开始

模式匹配

在本章中，我们将展示 Elixir 中的运算符实际上如何成为匹配运算符，以及如何使用它来模式匹配数据结构内部。最后，我们将了解用于访问先前绑定值的引脚运算符。`=` `^`

匹配运算符

我们已经多次使用运算符在 Elixir 中分配变量：`=`

```
iex> x = 1
1
iex> x
1
```

在 Elixir 中，操作员实际上被称为*匹配运算符*。让我们看看为什么：`=`

```
iex> x = 1
1
iex> 1 = x
1
iex> 2 = x
** (MatchError) no match of right hand side value: 1
```

请注意，这是一个有效的表达式，并且它匹配，因为左侧和右侧都等于 1。当两侧不匹配时，将升起 a。`1 = x` `MatchError`

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

变量只能在 的左侧赋值：`=`

```
iex> 1 = unknown
** (CompileError) iex:1: undefined variable "unknown"
```

模式匹配

match 运算符不仅用于匹配简单值，还可用于解构更复杂的数据类型。例如，我们可以在元组上进行模式匹配：

```
iex> {a, b, c} = {:hello, "world", 42}
{:hello, "world", 42}
iex> a
:hello
iex> b
"world"
```

如果边无法匹配，则会发生模式匹配错误，例如，如果元组具有不同的大小：

```
iex> {a, b, c} = {:hello, "world"}
** (MatchError) no match of right hand side value:
{:hello, "world"}
```

在比较不同类型的类型时，例如，如果将左侧的元组与右侧的列表匹配：

```
iex> {a, b, c} = [:hello, "world", 42]
** (MatchError) no match of right hand side value:
[:hello, "world", 42]
```

更有趣的是，我们可以匹配特定的值。下面的示例断言，只有当右侧是以原子开头的元组时，左侧才会与右侧匹配：`:ok`

```
iex> {:ok, result} = {:ok, 13}
{:ok, 13}
iex> result
13
```

```
iex> {:ok, result} = {:error, :oops}
** (MatchError) no match of right hand side value:
{:error, :oops}
```

我们可以在列表上进行模式匹配：

```
iex> [a, b, c] = [1, 2, 3]
[1, 2, 3]
iex> a
1
```

列表还支持在自己的头和尾上进行匹配：

```
iex> [head | tail] = [1, 2, 3]
[1, 2, 3]
iex> head
1
iex> tail
[2, 3]
```

与 and 函数类似，我们无法将空列表与头尾模式匹配：`hd/1` `tl/1`

```
iex> [head | tail] = []
** (MatchError) no match of right hand side value: []
```

该格式不仅用于模式匹配，还用于将项目预置到列表中：`[head | tail]`

```
iex> list = [1, 2, 3]
[1, 2, 3]
iex> [0 | list]
[0, 1, 2, 3]
```

模式匹配允许开发人员轻松解构元组和列表等数据类型。正如我们将在以下章节中看到的，它是 Elixir 中递归的基础之一，也适用于其他类型，如映射和二进制文件。

引脚运算符

Elixir 中的变量可以反弹：

```
iex> x = 1
1
iex> x = 2
2
```

但是，有时我们不希望变量反弹。

如果要根据变量*的现有值*进行模式匹配，而不是重新绑定变量，请使用引脚运算符。 `^`

```
iex> x = 1
1
iex> ^x = 2
** (MatchError) no match of right hand side value: 2
```

因为我们在绑定到的值时固定了 ，所以它等效于以下内容： `x` `1`

```
iex> 1 = 2
** (MatchError) no match of right hand side value: 2
```

请注意，我们甚至会看到完全相同的错误消息。

我们可以在其他模式匹配中使用 pin 运算符，例如元组或列表：

```
iex> x = 1
1
iex> [^x, 2, 3] = [1, 2, 3]
[1, 2, 3]
iex> {y, ^x} = {2, 1}
{2, 1}
iex> y
2
iex> {y, ^x} = {2, 2}
** (MatchError) no match of right hand side value: {2, 2}
```

由于绑定到固定时的值，因此最后一个示例可以编写为： `x` `1`

```
iex> {y, 1} = {2, 2}
```

```
** (MatchError) no match of right hand side value: {2, 2}
```

如果一个变量在模式中被多次提及，则所有引用都应绑定到相同的值：

```
iex> {x, x} = {1, 1}
{1, 1}
iex> {x, x} = {1, 2}
** (MatchError) no match of right hand side value: {1, 2}
```

在某些情况下，您并不关心模式中的特定值。通常的做法是将这些值绑定到下划线。例如，如果只有列表的头部对我们很重要，我们可以将尾部分配给下划线：`_`

```
iex> [head | _] = [1, 2, 3]
[1, 2, 3]
iex> head
1
```

该变量的特殊之处在于它永远无法读取。尝试从中读取会给出编译错误：`_`

```
iex> _
** (CompileError) iex:1: invalid use of _. "_" represents
a value to be ignored in a pattern and cannot be used in
expressions
```

尽管模式匹配允许我们构建强大的构造，但它的使用是有限的。例如，不能在匹配项的左侧进行函数调用。以下示例无效：

```
iex> length([1, [2], 3]) = 3
** (CompileError) iex:1: cannot invoke remote function
:erlang.length/1 inside match
```

我们对模式匹配的介绍到此结束。正如我们将在下一章中看到的，模式匹配在许多语言结构中非常普遍。