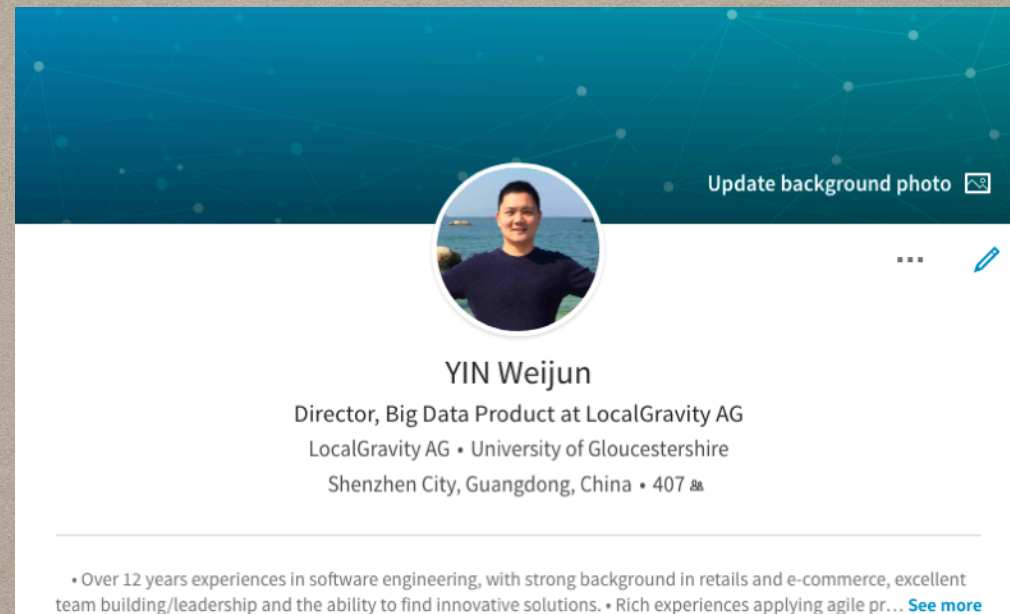



分布式原住民

Distribution Primitives

Weijun Yin

A digital representation of a LinkedIn profile card mounted on a textured, light brown cardstock. The profile card has a teal header with a network diagram. It features a circular profile picture of a man in a dark blue shirt. To the right of the photo is a button that says "Update background photo" with a camera icon. Below the photo, the name "YIN Weijun" is displayed in bold, followed by his title "Director, Big Data Product at LocalGravity AG" and his affiliations: "LocalGravity AG • University of Gloucestershire" and "Shenzhen City, Guangdong, China • 407 88". A horizontal line separates the header from the bio. The bio text reads: "• Over 12 years experiences in software engineering, with strong background in retails and e-commerce, excellent team building/leadership and the ability to find innovative solutions. • Rich experiences applying agile pr... See more".

Update background photo 

YIN Weijun
Director, Big Data Product at LocalGravity AG
LocalGravity AG • University of Gloucestershire
Shenzhen City, Guangdong, China • 407 88

• Over 12 years experiences in software engineering, with strong background in retails and e-commerce, excellent team building/leadership and the ability to find innovative solutions. • Rich experiences applying agile pr... [See more](#)

尹伟君

[linkedin.com/in/yin-weijun-93a08816/](https://www.linkedin.com/in/yin-weijun-93a08816/)

AGENDA

- Set up a Node cluster
- Send some messages to a process that lives in another node
- The concepts are often mentioned during clustering discussion
- As a node, to register an unique name so others can find you
- As a node, to ensure your message will be received by multiple processes

原住民**NODE**是什么？

- ✦ “BEAM-powered distributed systems are built by connecting multiple nodes into a cluster”
- ✦ “A Node is a BEAM instance that has a name associated with it.”
- ✦ Often we're called distributed Erlang/Elixir

source: “Elixir in Action, 2nd”, published by Manning

怎样将NODE团结起来？

- ✦ 先来一起读使用手册
`man iex`
- ✦ 常见nodes连接操作
(见右图)
- ✦ Hidden node
- ✦ monitor/2

```
$ iex --sname weij@capricorn
Erlang/OTP 22 [erts-10.4.1] [source] [64-bit] [smp:4:4] [ds:4:4]
Interactive Elixir (1.9.0) - press Ctrl+C to exit (type h() ENTER)
iex(weij@capricorn)1> Node.list()
[]
iex(weij@capricorn)2> Node.ping(:jane@localhost)
:pang
iex(weij@capricorn)3> Node.ping(:jane@capricorn)
:pong
iex(weij@capricorn)4> Node.list()
[:jane@capricorn]
iex(weij@capricorn)5> Node.get_cookie()
:CSFTLMJTFFQTPNLUROWB
iex(weij@capricorn)6> Node.list([:this, :connected])
[:weij@capricorn, :jane@capricorn, :chase@capricorn]
iex(weij@capricorn)7> Node.list([:this, :visible])
[:weij@capricorn, :jane@capricorn]
iex(weij@capricorn)8> Node.monitor(:chase@capricorn, :true)
true
iex(weij@capricorn)9> flush
{:nodedown, :chase@capricorn}
:ok
iex(weij@capricorn)10>
```

Node.spawn/2 anonymous functions

:global, :rpc, and :pg2 always ignore hidden nodes.

