Elixir语言学习笔记

aborn

Published
with GitBook

目錄

Elixir语言学习笔记

在线阅读，笔记网址：http://nle.popkit.org/。

为什么有这个笔记？

一方面，elixir的中文资源很少。另一方面，也可以记录自己在学习elixir的一些笔记。

大家一起来参与

参与方式： 提交一次pull/request，然后我会把你加入到Collaborators里。

注意事项

注意这个项目的作用不是对http://elixir-lang.org/官网 的简单中文翻译。它用来记录一些核心的语法，及高级主题。对于很简单，或者能用语言都有的部分 不会纳入其中。

代码

本项目所有的代码在nle目录下

更新时间

2016-02-21

第一章节

这是第一章节 GitBook allows you to organize your book into chapters, each chapter is stored in a separate file like this one.

基本数据类型

elixir的基本数据类型有：整型(integer)、浮点型(float)、布尔类型(boolean)、 原子类型(atom)、字符串(string)、列表(list)、元组(tuple).

这里的整型、浮点、布尔类型和其他语言差不多。下面是差异部分。

整型的除法

整型的除法和有些语言略有不同：如下10/2得到的结果是一个浮点类型

```
iex(5)> 10/2
5.0
```

如果想整除取整,则采用div/2,整除取余，则采用rem/2

原子类型

原子类型是一种常量，它的名字即它值本身,下面是一个例子：

```
iex(7)> :atom
:atom
```

注意:原子类型以:开头，并且它是内存常驻的，beam虚拟机不会对这部分内存做gc操作.

列表

elixir的列表相当于java里的linked list，但列表里的每个元素可以为任意类型:

```
iex(1)> a = [1,2, "3",4]
[1, 2, "3", 4]
```

列表是由中括号括起来的部分，这点和java里的写法很不一样。 对列表取表头和表尾操作分别是hd/1和tl/1函数.

元组

元组相当于java里的数组，但它的写法是用大括号括起来，如下：

```
iex(18)> a = {:ok, "hello"}
{:ok, "hello"}
iex(19)> elem(a,1)
"hello"
```

元组在内存中是连续存储的,它的每个元素也可以为任意类型.这里取第n个元素是采用 elem/2 这个函数来操作的。

elixir的异常处理

elixir有三种异常处理的机制 分别是Errors, throws和exits

errors

errors相当于java里的exception，不过，它是运行时抛出的，如下面的例子：

```
iex(1)> String.to_integer("s")
** (ArgumentError) argument error
:erlang.binary_to_integer("s")
```

在运行时，想要抛出一个异常，只要调用raise/1，参数为消息

```
iex(1)> raise "exception example"
** (RuntimeError) exception example
```

定义自己的exception模块，相当简单，下面是一个例子：

```
defmodule Nle.ExceptionExample do
  defexception message: "example exception."
end
```

异常的捕获

对elixir的异常捕获,可采用try/rescue机制.下面是一个例子：

```
def demo(arg) do
  try do
    case arg do
      0 ->
        raise Nle.ExceptionExample
      1 ->
        String.to_integer("s")
      _ ->
        raise "running time error"
    end
  rescue
    e in ArgumentError -> e.message
    e in ExceptionExample -> e.message
    e in RuntimeError -> e.message
  end
end
```

注意rescue后面的Error类型要对应.

throws

exits

高级主题

这是第一章节 GitBook allows you to organize your book into chapters, each chapter is stored in a separate file like this one.

Genserver

genserver

引用与去引用(quote and unquote)

引用

elixir程序由三部分组成，例如求合函数sum(1,3,3)可以表示如下：

```
{:sum, [], [1, 2, 3]}
```

任何表达式都可以通过宏引用(quote macro)来表达:

```
iex(1)> quote do: sum(1, 2, 3)
{:sum, [], [1, 2, 3]}
```

第一个元素为函数名；第二个为keyword list，代表元数据；第三个元素为参数列表。许多语言把这种表达 式称为抽象语法树Abstract Syntax Tree(AST)，elixir称之为引用表达式。

下面是一个复杂一点的用户展开:

```
iex(2)> quote do: sum(1, 2 + 3, 4)
{:sum, [], [1, {:+, [context: Elixir, import: Kernel], [2, 3]}, 4]}
```

引用的通用表达

一般，上面的引用元组可以表达成以下结构：

```
{atom | tuple, list, list | atom}
```

其中:

1. 第一个元素为一个原子类型或者同样结构的元组;
2. 第二个元素为一个keyword list,它包含元数据、像numbers和contexts；
3. 第三个元素或者为一个参数列表，或者为原子类型。当它是一个原子类型时，意味着这个引用表达式表示 的是一个变量。

注意，有下面5种elixir字面量类型，它们直接返回自己，不翻译成用户表达式：

```
:sum          #=> Atoms
1.0           #=> Numbers
[1, 2]        #=> Lists
"strings"     #=> Strings
{key, value} #=> Tuples with two elements
```

大部分elixir代码是直接翻译成它对应的引用表达式。

引用转义(unquote)

有时候你想在一个引用中对某个变量进行转义，如下：

```
iex(5)> number = 13
13
iex(6)> Macro.to_string(quote do: 11 + number)
"11 + number"
```

这不是我们想要的，因为我们想把number变量换成它的值，那么可采用unquote实现：

```
iex(7)> number = 13
13
iex(8)> Macro.to_string(quote do: 11 + unquote(number))
"11 + 13"
```

宏

官方给的建议是宏是不到万不得已才用，因为它的可读性不好，并且写起来不是那么容易。

第一个宏

定义宏通过defmacro/2这个表达式来完成。官方给了一个unless的例子，如下：

```
defmodule Unless do
  def fun_unless(clause, expression) do
    if(!clause, do: expression)
  end

  defmacro macro_unless(clause, expression) do
    quote do
      if(!unquote(clause), do: unquote(expression))
    end
  end
end
```