Elixirの便利なところって何？

ヽ(´・肉・｀)ノ

はじめに

- スライド http://slide.rabbit-shocker.org/authors/niku/osc2016do-elixir/
- ファイル https://gist.github.com/niku/076cf8796c0a5e51ff2103c81ff49db9
- タグ #osc16do #サッポロビーム

だれですか

- にくです
- サッポロビームによくいます
  - Erlang とか Elixir とか**そうじゃないもの**で遊ぶ集まりです
  - なにしてもよい
- 最近は Ruby や TypeScript を書いて仕事してます

なんでサッポロビームなの

- 今広く利用されているErlangVMの実装がBEAMという名前
- サッポロビールと似ていておもしろいかなと思った

Elixirをやると，元になったErlangやVMへの敬意を感じたので，双方に共通するVM名にした

このセミナーでみなさんに提供したいこと

- 雰囲気
- やっていけそうな気持ち
  - 勘違いでもやってみると何か得られる(こともある)(かもしれない)(個人の感想です)

詳しくはwebや本で

今日やってみること

- ElixirやErlangVMのおはなし
- Stackを普通に動かす/監視付きで動かす
- EchoServerを作る

おわび

- タイトルに誤りがありました
- Elixirの **すごく** 便利なところって何？
- そもそも(他の言語と同じく)汎用言語なので **大体何にでも使えます**

Elixirの すごく 便利なところ

- ErlangVMを使わない他の言語と比べて
  - **Erlang/OTP**と**ErlangVM**が使えるぞ！

Elixirの すごく 便利なところ

- ErlangVMを使う言語と比べて(主にErlang)
  - モジュールや関数の分け方や名前周辺のツール(プロジェクト管理やREPL)などが**(僕には)想像しやすい/馴染みやすい**
  - **健全なマクロ**がある(不健全にもできる)の**メタプログラミング**がやりやすい

ErlangVMとErlang/OTPってどういいの

- 高負荷でも**安定**（らしい）
- 予想外のことが起きたときの**復旧**に強い
  - **処理**と**処理の監視**をひとまとめに書けて，**ひとまとめに動かせる**

ErlangVMとErlang/OTPについて詳しく

- 詳しい方（Voluntasさん）が2016-06-24に東京にてBPStudyで発表する
- 資料が公開されている
  - Erlang/OTP (仮)
  - https://gist.github.com/voluntas/b94b8e090e34da13f1e36bff13ab6320

サーバーとクライアント

プログラムをクライアントとサーバーに分類する

	クライアント	サーバー
仕事	依頼する	依頼待つ
終了	任意	終わること少ない
寿命	短い	長い

ErlangVMとErlang/OTPのいいところ

- 安定
- 復旧に強い

サーバープログラミングに向いている

復旧に強いとは

- Supervisorが存在する
  - [名詞] 監督者, 管理者.
- **働く**プログラムと**監視する**プログラムを作れる

Stack

```elixir
# 00-stack.exs
defmodule Stack do
  use GenServer

  def handle_call(:pop, _from, [h|t]) do
    {:reply, h, t}
  end

  def handle_cast({:push, item}, state) do
    {:noreply, [item|state]}
  end
end
```

Stack試す

```
$ iex -r 00-stack.exs
iex(1)> {:ok, pid} = GenServer.start_link(Stack, [:hello])
iex(2)> GenServer.call(pid, :pop)
:hello
iex(3)> GenServer.cast(pid, {:push, :world})
:ok
iex(4)> GenServer.call(pid, :pop)
:world
iex(5)> GenServer.call(pid, :pop)
** (EXIT from #PID<0.66.0>) an exception was raised:
(略)
iex(1)> GenServer.call(pid, :pop)
** (CompileError) iex:2: undefined function pid/0
    (stdlib) lists.erl:1353: :lists.mapfoldl/3
```

(参考)

コードはGenServerのドキュメントから持ってきた
http://elixir-lang.org/docs/v1.2/elixir/GenServer.html

普通のプログラミング

- エラーを扱わないと終わってしまう
- **働く** プログラムが気をつける
- `try-catch` / `begin-end`

ErlangVMでのプログラミング

- **働く** プログラム
- それを **監視する** プログラム

監視する

```elixir
# 01-stack.exs
defmodule Stack do
  use GenServer

  def start_link(state, opts \\ []) do
    GenServer.start_link(__MODULE__, state, opts)
  end

  def handle_call(:pop, _from, [h|t]) do
    {:reply, h, t}
  end

  def handle_cast({:push, item}, state) do
    {:noreply, [item|state]}
  end
end
```

監視を試す

```
% iex -r 01-stack.exs
iex(1)> import Supervisor.Spec
iex(2)> children = [
...(2)>   worker(Stack, [[:hello], [name: :sup_stack]])
...(2)> ]
iex(3)> {:ok, pid} = Supervisor.start_link(children, strategy: :one_for_one)
iex(4)> GenServer.call(:sup_stack, :pop)
:hello
iex(5)> GenServer.cast(:sup_stack, {:push, :world})
:ok
iex(6)> GenServer.call(:sup_stack, :pop)
:world
iex(7)> GenServer.call(:sup_stack, :pop)
** (exit) exited in: GenServer.call(:sup_stack, :pop, 5000)
(略)
iex(7)> GenServer.call(:sup_stack, :pop)
:hello
```