What will the use scenario for hidden node?

为什么要nodes connect起来成为一个cluster?

node居住在本地host, 和另外一个host有什么区别?

HIDDEN NODE

- 用于当不希望成为cluster一部分的时候，例如：
remote shell connection, metrics collection

NODE之间通信不得不聊的点

```
iex(weij@capricorn)11> Node.spawn(:jane@capricorn, fn -> IO.puts("hello from #{node()}") end)
hello from jane@capricorn
#PID<11406.143.0>
iex(weij@capricorn)12> caller = self()
#PID<0.114.0>
iex(weij@capricorn)13> Node.spawn(:jane@capricorn, fn -> send(caller, {:ok, "running in #{node()}"})) end)
#PID<11406.144.0>
iex(weij@capricorn)14> flush
{:ok, "running in jane@capricorn"}
:ok
iex(weij@capricorn)15> :erlang.term_to_binary({:ok, "running in jane@capricorn"})
<<131, 104, 2, 100, 0, 2, 111, 107, 109, 0, 0, 0, 25, 114, 117, 110, 110, 105,
  110, 103, 32, 105, 110, 32, 106, 97, 110, 101, 64, 99, 97, 112, 105, 99,
  111, 114, 110>>
iex(weij@capricorn)16> v(15) |> :erlang.binary_to_term()
{:ok, "running in jane@capricorn"}
iex(weij@capricorn)17>
```

- Group Leader
- Location Transparency
- Encoding/decoding
- Order Guarantee

Using Node.spawn/4 instead

GROUP LEADER

“By modeling IO devices with processes, the Erlang VM allows I/O messages to be routed between different nodes running Distributed Erlang or even exchange files to perform read/write operations across nodes.”

<https://elixir-lang.org/getting-started/io-and-the-file-system.html#processes-and-group-leaders>

Process.info to check out group leader info

The main task of the group leader is to collect the I/O output from all the processes in its group and pass that I/O to or from the underlying system. Basically the group leader owns stdin, stdout, and stderr on behalf of the group and handles passing information to and from channels like those.



```


iex(weij@capricorn)64> spawn(fn -> receive do: (msg -> IO.inspect(msg)) end)
#PID<0.2448.0>
iex(weij@capricorn)65> Process.info(v(64))
[
  current_function: {:prim_eval, :receive, 2},
  initial_call: {:erlang, :apply, 2},
  status: :waiting,
  message_queue_len: 0,
  links: [],
  dictionary: [],
  trap_exit: false,
  error_handler: :error_handler,
  priority: :normal,
  group_leader: #PID<0.70.0>,
  total_heap_size: 233,
  heap_size: 233,
  stack_size: 9,
  reductions: 4005,
  garbage_collection: [
    max_heap_size: %{error_logger: true, kill: true, size: 0},
    min_bin_vheap_size: 46422,
    min_heap_size: 233,
    fullsweep_after: 65535,
    minor_gcs: 0
  ],
  suspending: []
]
iex(weij@capricorn)66> self()
#PID<0.114.0>
iex(weij@capricorn)67> IO.write(v(64), "hello")
{:io_request, #PID<0.114.0>, #Reference<0.1285787874.634388481.208273>,
{:put_chars, :unicode, "hello"}}
** (ErlangError) Erlang error: :terminated
(stdlib) :io.put_chars(#PID<0.2448.0>, :unicode, "hello")
iex(weij@capricorn)67>

```

source: <https://elixir-lang.org/getting-started/io-and-the-file-system.html#the-io-module>

term_to_binary/1 and binary_to_term/1

 joeerl Creator of Erlang - Fondly Remembered

 NobbZ Dec '17

I thought so.

When I teach I go on and on and on about the greatness of `term_to_binary` and the inverse. These are *incredibly* useful.

They are also blindingly fast compared to any JSON/XML type serialisation.

Something like:

```
defmodule Term do
  def store(anything, path) do
    bin = :erlang.term_to_binary(anything)
    File.write!(path, bin)
  end

  def fetch(path) do
    File.read!(path) |> :erlang.binary_to_term
  end
end
```

What makes a web application fast?

