开始

二进制文件、字符串和字符列表

1 统一码和码位

2 UTF-8 和编码

3 位串

4 二进制文件

5 字符学家

在“基本类型”中，我们学习了一些关于字符串的知识，并使用了该函数进行检查：`is_binary/1`

```
iex> string = "hello"
"hello"
iex> is_binary(string)
true
```

在本章中，我们将清楚地了解二进制文件到底是什么，它们与字符串的关系，以及单引号值在 Elixir 中的含义。尽管字符串是计算机语言中最常见的数据类型之一，但它们非常复杂，并且经常被误解。要理解 Elixir 中的字符串，我们必须了解 Unicode 和字符编码，特别是 UTF-8 编码。`'like this'`

统一码和码位

为了促进多种语言的计算机之间的有意义通信，需要一个标准，以便一台机器上的 1 和 0 在传输到另一台机器时意味着相同的事情。Unicode 标准充当我们所知的几乎所有字符的官方注册表：这包括古典和历史文本中的字符、表情符号以及格式化和控制字符。

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

Unicode 将其曲目中的所有字符组织成代码图表，每个字符都有一个唯一的数字索引。此数字索引称为代码点。

在 Elixir 中，你可以在字符文字前面使用一个来显示其代码点：`?`

```
iex> ?a
97
iex> ?ł
322
```

请注意，大多数 Unicode 代码图表将通过十六进制 （hex） 表示来引用代码点，例如 翻译成十六进制，我们可以通过使用其码位编号的符号和十六进制表示来表示 Elixir 字符串中的任何 Unicode 字符：`97` `0061` `\uXXXX`

```
iex> "\u0061" == "a"
true
iex> 0x0061 = 97 = ?a
97
```

十六进制表示还将帮助您查找有关码位的信息，例如 https://codepoints.net/U+0061 有一个关于小写的数据表，也就是码位 97。`a`

UTF-8 和编码

现在我们了解了 Unicode 标准是什么以及什么是码位，我们终于可以谈谈编码了。代码点是我们存储的内容，而编码处理我们如何存储它：编码是一种实现。换句话说，我们需要一种机制将代码点数字转换为字节，以便它们可以存储在内存中，写入磁盘等。

Elixir 使用 UTF-8 对其字符串进行编码，这意味着代码点被编码为一系列 8 位字节。UTF-8 是一种可变宽度字符编码，它使用 <> 到 <> 个字节来存储每个代码点。它能够对所有有效的 Unicode 码位进行编码。让我们看一个例子：

```
iex> string = "héllo"
"héllo"
iex> String.length(string)
5
```


22. 调试
23. 类型规格和行为
24. 下一步去哪里

混合和一次性密码

1. 混音简介
2. 代理
3. GenServer
4. 主管和申请
5. 动态主管
6. 电子交易体系
7. 依赖项和伞形项目
8. 任务和 gen_tcp
9. 文档测试，模式和
10. 分布式任务和标签
11. 配置和发布

ELIXIR 中的元编程

1. 报价和取消报价
2. 宏
3. 域特定语言

```
iex> byte_size(string)
6
```

虽然上面的字符串有 5 个字符，但它使用 6 个字节，因为两个字节用于表示字符。`é`

> *注意：如果您在 Windows 上运行，则默认情况下您的终端可能不使用 UTF-8。您可以通过在输入 `（）` 之前运行来更改当前会话的编码。`chcp 65001` `iex` `iex.bat`*

除了定义字符外，UTF-8 还提供了字素的概念。字素可能由多个字符组成，这些字符通常被视为一个字符。例如，女消防员表情符号表示为三个字符的组合：女性表情符号 （）、隐藏的零宽度连接器和消防车表情符号（👩 🚒）：

```
iex> String.codepoints("👩‍🚒")
["👩", "‍", "🚒"]
iex> String.graphemes("👩‍🚒")
["👩‍🚒"]
```

