大学高性能Erlang计划（HiPE）[4]开发的原生代码编译器，自R11B-4版本开始，Erlang也支持脚本方式执行。在编程范型上，Erlang属于多重典范编程语言，涵盖函数式、并行及分布式。循序执行的Erlang是一个及早求值, 单次赋值和动态类型的函数式编程语言。

它由乔·阿姆斯特朗（Joe Armstrong）在瑞典电信设备制造商爱立信所辖的电脑科学研究室开发，目的是创造一种可以应付大规模并发活动的程序设计语言和执行环境。Erlang于1987年发布正式版本，最早是爱立信拥有的私有软件，经过十年的发展，于1998年发表开放原始码版本。

Erlang

编程范型	多重典范：函数式、并发
设计者	乔·阿姆斯特朗、Robert Virding、Mike Williams
实现者	爱立信
发行时间	1986年
当前版本	26.0.2 (2023年6月29日)[1]
类型系统	动态、强
许可证	Apache许可证2.0（从OTP 18.0开始） Erlang公共许可协议1.1（早期版本）
文件扩展名	.erl .hrl
网站	www.erlang.org (http://www.erlang.org)
主要实现产品	
Erlang	
启发语言	
Prolog, Smalltalk, PLEX,[2] LISP	
影响语言	
Akka, Clojure, Dart, Elixir, F♯, Opa, Oz, Reia, Rust, Scala	
维基教科书中有关Erlang Programming的文本	



LYME is Erlang-based

目录

开发及演变历史

Erlang得名于丹麦数学家及统计学家Agner Krarup *Erlang*，同时Erlang还可以表示*Er*icsson *Lang*uage。Erlang语言由瑞典爱立信电信公司的乔·阿姆斯特朗开始设计，开始于公元一九八零年代。最初是以Prolog程序设计语言为基础，几度改版之后，改成以Joe's Abstract Machine为基础的独立语言执行环境。虽然语言风格仍与Prolog相近，不过因Erlang语言设计的走向，Erlang成为具备函数语言特色的程序设计语言[5]。

发行版本

1998年起，Erlang发布开放原始码版本，称为开源电信平台。**开源电信平台**采用修改过的Mozilla公共许可证协议发放，同时爱立信仍然提供商业版本的技术支持。目前，Erlang最大的商业用户是爱立信，其他知名用户有北电网络、亚马逊以及T-Mobile等[6]。

语言特色

- **并行程序设计** 在语言中，可以借由spawn/*函数，将特定的函数设置为独立的进程，之后可以做跨进程通信。
- **函数式程序设计** 由于Erlang早期以Prolog开发制成，受语言特性影响，即成为函数式语言。
- **单次赋值** 每个变量只能跟数据绑一次，所以，不像一般程序设计语言的变量可以多次指定为不同的值。单次赋值的好处是状态单纯，使程序容易阅读。
- **及早求值或严格求值** Erlang基本求值策略为电脑语言中及早求值之特性。而且，可以借由明确使用无参数的λ表达式，将特定函数设置为惰性求值策略。
- **动态数据类型与类型系统** 有编译时期的类型检查系统支持。
- **快速失败** 在执行时期发生的错误，会由错误位置提交消息，发生错误的进程立刻停止执行。借由进程通讯机制，可以自动传递错误、捕捉错误，使其他进程能够帮助处理错误。
- **代码热更新** 由于Erlang是函数语言，可以撰写特定的程序结构，制作即时更换新版函数的机制。
- **脚本语言** Erlang实现提供了脚本执行方式。

语言构成

Erlang程序结构以函数定义为主。函数是一组将输入分别对应到输出的规则，对应方式遵守数学函数的惯例。此外，Erlang语言由几项构句要素所组成，包括文字(或称原子)、数字、列表、值组、字符、字符串、二进制资料、模块、与特定用途的关键字如fun ... end, if ... end, case ... of ... end, spawn, !, receive ... end等等。以下段落分别列示并举例说明Erlang程序的基本构成部分，涵盖**资料格式**、**表达式格式**与**内置函数**。