You have to remember that any SQL database will be a bottleneck since there is an impedance mismatch between the way data is represented in a database and the way it is represented in the beam VM.

Databases are basically rectangular tables of cells, where the cells contain very simple types like strings and integers - every time you access a row of an external database this list of cells has to be converted to beam internal data structures - this conversion is extremely expensive.

The best way to persist data is in a process - then no conversion is needed, but this is not fault tolerant - so you need to keep a trail of updates to the data and store this on disk.

Often you don't need a database for example you might like to have a system where you store all the user data as in the file system with "one file per user" this will scale very nicely - just move the files to a new machine if you need more capacity.

Erlang has two primitives `term_to_binary` and the inverse `binary_to_term` that serialise any term and reconstruct it - so storing complex terms on disk is really easy.

I have mixed feelings about databases, they are great for aggregate operations (for example, find all users that have these attributes) but terrible for operations on individual users (where a single file per user is far better).

If I were designing a new system I'd go for 'one file per user' as much as possible and try to limit databases for operations over all users.

source: <https://elixirforum.com/t/acceptance-of-erlangs-term-to-binary-and-vice-versa-in-elixir/11034/>

ORDER GUARANTEE

- **At least once** while delivering message between process
- Message ordering between two same processes is ensured (A -> B)
- Ordering between different process pairs is **NOT** ensured (A -> C, B -> C)

原住民**PROCESS DISCOVERY**

- ✦ Process.register/2
- ✦ Registry module
- ✦ :global
- ✦ :gp2

原住民GLOBAL

- :global apis demo
- Stored in replica global name tables on every node, so reading the names fast but registering may be slow
- Register while startup
GenServer, start_link(__MODULE__, args, { :global, :any_term })
- Favorite availability over consistency

source: <http://erlang.org/doc/man/global.html>

```
:global.register_name({:meetup, "weijun"}, self())
```

```
:global.registered_names
```

```
:global.whereis_name(:meetup, "weijun")
```

```
:global.un_register_name({:meetup, "weijun"})
```

原住民PG2能帮忙我们解决问题？

- :pg2 usage
- 注册多个Processes共用一个global名字
- 允许一条信息同时发送给1个，或者多个Process
- 当联系不上member process时候，会自动将其从列表中剔除

source: <http://erlang.org/doc/man/pg2.html>

```
:pg2.start()
:pg2.create({:meetup, :elixir})
:pg2.join({:meetup, :elixir}, self())
:pg2.which_groups()
:pg2.get_members({:meetup, :elixir})
:pg2.get_closeted_pid({:meetup, :elixir})
```


Phoenix.PubSub.PG2

```
52 | @impl true
    | @spec init({any, any}) :: {:ok, any}
53 | def init({name, adapter_name}) do José Valim, a month ago • Port PubSub to Registry
    |   pg2_group = pg2_namespace(adapter_name)
54 |   :ok = :pg2.create(pg2_group)
55 |   :ok = :pg2.join(pg2_group, self())
56 |   {:ok, name}
57 | end
58 |
59 |
60 | def handle_info({:forward_to_local, topic, message, dispatcher}, pubsub) do
61 |   Phoenix.PubSub.local_broadcast(pubsub, topic, message, dispatcher)
62 |   {:noreply, pubsub}
```

```
16 | @impl true
    | @spec broadcast(any, any, any, any) :: :ok | {:error, :no_such_group}
17 | def broadcast(adapter_name, topic, message, dispatcher) do José Valim, a month ago
    |   case :pg2.get_members(pg2_namespace(adapter_name)) do
18 |     {:error, {:no_such_group, _}} ->
19 |       {:error, :no_such_group}
20 |   end
21 |
22 |   pids ->
23 |     message = forward_to_local(topic, message, dispatcher)
24 |
25 |     for pid <- pids, node(pid) != node() do
26 |       send(pid, message)
27 |     end
28 |
29 |     :ok
30 |   end
31 | end
```

source: https://github.com/phoenixframework/phoenix_pubsub/blob/master/lib/phoenix/pubsub/pg2.ex

FURTHER READING

ElixirConf 2016 - pg2 and You: Getting Distributed with Elixir by Eric Entin

https://www.youtube.com/watch?v=_O-bLuVhcCA

Pg2 Basics — <https://stephenbussey.com/2018/02/17/pg2-basics-use-process-groups-for-orchestration-across-a-cluster.html>

:rpc service — <http://erlang.org/doc/man/rpc.html>

:net_kernel — http://erlang.org/doc/man/net_kernel.html

:net_adm — http://erlang.org/doc/man/net_adm.html

Book — Elixir in Action, 2nd, published by Manning