然而，Elixir 足够聪明，知道它们被视为一个单一的角色，因此长度仍然是一个：

```
iex> String.length("👩‍🚒")
1
```

> *注意：如果您在终端中看不到上面的表情符号，则需要确保您的终端支持表情符号，并且您使用的字体可以呈现它们。*

尽管这些规则听起来很复杂，但 UTF-8 编码的文档无处不在。此页面本身以 UTF-8 编码。编码信息将提供给您的浏览器，然后浏览器知道如何相应地呈现所有字节、字符和字素。

如果要查看字符串将存储在文件中的确切字节，一个常见的技巧是将空字节连接到它：`<<0>>`

```
iex> "hełło" <> <<0>>
<<104, 101, 197, 130, 197, 130, 111, 0>>
```

或者，可以使用 IO.inspect/2 查看字符串的二进制表示形式：

```
iex> IO.inspect("hełło", binaries: :as_binaries)
<<104, 101, 197, 130, 197, 130, 111>>
```

我们有点超前了。让我们谈谈位串，以了解构造函数的确切含义。 `<<>>`

位串

虽然我们已经介绍了码位和 UTF-8 编码，但我们仍然需要更深入地了解如何准确存储编码字节，这就是我们介绍位串的地方。位串是Elixir中的基本数据类型，用语法表示。位字符串是内存中连续的位序列。 `<<>>`

关于二进制/位字符串构造函数的完整参考可以在Elixir 文档中找到。 `<<>>`

默认情况下，8 位（即 1 个字节）用于将每个数字存储在位字符串中，但您可以通过修饰符手动指定位数以表示以位为单位的大小，或者您可以使用更详细的声明： `::n` `n` `::size(n)`

```
iex> <<42>> == <<42::8>>
true
iex> <<3::4>>
<<3::size(4)>>
```

例如，以 4 为底的 2 位表示的十进制数为 ，相当于值 、 ，每个值使用 1 位存储： `3` `0011` `0` `0` `1` `1`

```
iex> <<0::1, 0::1, 1::1, 1::1>> == <<3::4>>
true
```

任何超过预配位数可以存储的值都将被截断：

```
iex> <<1>> == <<257>>
true
```

在这里，基数 257 中的 2 将表示为 ，但由于我们只保留了 8 位用于表示（默认情况下），最左边的位将被忽略，并且该值被截断为 ，或者只是十进制。`100000001` `00000001` `1`

二进制文件

二进制是位串，其中位数可被 8 整除。这意味着每个二进制文件都是一个位串，但不是每个位串都是二进制文件。我们可以使用 and 函数来演示这一点。`is_bitstring/1` `is_binary/1`

```
iex> is_bitstring(<<3::4>>)
true
iex> is_binary(<<3::4>>)
false
iex> is_bitstring(<<0, 255, 42>>)
true
iex> is_binary(<<0, 255, 42>>)
true
iex> is_binary(<<42::16>>)
true
```

我们可以在二进制文件 / 位串上进行模式匹配：

```
iex> <<0, 1, x>> = <<0, 1, 2>>
<<0, 1, 2>>
iex> x
2
iex> <<0, 1, x>> = <<0, 1, 2, 3>>
** (MatchError) no match of right hand side value: <<0, 1,
2, 3>>
```

请注意，除非显式使用修饰符，否则二进制模式中的每个条目都应匹配单个字节（正好 8 位）。如果我们想匹配未知大小的二进制文件，我们可以在模式末尾使用修饰符：`::` `binary`

```
iex> <<0, 1, x::binary>> = <<0, 1, 2, 3>>
<<0, 1, 2, 3>>
iex> x
<<2, 3>>
```

在对二进制文件进行模式匹配时，还有其他几个修饰符很有用。修饰符将匹配二进制文件中的字节：`binary-size(n)` `n`

```
iex> <<head::binary-size(2), rest::binary>> = <<0, 1, 2,
3>>
<<0, 1, 2, 3>>
iex> head
<<0, 1>>
iex> rest
<<2, 3>>
```

字符串是 **UTF-8 编码的二进制文件**，其中每个字符的代码点使用 1 到 4 个字节进行编码。因此，每个字符串都是二进制文件，但由于 UTF-8 标准编码规则，并非每个二进制文件都是有效的字符串。

```
iex> is_binary("hello")
true
iex> is_binary(<<239, 191, 19>>)
true
iex> String.valid?(<<239, 191, 19>>)
false
```