资料格式

类型	意义与构词规则	例子
原子	原子是基本资料单元，以一般文字构成。构词规则有： 1. 以小写英文字符开头、不包含空白的连续文字。 2. 以单引号包含的任意连续文字。	▪ hello ▪ 'Hello, World!' ▪ true ▪ a3b
数字	数字是基本资料单元，可以是整数或实数。 1. 连续数字符号。 2. 包含一个小数点的连续数字符号，并不以小数点开头也不以小数点结尾。 3. 符合前二项原则，并以 + 或 - 符号开头。 4. 以#分割的数字，前者将表示进制。	▪ 302 ▪ 3.1416 ▪ +1 ▪ -2 ▪ 16#10
列表	列表是与链接序列相同的数据结构。任一列表大致区分为头部与尾部，头部是列表的第一项，尾部是列表除第一项之外的其他部分。 1. 左边以 [、右边以] 符号，包含一串以逗号分隔的零或多项构句要素。 2. 符合前项原则，当存在任一 | 符号时， | 的左边有一串逗号分隔的构句要素， | 的右边只有一个构句要素。	▪ [] ▪ [1,2,3] ▪ [[1],2|[]]
值组	值组是将二个、三个或多个资料放在一起的数据结构。 ▪ 左边以 { 、右边以 } 符号，包含一串以逗号分隔的零或多项构句要素。	▪ {} ▪ {{1},2}
字符	Erlang将字符存为32位的整数。 1. 任何可见的字符，以 $ 开头、后接该字符符号，即表示字符本身。 2. 任何不可见的字符，可使用以 $ 开头、后接该字符符号的转义序列表达。	▪ $3 ▪ $) ▪ $\012 ▪ $\x0A ▪ $\n
字符串	Erlang将字符串视同一列整数列表。 1. 以双引号包含任意多个文字，即为字符串。 2. 以一列整数列表表达，使其中每个整数项目都落在合理的字符的值范围，此列也是字符串。	▪ "Hello, World!" ▪ [65,66,67]
二进制资料	以左边 << 、右边 >> 符号，包含由位元语法 (http://www.erlang.org/doc/reference_manual/expressions.html#bit_syntax)（页面存档备份 (https://web.archive.org/web/20100613064301/http://www.erlang.org/doc/reference_manual/expressions.html#bit_syntax)，存于互联网档案馆）表示的资料。	▪ <<"Hello, World!">> ▪ <<65:8,66:8,67:8>>
函数识别项	Erlang容许用文字表示函数识别项，使程序中可以对指定函数做函数调用，或者当做资料传递。函数识别项格式为： ▪ fun 函数名称/参数数目	▪ fun a_function/3 用途见以下“函数式程序设计”小节。

程序代号	Erlang容许以内置函数erlang:spawn/3、erlang:spawn/4、erlang:spawn/1、erlang:spawn/2等等，将指定函数启动为一个程序。程序启动之后，Erlang以左边 < 、右边 > ，包含一个数字和点号组成的编号，表示此程序代号。	见以下“平行式程序设计”小节。

模块	Erlang容许将一些程序整理为一个模块。模块的设置，是在原始码文件开头书写模块标记，格式为：
	`-module(模組名稱).`
	`-export( [ 函數名稱/參數數目 , 函數名稱/參數數目 , ... ] ).`
	`-import( 模組名稱, [ 函數名稱/參數數目 , 函數名稱/參數數目 , ... ] ).`
	模块名称和函数名称都是原子。 -module(模块名称) 定义模块的名字，要与文件名相同。 -export(...) 定义模块发布的函数，模块内的任何函数必须要发布才能让外部透过模块调用该函数。 -import(...) 定义本模块要从其他模块导入哪些函数，以便本模块自己使用。另外，为了方便程序的撰写并测试，还容许 -compile(export_all) 定义本模块的所有函数全部对外发布。
	`-compile(export_all).`
		(略)

宏	宏是将一项资料以另一个文字做为代名。
	▪ 定义宏的语法是：
	`-define ( 代名 , 資料 ).`
	▪ 使用宏的语法是：
	`? 代名`
	▪ Erlang有一些内定语法，例如模块名称为
	`?MODULE`
		▪ -define(hello, world).
		test() -> ?hello.

表达式格式

类型	构词规则	例子
变量	变量是一种提供与资料绑定、赋值的词汇。Erlang的变量是单一赋值，一个变量只能赋值一次。 1. 以大写英文字符开头的任意连续文字，是具名变量。 2. 以 _ 开头的任意连续文字，是匿名变量，用于变量必须使用、但相对的值可以忽略的场合。	▪ Number1 ▪ _ ▪ _nothing
样式匹配	1. 样式是指以原子、列表或值组表达的结构，结构中可能包含一些未赋值的变量。 2. 给二个样式 A 和 B ，样式匹配是用 A = B 表示法，表示要让 A 对 B 匹配。如果匹配成功， A 包含的未赋值变量都会赋值，并且传回 B 的值。	▪ A = 42 ▪ {ok, Node} = {ok, 'Wikipedia'} ▪ [H|T] = [1,2,3]
函数	函数是由一或多项对应规则组成。每一项规则是将一部分匹配样式的输入映射到相对的输出。 ▪ 规则：格式为 原子 (變數 , 變數 , ...) -> 表達式 , 表達式 , ... 在 -> 左邊是函數名稱及搭配的參數列，右邊為函數本體。 ▪ 函数：格式为 規則 ; 規則 ; ... ; 規則 . 以分號分隔一或多項規則，並最後以句號結束。 同一函數的每一規則必須以相同的原子開頭，並接受相同數量的 參數列。 函数被调用时，会让调用方依序对被调用方的每一条函数规则做样式匹配，比对函数名称、参数数目、参数样式等等。首先完成匹配的函数规则会被执行，并且后面的函数规则会被忽略。	见以下“函数式程序设计”小节
函数调用	格式为 原子 (資料 , 資料 , ...) 表示函數名稱及搭配的參數列。呼叫符合函數名稱及 相同參數數目的函數。 函数调用时，所给予的参数可能是已赋值的变量。并且，如果参数是变量，必须是已赋值的变量。	见以下“函数式程序设计”小节

