Erlang 库

Elixir 提供了与 Erlang 库的出色互操作性。事实上 Elixir 不鼓励简单地包装 Erlang 库，而直接包装 Erlang 库 与 Erlang 代码接口。在本节中，我们将介绍一些 Elixir 中没有的最常见和最有用的 Erlang 功能。

随着你越来越精通Elixir，你可能想在更多中探索Erlang STDLIB 参考手册。 细节。

二进制模块

内置的Elixir String模块处理UTF-8编码的二进制文件。二进制模块在以下情况下很有用 您正在处理不一定是 UTF-8 编码的二进制数据。

```
iex> String.to_charlist "Ø"
[216]
iex> :binary.bin_to_list "Ø"
[195, 152]
```

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

上面的例子显示了差异;模块返回 Unicode 代码点，同时处理原始数据字节。 `String` `:binary`

格式化文本输出

Elixir不包含类似于C和其他函数的功能 语言。幸运的是，Erlang 标准库可以正常工作并且可以使用。第一种格式为终端输出，而 第二种格式为IOLIST。格式说明符与 不同，有关详细信息，请参阅 Erlang 文档。 `printf` `:io.format/2` `:io_lib.format/2` `printf`

```
iex> :io.format("Pi is approximately given by:~10.3f~n",
[:math.pi])
Pi is approximately given by:     3.142
:ok
iex> to_string :io_lib.format("Pi is approximately given
by:~10.3f~n", [:math.pi])
"Pi is approximately given by:     3.142\n"
```

另请注意，Erlang 的格式化函数需要特别注意 统一码处理。

加密模块

加密模块包含哈希 功能、数字签名、加密等：

```
iex> Base.encode16(:crypto.hash(:sha256, "Elixir"))
"3315715A7A3AD57428298676C5AE465DADA38D951BDFAC9348A8A31E9
C7401CB"
```

该模块是附带的应用程序的一部分 二郎。这意味着您必须将应用程序列为附加应用程序 项目配置中的应用程序。为此，请编辑您的文件以包括： `:crypto` `:crypto` `:crypto` `mix.exs`

```
def application do
  [extra_applications: [:crypto]]
end
```

不属于 或 Erlang 应用程序的任何模块 必须在您的。你可以找到 Erlang 文档中任何 Erlang 模块的应用程序名称，立即 在侧边栏中的 Erlang 徽标

下方。`:kernel` `:stdlib` `mix.exs`

二合字母模块

二合字母模块（以及 digraph_utils）包含 用于处理由顶点和边构建的有向图的函数。 构建图后，其中的算法将有助于找到， 例如，两个顶点之间的最短路径或图形中的循环。

给定三个顶点，找到从第一个到最后一个的最短路径。

```
iex> digraph = :digraph.new()
iex> coords = [{0.0, 0.0}, {1.0, 0.0}, {1.0, 1.0}]
iex> [v0, v1, v2] = (for c <- coords, do:
:digraph.add_vertex(digraph, c))
iex> :digraph.add_edge(digraph, v0, v1)
iex> :digraph.add_edge(digraph, v1, v2)
iex> :digraph.get_short_path(digraph, v0, v2)
[{0.0, 0.0}, {1.0, 0.0}, {1.0, 1.0}]
```

请注意，中的函数就地改变了图结构，这 是可能的，因为它们是作为 ETS 表实现的，如下所述。`:digraph`

Erlang 术语存储

模块 ets 和 dets 处理大型存储 数据结构分别位于内存或磁盘上。

ETS 允许您创建包含元组的表。默认情况下，ETS 表 受保护，这意味着只有所有者进程可以写入表 但任何其他进程都可以读取。ETS 具有一些功能，允许 用作简单数据库、键值存储或缓存的表 机制。

模块中的函数会将表的状态修改为 副作用。`ets`

```
iex> table = :ets.new(:ets_test, [])
# Store as tuples with {name, population}
iex> :ets.insert(table, {"China", 1_374_000_000})
iex> :ets.insert(table, {"India", 1_284_000_000})
iex> :ets.insert(table, {"USA", 322_000_000})
iex> :ets.i(table)
<1   > {<<"India">>,1284000000}
```

```
<2    > {<<"USA">>,322000000}
<3    > {<<"China">>,1374000000}
```

数学模块

数学模块包含通用 涵盖三角函数、指数和对数的数学运算 功能。

```
iex> angle_45_deg = :math.pi() * 45.0 / 180.0
iex> :math.sin(angle_45_deg)
0.7071067811865475
iex> :math.exp(55.0)
7.694785265142018e23
iex> :math.log(7.694785265142018e23)
55.0
```

队列模块

队列是一种数据结构，可有效地实现（双端）FIFO（先进先出）队列：

```
iex> q = :queue.new
iex> q = :queue.in("A", q)
iex> q = :queue.in("B", q)
iex> {value, q} = :queue.out(q)
iex> value
{:value, "A"}
iex> {value, q} = :queue.out(q)
iex> value
{:value, "B"}
iex> {value, q} = :queue.out(q)
iex> value
:empty
```

兰德模块

兰德具有返回功能随机值并设置随机种子。

```
iex> :rand.uniform()
0.8175669086010815
iex> _ = :rand.seed(:exs1024, {123, 123534, 345345})
iex> :rand.uniform()
```

```
0.5820506340260994
iex> :rand.uniform(6)
6
```

zip 和 zlib 模块

`zip` 模块可让您读写 ZIP 文件传入和传出磁盘或内存，以及提取文件信息。

此代码计算 ZIP 文件中的文件数：

```
iex> :zip.foldl(fn _, _, _, acc -> acc + 1 end, 0,
:binary.bin_to_list("file.zip"))
{:ok, 633}
```

zlib 模块处理 `zlib` 格式的数据压缩，如命令所示。 `gzip`

```
iex> song = "
...> Mary had a little lamb,
...> His fleece was white as snow,
...> And everywhere that Mary went,
...> The lamb was sure to go."
iex> compressed = :zlib.compress(song)
iex> byte_size song
110
iex> byte_size compressed
99
iex> :zlib.uncompress(compressed)
"\nMary had a little lamb,\nHis fleece was white as
snow,\nAnd everywhere that Mary went,\nThe lamb was sure
to go."
```

现在让我们来看看调试时可能使用的现有 Elixir（和 Erlang）库。