字符串串联运算符实际上是一个二进制连接运算符：`<>`

```
iex> "a" <> "ha"
"aha"
iex> <<0, 1>> <> <<2, 3>>
<<0, 1, 2, 3>>
```

鉴于字符串是二进制文件，我们也可以在字符串上进行模式匹配：

```
iex> <<head, rest::binary>> = "banana"
"banana"
iex> head == ?b
true
iex> rest
"anana"
```

但是，请记住，二进制模式匹配适用于字节，因此将字符串（如“über”）与多字节字符的匹配不会与字符匹配，它将*在该字符的第一个字节*上匹配：

```
iex> "ü" <> <<0>>
<<195, 188, 0>>
iex> <<x, rest::binary>> = "über"
"über"
iex> x == ?ü
false
iex> rest
<<188, 98, 101, 114>>
```

上面，仅在多字节字符的第一个字节上匹配。`x` `ü`

因此，在字符串上进行模式匹配时，使用修饰符很重要：`utf8`

```
iex> <<x::utf8, rest::binary>> = "über"
"über"
iex> x == ?ü
true
iex> rest
"ber"
```

字符学家

我们对位串、二进制文件和字符串的浏览已接近完成，但我们还有一种数据类型需要解释：字符列表。

字符列表是整数列表，其中所有整数都是有效的代码点。在实践中，您不会经常遇到它们，只会在特定场景中遇到它们，例如与不接受二进制文件作为参数的旧 Erlang 库接口。

```
iex> ~c"hello"
~c"hello"
iex> [?h, ?e, ?l, ?l, ?o]
~c"hello"
```

符号（我们将在后面的“符号”部分中介绍符号） 表示我们正在处理的是字符列表而不是常规字符串的事实。 `~c`

双引号创建字符串，而单引号创建字符列表文本。 字符列表曾经在 Elixir <1.15 中用单引号表示：

```
iex> 'hello'
~c"hello"
```

关键要点是这与 .一般来说，双引号必须始终用于表示 **Elixir** 中的字符串。无论如何，让我们了解字符列表的工作原理。 `"hello"` `'hello'`

字符列表不包含字节，而是包含整数代码点。但是，如果所有码位都在 ASCII 范围内，则仅以单引号打印列表：

```
iex> ~c"hełło"
[104, 101, 322, 322, 111]
iex> is_list(~c"hełło")
true
```

将整数解释为代码点可能会导致一些令人惊讶的行为。例如，如果要存储介于 0 和 127 之间的整数列表，则默认情况下，IEx 会将其解释为字符列表，并显示相应的 ASCII 字符。

```
iex> heartbeats_per_minute = [99, 97, 116]
~c"cat"
```

您可以使用 and 函数将字符列表转换为字符串并返回：
`to_string/1` `to_charlist/1`

```
iex> to_charlist("hełło")
[104, 101, 322, 322, 111]
iex> to_string(~c"hełło")
"hełło"
iex> to_string(:hello)
"hello"
iex> to_string(1)
"1"
```

请注意，这些函数是多态的 - 它们不仅将字符转换为字符串，还对整数、原子等进行操作。

字符串（二进制）连接使用运算符，但字符列表作为列表，使用列表连接运算符：`<>` `++`

```
iex> ~c"this " <> ~c"fails"
** (ArgumentError) expected binary argument in <> operator
but got: ~c"this "
    (elixir) lib/kernel.ex:1821:
Kernel.wrap_concatenation/3
    (elixir) lib/kernel.ex:1808:
Kernel.extract_concatenations/2
    (elixir) expanding macro: Kernel.<>/2
    iex:1: (file)
iex> ~c"this " ++ ~c"works"
~c"this works"
iex> "he" ++ "llo"
** (ArgumentError) argument error
    :erlang.++("he", "llo")
iex> "he" <> "llo"
"hello"
```

有了二进制文件、字符串和字符列表，是时候讨论键值数据结构了。

← 上一页　返回页首　下一→