真值比较	▪ 比较运算： == 相等 /= 不相等 =< 小于或等于 < 小于 >= 大于或等于 > 大于 =:= 确实相等 =/= 确实不相等 ▪ 真值运算： not 非 and 且 or 或 xor 非此即彼 orelse 或 (快捷方式求值) andalso 且 (快捷方式求值) 真值比较的结果，如果成功则传回true原子，失败则传回false原子。 请记得，Erlang是以true和false表示布尔数据类型。	(略)
运算符	Erlang提供常用的运算符方便基本运算。运算符是用在**中序**的表达式里，包含 + - * / div(商) rem(余) 等。位元算算有 bnot, band, bor, bxor, bsl(算术左移), bsr(算术右移) 等。用于列表有 ++(列表衔接) --(列表剔除) 等。各种表达式皆可用 ( ) 调整运算优先级。	(略)
防卫式 (https://en.wikipedia.org/wiki/Guard_(computing))（页面存档备份 (https://web.archive.org/web/20110412054042/http://en.wikipedia.org/wiki/Guard_(computing))，存于互联网档案馆）	防卫式是接在when关键字之后的一组表达式，借由防卫式的真伪值做程控处理。 防卫式的原则如下方所述： 1. 代表true或false的变量或原子，是防卫式。 2. 任何真值表达式，包括比较算式和逻辑算式，是防卫式。 3. 传回true或false的函数调用，是防卫式。 4. 以逗号分隔的多个防卫式，是防卫式。	▪ false ▪ A =< 10 ▪ erlang:is_number(N), erlang:is_atom(A)
受防卫式限制的函数	函数对应规则格式为： `原子 ( 變數 , 變數 , ... ) -> 表達式 , 表達式 , ...`	▪ atom_pair(A, B) when is_atom(A), is_atom(B) -> {A, B}.

	若一条函数规则加上防卫式，此规则的处理范围会多一些限制。受防卫式限制的函数对应规则格式为： ``` 原子 (變數 , 變數 , ...) when 防衛式 -> 表達式 , 表達式 , ... ```	
行后注解	任何 % 符号开头，往后到行尾的文字皆为注解文字。	'H.W.'. % Hello, World!
λ演算式	λ演算式是匿名函数，在Erlang以 fun ... end 关键字叙述。格式为： ``` fun (變數 , 變數 , ...) -> 表達式 , 表達式 , ... end ``` 使用无参数的λ演算式，可以做出惰性求值的效果。	▪ (*fun(X)->X>0 end*)(1). % (λ x . x > 0) 1 ▪ lists:takewhile(*fun(X)->* *X>0 and X=<10 end*, [1,5,11]).
因果式	使用 if ... end 关键字叙述条件判断原则。格式为： ``` if 防衛式 -> 表達式, 表達式, ... ; 防衛式 -> 表達式, 表達式, ... ; ..... 防衛式 -> 表達式, 表達式, ... end ```	▪ parse(A) -> if is_number(A), round(A) == A, A >= 0 -> {natural, A}; is_number(A) -> {real, A}; is_atom(A) -> {word, A}; true -> {unknown, A} end.
案例式	使用 case ... of ... end 关键字，根据一个变量的案例，带往相对的进程。格式为： ``` case 表達式 of 樣式 -> 表達式, 表達式, ... ; 樣式 -> 表達式, 表達式, ... ; ..... 樣式 -> 表達式, 表達式, ... end ``` ▪ 样式之后可接when防卫式。	▪ show(A) -> case A of {natural, N} -> io:format("Natural number ~.10B is met.~n", [N]); {real, R} -> io:format("Real number ~f is met.~n", [R]); {word, W} ->

		io:format("\"~w\" is met.~n", [W]); {unknown, U} ->io:format("Unknown structure ~w.~n", [U]) end.

