开始

可选语法表

到目前为止，在本指南中，我们了解到 Elixir 语法允许开发人员在某些情况下省略分隔符，以使代码更具可读性。例如，我们了解到括号是可选的：

```
iex> length([1, 2, 3]) == length [1, 2, 3]
true
```

并且 - 块等同于关键字列表：`do` `end`

```
# do-end blocks
iex> if true do
...>   :this
...> else
...>   :that
...> end
:this

# keyword lists
iex> if true, do: :this, else: :that
:this
```

关键字列表使用 Elixir 的常规符号来分隔参数，其中我们用逗号分隔每个键值对，每个键后跟。在 -blocks 中，我们去掉了冒号、逗号，并用换行符分隔每个关键字。它们之所以有用，正是因为它们消除了编写代码块时的冗长。大多数时候，我们使用块语法，但很高兴知道它们是等效的。`:` `do`

这些便利，我们在这里称之为“可选语法”，允许语言语法核心很小，而不会牺牲代码的可读性和表现力。在这个简短的章节中，我们将回顾该语言提供的四个规则，使用一个简短的片段作为游乐场。

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

演练

采用以下代码：

```
if variable? do
  Call.this()
else
  Call.that()
end
```

现在让我们一一删除便利：

1. `do - end` 块等效于关键字：

```
if variable?, do: Call.this(), else: Call.that()
```

2. 关键字列表作为最后一个参数不需要方括号，但让我们添加它们：

```
if variable?, [do: Call.this(), else: Call.that()]
```

3. 关键字列表与双元素元组列表相同：

```
if variable?, [{:do, Call.this()}, {:else,
Call.that()}]
```

4. 最后，括号在函数调用中是可选的，但让我们添加它们：

```
if(variable?, [{:do, Call.this()}, {:else,
Call.that()}])
```

就是这样！这四条规则概述了 Elixir 中可用的可选语法。

为了理解为什么这些规则很重要，我们可以简单地将 Elixir 与许多其他编程语言进行比较。大多数编程语言都有几个关键字，用于定义方法、函数、条件、循环等。这些关键字中的每一个都有自己的语法规则。

然而，在 Elixir 中，这些语言功能都不需要特殊的“关键字”，而是它们都是从这一小组规则构建的。另一个好处是开发人员还可以以与语言本身一致的方式扩展语言，因为设计和扩展语言的构造是相同的。我们将在 Elixir 中的元编程指南中进一步探讨这个主题。

归根结底，这些规则使我们能够编写：

```
defmodule Math do
  def add(a, b) do
    a + b
  end
end
```

而不是：

```
defmodule(Math, [
  {:do, def(add(a, b), [{:do, a + b}])}
])
```

每当您有任何问题时，这个快速演练都能满足您的需求。

最后，如果您担心何时应用这些规则，值得注意的是，Elixir 格式化程序会为您处理这些问题。大多数Elixir开发人员使用 混合格式 任务根据Elixir团队和社区定义的一组明确定义的规则来格式化他们的代码库。例如，将始终向函数调用添加括号，除非明确配置为不这样做。这有助于在组织和更广泛的社区内保持所有代码库的一致性。`mix format`

← 上一页　返回页首　下一→