开始

案例、cond 和 if

在本章中，我们将了解 、 和控制流结构。 `case` `cond` `if`

case

`case` 允许我们将一个值与许多模式进行比较，直到找到匹配的模式：

```
iex> case {1, 2, 3} do
...>   {4, 5, 6} ->
...>     "This clause won't match"
...>   {1, x, 3} ->
...>     "This clause will match and bind x to 2 in this
clause"
...>   _ ->
...>     "This clause would match any value"
...> end
"This clause will match and bind x to 2 in this clause"
```

如果要对现有变量进行模式匹配，则需要使用运算符： `^`

```
iex> x = 1
1
iex> case 10 do
...>   ^x -> "Won't match"
...>   _ -> "Will match"
```

新闻： Elixir v1.15 发布

搜索。。。

接口文档

开始

1. 介绍
2. 基本类型
3. 基本运算符
4. 模式匹配
5. 案例、cond 和 if
6. 二进制文件、字符串和字符列表
7. 关键字列表和地图
8. 模块和功能
9. 递归
10. 枚举项和流
11. 过程
12. IO 和文件系统
13. 别名、要求和导入
14. 模块属性
15. 结构体
16. 协议
17. 理解
18. 印记
19. 尝试、捕捉和救援
20. 可选语法表
21. Erlang 库

```
...> end
"Will match"
```

条款还允许通过防护装置指定额外的条件：

```
iex> case {1, 2, 3} do
...>   {1, x, 3} when x > 0 ->
...>     "Will match"
...>   _ ->
...>     "Would match, if guard condition were not
satisfied"
...> end
"Will match"
```

上面的第一个子句仅在为正时匹配。`x`

请记住，防护装置中的错误不会泄漏，而只会使防护装置失效：

```
iex> hd(1)
** (ArgumentError) argument error
iex> case 1 do
...>   x when hd(x) -> "Won't match"
...>   x -> "Got #{x}"
...> end
"Got 1"
```

如果所有子句都不匹配，则会引发错误：

```
iex> case :ok do
...>   :error -> "Won't match"
...> end
** (CaseClauseError) no case clause matching: :ok
```

有关防护装置、防护装置的使用方式以及其中允许的表达式的详细信息，请参阅防护装置的完整文档。

注意匿名函数也可以有多个子句和防护：

```
iex> f = fn
...>   x, y when x > 0 -> x + y
```

```
...>   x, y -> x * y
...> end
#Function<12.71889879/2 in :erl_eval.expr/5>
iex> f.(1, 3)
4
iex> f.(-1, 3)
-3
```

每个匿名函数子句中的参数数必须相同，否则将引发错误。

```
iex> f2 = fn
...>   x, y when x > 0 -> x + y
...>   x, y, z -> x * y + z
...> end
** (CompileError) iex:1: cannot mix clauses with different
arities in anonymous functions
```

cond

`case` 当您需要匹配不同的值时很有用。但是，在许多情况下，我们希望检查不同的条件并找到第一个不计算为 或 的条件。在这种情况下，可以使用： `nil` `false` `cond`

```
iex> cond do
...>   2 + 2 == 5 ->
...>     "This will not be true"
...>   2 * 2 == 3 ->
...>     "Nor this"
...>   1 + 1 == 2 ->
...>     "But this will"
...> end
"But this will"
```

This is equivalent to clauses in many imperative languages - although used less frequently in Elixir. `else if`

If all of the conditions return or , an error () is raised. For this reason, it may be necessary to add a final condition, equal to , which will always match: `nil` `false` `CondClauseError` `true`

```
iex> cond do
...>   2 + 2 == 5 ->
...>     "This is never true"
...>   2 * 2 == 3 ->
...>     "Nor this"
...>   true ->
...>     "This is always true (equivalent to else)"
...> end
"This is always true (equivalent to else)"
```

Finally, note considers any value besides and to be true: `cond` `nil` `false`

```
iex> cond do
...>   hd([1, 2, 3]) ->
...>     "1 is considered as true"
...> end
"1 is considered as true"
```

`if` and `unless`

Besides and , Elixir also provides and , which are useful when you need to check for only one condition: `case` `cond` `if/2` `unless/2`

```
iex> if true do
...>   "This works!"
...> end
"This works!"
iex> unless true do
...>   "This will never be seen"
...> end
nil
```

如果给出的条件返回 or 或 ， 则在 - 之间给出的主体不执行，而是返回 。 相反的情况发生在 上。 `if/2` `false` `nil` `do` `end` `nil` `unless/2`

它们还支持块： `else`

```
iex> if nil do
...>   "This won't be seen"
...> else
```

```
...>   "This will"
...> end
"This will"
```

这也是讨论 Elixir 中的变量范围的好机会。如果在 、 和类似构造中声明或更改了任何变量，则声明和更改仅在构造中可见。例如： `if` `case`

```
iex> x = 1
1
iex> if true do
...>   x = x + 1
...> end
2
iex> x
1
```

在上述情况下，如果要更改值，则必须从 ： `if`

```
iex> x = 1
1
iex> x = if true do
...>   x + 1
...> else
...>   x
...> end
2
```

> *注意：关于 和 的一个有趣的说明是，它们是在语言中作为宏实现的;它们不像许多语言那样是特殊的语言结构。您可以在* `内核` *模块文档中查看文档和源代码。该模块也是定义运算符 like 和函数 like 的地方，默认情况下，所有这些都会自动导入并在代码中可用。*
> `if/2` `unless/2` `if/2` `Kernel` `+/2` `is_function/2`

我们已经结束了对 Elixir 中最基本的控制流结构的介绍。现在是时候谈谈“二进制文件、字符串和字符列表”了。