试误	使用 try ... catch ... end 关键字叙述试误的情况与结果。格式为：
	`try 表達式 of`
	`  樣式 -> 表達式, 表達式, ... ;`
	`  樣式 -> 表達式, 表達式, ... ;`
	`  ......`
	`  樣式 -> 表達式, 表達式, ...`
	`catch`
	`  樣式(例外) -> 表達式, 表達式, ... ;`
	`  樣式(例外) -> 表達式, 表達式, ... ;`
	`  ......`
	`  樣式(例外) -> 表達式, 表達式, ...`
	`after`
	`  表達式, 表達式, ...`
	`end`
	▪ 不需要使用after段落时，可省略after段落。
	▪ 样式之后可接when防卫式。
		(略)

接收消息	每个Erlang程序执行时，都可以从自己程序的邮箱中获取由其他程序送到的消息。可以使用 receive ... end 关键字接收消息，格式为：
	`receive`
	`  樣式 -> 表達式, 表達式, ... ;`
	`  樣式 -> 表達式, 表達式, ... ;`
	`  ......`
	`  樣式 -> 表達式, 表達式, ...`
	`end`
	▪ 样式之后可接when防卫式。
		▪ loop(FromPid, Result) ->receive{FromPid, stop} ->Result;{FromPid, Any} ->loop(FromPid, [Any|Result])end.

发送消息	Erlang容许向程序发送消息。使用！关键字，格式为： `程序代號 ! 訊息` ▪ 消息可以是各种资料格式。消息资料格式可以是用各种表达式求出的值。	▪ Pid = erlang:spawn(fun() ->receiveX -> Xend) ▪ 以上产生一个程序。 ▪ Pid ! {hello, world}

		▪ 以上对Pid提交消息。
列表解析 (https://en.wikipedia.org/wiki/List_comprehension)（页面存档备份 (https://web.archive.org/web/20100323163714/http://en.wikipedia.org/wiki/List_comprehension)，存于互联网档案馆）	列表解析，是提供快速创建列表的语法。语法等同于集合建构式 (https://en.wikipedia.org/wiki/Set-builder_notation)（页面存档备份 (https://web.archive.org/web/20100702150440/http://en.wikipedia.org/wiki/Set-builder_notation)，存于互联网档案馆）。格式为： ▪ [变量(表达式中的元素) || 变量(表达式中的元素) <- 表达式 , 防卫式] 若无防卫条件，列表解析中不写防卫式。	▪ [X || X <- [1,2,3]] 运算结果为[1,2,3]

内置函数

开源电信平台包括一个Erlang解释器、一个Erlang编译器、程序节点通信协议、CORBA、一个分布式数据库Mnesia (https://en.wikipedia.org/wiki/Mnesia)（页面存档备份 (https://web.archive.org/web/20100305050923/http://en.wikipedia.org/wiki/Mnesia)，存于互联网档案馆）、以及许多程序库[7]。**内置函数**涵盖了各种方面的功能，涵盖了系统命令、资料存取、格式转换、网络通信、图形接口、 ... 等。以下列表介绍几项常用的Erlang内置函数。(参阅文件 (http://www.erlang.org/doc/)（页面存档备份 (https://web.archive.org/web/20100612213537/http://www.erlang.org/doc/)，存于互联网档案馆）或索引 (http://www.erlang.org/doc/man_index.html) （页面存档备份 (https://web.archive.org/web/20220315092402/http://www.erlang.org/doc/man_index.html)，存于互联网档案馆）)

模块:函数名称 / 参数数目	用途

c:cd / 1	切换到指定目录位置。
	`> c:cd("D:\\code").`
	`D:/code/`
	`ok`
	当指定目录不正确时，则保持在原目录位置。

c:c / 1	编译指定的代码，之后加载新编译好的程序。
	`> c:c(test). % test.erl 必須存在於目錄位置`
	`{ok, test}`
	`> c:c(test1).`
	`./test1.erl:none: ...`
	`error`

io:format / 2	按照指定的格式文字将资料印在标准输出端口。
	`> io:format("~.8B, ~c, ~s, ~.2f~n", [32, $a, "hello", 3.1416]).`
	`40, a, hello, 3.14`
	`ok`

lists:sublist / 3	由列表中截取子列表。Erlang字符串是整数列表，于是本函数视同截取子字符串。
	`> lists:sublist("Hello, World!", 2, 2).`
	`"el"`

Hello World 程序

这是输出 Hello World 的一种方式：[8]

```
-module(hello).
-export([hello_world/0]).

hello_world() -> io:fwrite("hello, world\n").
```

若要编译这个程序，将它存为一个名为 hello.erl 的文字档，然后从 Erlang终端 进行编译。不要忘了在每个命令的最后加上一个句号（.）。例如：

```
Erlang (BEAM) emulator version 4.9.1 [source]
Eshell V4.9.1  (abort with ^G)
1> c(hello).
{ok,hello}
```

（在 Unix系统 上，你可以通过在命令行里输入 "erl" 来进入 Erlang终端。在 Windows系统 上，你需要打开一个 命令提示符 视窗，然后输入 "werl"来进入 Erlang终端，或者在程序功能表中找到 Erlang 的图标。）从 Erlang终端 上运行这个程序：

```
2> hello:hello_world().
hello, world
ok
```

函数式程序设计

Erlang支持函数式程序设计的一般特色，特色包括单次赋值、递归定义、λ演算与高阶函数等等。Erlang函数大致写法如下，以整数阶乘模块为例：

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

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

以下是快速排序算法的Erlang实现：

```
%% quicksort:qsort(List)
%% Sort a list of items
-module(quicksort).
-export([qsort/1]).

qsort([]) -> [];
qsort([Pivot|Rest]) ->
    qsort([ X || X <- Rest, X =< Pivot]) ++ [Pivot] ++ qsort([ Y || Y <- Rest, Y > Pivot]).
```

以下是费氏数列求解函数：

```
-module(example).
-export([fibo/1]).

fibo(N) when N > 1 ->
    fibo(N-1) + fibo(N-2);
fibo(1) ->
    1;
fibo(0) ->
    0.
```

```
> c(example).
{ok,example}
> lists:map(fun(X)->example:fibo(X) end, lists:seq(1,10)).
[1,1,2,3,5,8,13,21,34,55]
```

函数式程序设计难免以递归计算，而消耗了大量递归堆栈空间。为了克服这个问题，一般使用**累积参数**与**尾部递归**等技巧节省递归数目：如以下例子。

```
-module(test).
-export([fibo_accu/1]).

fibo_accu(N) ->
    fibo(N, 0, 1).
fibo(N, C1, C2) when N > 2 ->
    fibo(N-1, C2, C1+C2);
fibo(0, _, _) ->
    0;
fibo(1, _, _) ->
    1;
fibo(_, C1, C2) ->
    C1+C2.
```

```
> c(example).
{ok,test}
> lists:map(fun(X)->test:fibo_accu(X) end, lists:seq(1,10)).
[1,1,2,3,5,8,13,21,34,55]
```

函数式程序设计容许使用高阶函数求解。以下例子说明Erlang实做复合函数。（f o g，念作 f *after* g。）

```
'After'(F, G) ->
     fun(X) ->
         erlang:apply(F, [erlang:apply(G, [X])])
     end.
```

- 请注意**after**是Erlang关键字。因此，以上函数命名为*'After'*避开关键字。

```
> (example:'After'(fun test:show/1, fun test:parse/1))(3.1416).
Real number 3.141600 is met.
ok
```

平行式程序设计

Erlang最主要的特色是**面向并发程序设计**，强调多程序平行运作，并且以消息对彼此沟通[9]。Erlang提供了**spawn**函数和！、 **receive** ... **end** 等关键字，可以描述在Erlang/开源电信平台中的如何启动一些程序、并且如何让程序传递消息。此外，**面向并发程序设计**的精神还强调程序的容错处理，借由程序发生错误时的消息传递，使其他程序可以得知错误的发生，使方便于后续处理。以下分别介绍面向并发程序设计的一般程序撰写方式，以及错误处理的使用方式。

面向并发程序设计

基本的平行程序示范如下：

- 以下启动一个程序。

```
% create process and call the function web:start_server(Port, MaxConnections)
ServerProcess = spawn(web, start_server, [Port, MaxConnections]),
```

- 以下是在任何程序中，对先前起动的程序送一则消息 {*pause*, *10*} 。

```
% send the {pause, 10} message (a tuple with an atom "pause" and a number "10")
% to ServerProcess (asynchronously)
ServerProcess ! {pause, 10},
```

- 以下是一段接收消息的程序。每个程序都拥有一份邮箱，可伫留收到的消息； receive ... end 程序片断是从程序的邮箱中取出最早伫留的消息。

```
% receive messages sent to this process
receive
        a_message -> do_something;
        {data, DataContent} -> handle(DataContent);
        {hello, Text} -> io:format("Got hello message: ~s", [Text]);
        {goodbye, Text} -> io:format("Got goodbye message: ~s", [Text])
end.

收到 a_message 結果就是 do_something ; 收到 {data, DataContent} 結果會呼叫 handle(DataContent) ;
收到 {hello, Text} 結果教是印出 "Got hello message: ..." , 收到 {goodbye, Text} 結果是印出
"Got goodbye message: ..." 。
```

以下程序，示范产生一组环状传递消息的程序。

```
ring_proc(Funs) ->
    Ns = lists:seq(1, length(Funs)),
    [P|Pids] = [ spawn(?MODULE, lists:nth(Nth,Funs),[]) || Nth <- Ns ],
    [ Pid ! ToPid || {Pid, ToPid} <- lists:zip([P|Pids], Pids++[P]) ].

func() ->
    receive
        ToPid ->
            func_msg_(ToPid)
    end.

func_msg_(ToPid) ->
    receive
        stop ->
            io:format("Stop process ~w~n", [self()]),
            ToPid ! stop;
        Message ->
            io:format("~w: transmit message to ~w~n", [self(), ToPid]),
            ToPid ! Message,
            func_msg_(ToPid)
    end.

接收stop訊息，就對下一個程序送stop訊息；接收到其他任何訊息，就對下一個程序送同樣的訊息。
```

如果发送任何其他消息，就会让所有的程序不断对下一个程序传递消息。而以下是测试发送*stop*消息的执行结果。

```
> [P|_] = example:ring_proc([func,func,func]).
[<0.233.0>,<0.234.0>,<0.232.0>]
> P ! stop.
Stop process <0.233.0>
stop
Stop process <0.234.0>
> Stop process <0.232.0>
>
```

容错处理

Erlang容错处理机制，由二个步骤实现：一是将二个程序连接起来，二者之间存在一道通信管道，可提供错误消息的传递 —— 在此使用**link/1**函数；二是将程序回报错误的机制打开 —— 在此使用**process_flag/2**函数。

- 使用link(*Pid*)让程序连接到另一个程序。

```
-module(example).
-compile(export_all).
hello() ->
    Pid = spawn(?MODULE, world, []),
    link(Pid),
    ... .

執行時，以 Pid = spawn(example, hello, []) 啟動程序，此程序將啟動另一個程序，並且與它連接。
```

但以上程序还不会有错误消息的传递机制，因为回报错误的开关还没有打开。

- 开启程序回报错误机制。

```
以上 hello/0 函數前段使用process_flag/2函數，將trap_exit標籤打開，即可開啟程序回報錯誤機制。
```

```
hello() ->
    process_flag(trap_exit, true),
    Pid = spawn(?MODULE, world, []),
    link(Pid),
    ... .
```

于是，当程序结束时，会提交{'EXIT', *From*, *Reason*}资料。程序正常结束时，*Reason*为*normal*。

另外，**spawn**函数另外有程序连接版本，**spawn_link**函数，同时启动并连接到新程序。

分布式程序设计

Erlang提供分布式机制，能在另一台电脑启动一些Erlang程序，并由本机电脑对其他电脑的Erlang程序传递消息。

- 当启动Erlang环境时，加上一个网络节点名称，就进入分布式Erlang模式。节点可以使用端口号与其他节点通信。

```
$> erl -name node_1
```

- 在同一个网域中，网络节点名称可以使用短名。

```
$> erl -sname node_1
```

启动新的网络节点时，Erlang使用**epmd** (Erlang端口号对应管理系统) 指派端口号，提供节点使用。

当知道一个网络节点名称时，可以在该节点产生新程序。

- 在指定节点*RemoteNode*启动一个程序，**spawn**启动参数依序为节点名称、模块名称、函数名称、函数的参数表。

```
% create a remote process and call the function web:start_server(Port, MaxConnections)
% on machine RemoteNode
RemoteProcess = spawn(RemoteNode, web, start_server, [Port, MaxConnections]),

在遠端節點產生新程序之後，可以使用平行式程式設計的技巧，與遠端程序通訊。
```

Erlang / 开源电信平台提供的程序库，于分布式程序设计可以使用*net_adm*、*net_kernel*、*slave*、... 等模块，做网络通信[10]。

其他程序设计典范

惰性求值

Erlang程序员可以使用惰性求值。不过，必须使用λ演算式，才能做到惰性求值。

```
以下是惰性求值的一例：假設有個剖析器程式如下，由於及早求值特徵，本程式將不會求解。
expr() -> alt(then(factor(), then(literal($+), factor())),
              then(factor(), then(literal($-), factor()))).
factor() -> alt(then(term(), then(literal($*), term())),
                then(term(), then(literal($/), term()))).
term() -> alt(number(),
              xthen(literal($(), thenx(expr(), literal($))))).
此處使用λ演算式及適當使用函數名稱表示，就能進行求值。示例如下。
expr() ->
    fun () ->
          alt(then(fun factor/0, then(literal($+), fun factor/0)),
              then(fun factor/0, then(literal($-), fun factor/0)))
    end.
factor() ->
    fun () ->
           alt(then(fun term/0, then(literal($*), fun term/0)),
               then(fun term/0, then(literal($/), fun term/0)))
    end.
term() ->
    fun () ->
          alt(number(),
              xthen(literal($(), thenx(expr(), literal($)))))
    end.
```

应用

- Wings 3D，一个用Erlang编写的三维电脑图形软件。
- YAWS (http://yaws.hyber.org/)（页面存档备份 (https://web.archive.org/web/20100826033126/http://yaws.hyber.org/)，存于互联网档案馆），以Erlang编写的高效HTTP服务器。
- DISCO (http://discoproject.org/)（页面存档备份 (https://web.archive.org/web/20100916105505/http://discoproject.org/)，存于互联网档案馆），以Erlang编写的MapReduce架构系统。
- Apache CouchDB (http://couchdb.apache.org/)（页面存档备份 (https://web.archive.org/web/20110220214955/http://couchdb.apache.org/)，存于互联网档案馆），以Erlang编写的MapReduce文件式数据库系统。
- RabbitMQ (http://www.rabbitmq.com/)（页面存档备份 (https://web.archive.org/web/20110312041529/http://www.rabbitmq.com/)，存于互联网档案馆），能搭配Erlang运作的消息队列系统。
- 开放电信平台

- WhatsApp：其后端服务器应用使用了**Erlang**及FreeBSD[11]。支持了4.5亿的活跃用户
- ejabberd (http://www.process-one.net/en/ejabberd/)（页面存档备份 (https://web.archive.org/web/20140331224754/http://www.process-one.net/en/ejabberd/)，存于互联网档案馆）,世界上最流行的XMPP即时通讯服务器
- EMQX (https://github.com/emqx/emqx)（页面存档备份 (https://web.archive.org/web/20200529115641/https://github.com/emqx/emqx)，存于互联网档案馆），以Erlang编写的高可用、分布式MQTT消息服务器。

社区

- Erlang Central (https://web.archive.org/web/20141014063204/http://erlangcentral.org/)（英文）
- Erlang Resources 豆瓣小站 (http://site.douban.com/204209/)（页面存档备份 (https://web.archive.org/web/20150504114949/http://site.douban.com/204209/)，存于互联网档案馆）（简体中文）
- Erlang中文社区 erlang-china.org (https://web.archive.org/web/20080401034625/http://erlang-china.org/)（简体中文）
- Erlang中文教程 erlang-cn.com (http://www.erlang-cn.com/)（页面存档备份 (https://web.archive.org/web/20141014214421/http://www.erlang-cn.com/)，存于互联网档案馆）（简体中文）
- Erlang中文社区 cnerlang.com (https://web.archive.org/web/20141016141155/http://www.cnerlang.com/)（简体中文）
- Erlang中文 erlang-cn.org (https://web.archive.org/web/20141014051225/http://www.erlang-cn.org/)（简体中文）

参考资料

1. Release 26.0.2. 2023年6月29日 [2023年7月2日].
2. 18:30. [2018-05-05].（原始内容存档于2017-07-15）.
3. Releases. [2021-09-22].（原始内容存档于2021-08-18）.
4. High Performance Erlang. [2008-04-13].（原始内容存档于2011-06-16）.
5. Coders At Work. Book introduction. [2010-08-30].（原始内容存档于2010-03-05）.见*Coders At Work*一书对Joe Armstrong的口述记录。
6. Who uses Erlang for product development?. Frequently asked questions about Erlang. [2008-04-13].（原始内容存档于2008-04-19）. “*The largest user of Erlang is Ericsson. Ericsson use it to write software used in telecommunications systems. Many (dozens) projects have used it, a particularly large one is the extremely scalable AXD301 ATM switch.*” FAQ中列出的其他用户包括: Nortel、Deutsche Flugsicherung、T-Mobile等
7. 存档副本. [2010-09-03].（原始内容存档于2011-05-12）.
8. 译自官网 http://www.erlang.org/faq/getting_started.html （页面存档备份 (https://web.archive.org/web/20151004023544/http://www.erlang.org/faq/getting_started.html)，存于互联网档案馆）
9. Armstrong, Joe. Erlang. Communications of the ACM. September 2010, **53** (9): 68–75. doi:10.1145/1810891.1810910. “Erlang is conceptually similar to the occam programming language, though it recasts the ideas of CSP in a functional framework and uses asynchronous message passing.”

10. 参考分布式Erlang (http://www.erlang.org/doc/reference_manual/distributed.html) （页面存档备份 (https://web.archive.org/web/20101123232507/http://www.erlang.org/doc/reference_manual/distributed.html)，存于互联网档案馆），http://www.erlang.org/doc/reference_manual/distributed.html （页面存档备份 (https://web.archive.org/web/20101123232507/http://www.erlang.org/doc/reference_manual/distributed.html)，存于互联网档案馆）
11. **Erlang**及FreeBSD存档副本. [2014-02-22]. （原始内容存档于2014-02-25）.

延伸阅读

- Armstrong, Joe. Making reliable distributed systems in the presence of software errors (PDF). Ph.D. Dissertation. The Royal Institute of Technology, Stockholm, Sweden. 2003 [2016-02-13]. （原始内容存档 (PDF)于2015-03-23）. |url-status=和|dead-url=只需其一 (帮助)
- Armstrong, Joe. A history of Erlang. Proceedings of the third ACM SIGPLAN conference on History of programming languages – HOPL III. 2007: 6–1. ISBN 978-1-59593-766-7. doi:10.1145/1238844.1238850.
- Early history of Erlang (http://www.erlang.se/publications/bjarnelic.pdf)（页面存档备份 (https://web.archive.org/web/20190829000127/http://www.erlang.se/publications/bjarnelic.pdf)，存于互联网档案馆） by Bjarne Däcker
- Mattsson, H.; Nilsson, H.; Wikstrom, C. Mnesia – A distributed robust DBMS for telecommunications applications. First International Workshop on Practical Aspects of Declarative Languages (PADL '99). 1999: 152–163.
- Armstrong, Joe; Virding, Robert; Williams, Mike; Wikstrom, Claes. Concurrent Programming in Erlang 2nd. Prentice Hall. 1996-01-16: 358 [2019-10-16]. ISBN 978-0-13-508301-7. （原始内容存档于2012-03-06）.
- Armstrong, Joe. Programming Erlang: Software for a Concurrent World 1st. Pragmatic Bookshelf. 2007-07-11: 536. ISBN 978-1-934356-00-5.
- Thompson, Simon J.; Cesarini, Francesco. Erlang Programming: A Concurrent Approach to Software Development 1st. Sebastopol, California: O'Reilly Media, Inc. 2009-06-19: 496 [2020-01-20]. ISBN 978-0-596-51818-9. （原始内容存档于2019-10-16）.
- Logan, Martin; Merritt, Eric; Carlsson, Richard. Erlang and OTP in Action 1st. Greenwich, CT: Manning Publications. 2010-05-28: 500. ISBN 978-1-933988-78-8.
- Martin, Brown. Introduction to programming in Erlang, Part 1: The basics. developerWorks. IBM. 2011-05-10 [2011-05-10]. （原始内容存档于2019-10-16）.
- Martin, Brown. Introduction to programming in Erlang, Part 2: Use advanced features and functionality. developerWorks. IBM. 2011-05-17 [2011-05-17]. （原始内容存档于2019-10-16）.
- Wiger, Ulf. Four-fold Increase in Productivity and Quality: Industrial-Strength Functional Programming in Telecom-Class Products (PDF). FEmSYS 2001 Deployment on distributed architectures. Ericsson Telecom AB. 2001-03-30 [2014-09-16]. （原始内容存档 (PDF)于2019-08-19）.

外部链接

- Erlang开放原始码版本 (http://www.erlang.org/)（页面存档备份 (https://web.archive.org/web/20060613061633/http://www.erlang.org/)，存于互联网档案馆） （英文）

- Erlang爱立信授权版本 (http://www.erlang.se/)（页面存档备份 (https://web.archive.org/web/20080415124930/http://www.erlang.se/)，存于互联网档案馆）（英文）
- 因应软件错误建构可靠的分布式系统 (http://www.erlang.org/download/armstrong_thesis_2003.pdf)（页面存档备份 (https://web.archive.org/web/20080516051720/http://www.erlang.org/download/armstrong_thesis_2003.pdf)，存于互联网档案馆）（页面存档备份 (https://web.archive.org/web/20080516051720/http://www.erlang.org/download/armstrong_thesis_2003.pdf)，存于互联网档案馆）（英文）
- 开放式目录计划中和Erlang (https://dmoz-odp.org/Computers/Programming/Languages/Erlang)相关的内容
- Erlang教学 (http://www.tw511.com/2/20/701.html)（页面存档备份 (https://web.archive.org/web/20191116001849/http://www.tw511.com/2/20/701.html)，存于互联网档案馆）