(参考)

コードはSupervisorのドキュメントから持ってきた
http://elixir-lang.org/docs/v1.2/elixir/Supervisor.html

エラー後を比較

```
# 監視なし
iex(1)> GenServer.call(pid, :pop)
** (CompileError) iex:2: undefined function pid/0
    (stdlib) lists.erl:1353: :lists.mapfoldl/3

# 監視あり
iex(7)> GenServer.call(:sup_stack, :pop)
:hello
```

監視プログラムのお仕事

- **働く**プログラムの**エラーを検知**
- **初期値**で**復活**させる

Echoサーバー

- RFC862(Echo Protocol) https://tools.ietf.org/html/rfc862

Network Working Group
Request for Comments: 862

J. Postel
ISI
May 1983

Echo Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement an Echo Protocol are expected to adopt and implement this standard.

A very useful debugging and measurement tool is an echo service. An echo service simply sends back to the originating source any data it receives.

TCP Based Echo Service

One echo service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 7. Once a connection is established any data received is sent back. This continues until the calling user terminates the connection.

UDP Based Echo Service

Another echo service is defined as a datagram based application on

Echo Protocol

> A very useful debugging and measurement tool is an echo service. An
> echo service simply sends back to the originating source any data it
> receives.

> デバッグや計測のためにとても便利なツールとしてエコーサービスがある.
> エコーサービスは，受け取ったどんなデータも，単に送信元へと送り返す.

Echo Protocolの種類

- **TCP Based Echo Service**
- UDP Based Echo Service

TCP Based Echo Service

> One echo service is defined as a connection based application on TCP.
> A server listens for TCP connections on TCP port 7. Once a
> connection is established any data received is sent back. This
> continues until the calling user terminates the connection.

> 一つめのエコーサービスは，TCP上で接続をベースにしたアプリケーションとして定義されている．
> サーバーはTCP7番ポートでTCP接続を待ち受け(listen)する．
> 接続が確立(established)されたら，どんなデータでも，受け取ったものを送り返す．
> これを呼び出した側のユーザーが接続を切断(terminate)するまで続ける．

EchoServer実装

1. サーバーはTCP7番ポートで
   TCP接続を待ち受け(listen)する
   (今回はrootでなくても使える
   29297番ポートにする)
2. 接続を確立(established)する
3. 受け取ったデータを送り返す
4. 呼び出した側のユーザーが切断
   するまで 3. を続ける

準備

```
% cat 02-echo.exs
# EMPTY FILE
% elixir -r 02-echo.exs --no-halt
----
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
telnet: Unable to connect to remote host
```

準備

- つなげない

待ちうける - 実装

```
# 03-echo.exs
defmodule EchoServer do
  def accept(port) do
    :gen_tcp.listen(port,
      [:binary, packet: :line, active: :false, reuseaddr: true])
    IO.puts "Accepting connections on port #{port}"
  end
end

EchoServer.accept(29297)
:timer.sleep(:infinity)
```

待ちうける - 試す

```
% elixir -r 03-echo.exs
Accepting connections on port 29297
----
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello~
```

待ちうける

- つなげた
- hello~ をサーバーが読んでいない

データを読む - 実装

```
% diff 03-echo.exs 04-echo.exs
3c3
<     :gen_tcp.listen(port,
---
>     {:ok, socket} = :gen_tcp.listen(port,
5a6,8
>     {:ok, client} = :gen_tcp.accept(socket)
>     {:ok, data} = :gen_tcp.recv(client, 0)
>     IO.puts "Reading data: #{data}"
```

データを読む - 実装

```
# 04-echo.exs
defmodule EchoServer do
  def accept(port) do
    {:ok, socket} = :gen_tcp.listen(port,
      [:binary, packet: :line, active: :false, reuseaddr: true])
    IO.puts "Accepting connections on port #{port}"
    {:ok, client} = :gen_tcp.accept(socket)
    {:ok, data} = :gen_tcp.recv(client, 0)
    IO.puts "Reading data: #{data}"
  end
end

EchoServer.accept(29297)
:timer.sleep(:infinity)
```

データを読む - 試す

```
% elixir -r 04-echo.exs
Accepting connections on port 29297
Reading data: hello~
----
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello~
```

データを読む

- hello~をサーバーが読んだ
- クライアントに返していない

データを返す - 実装

```
% diff 04-echo.exs 05-echo.exs
8a9
>       :gen_tcp.send(client, data)
```

データを返す - 実装

```
# 05-echo.exs
defmodule EchoServer do
  def accept(port) do
    {:ok, socket} = :gen_tcp.listen(port,
      [:binary, packet: :line, active: :false, reuseaddr: true])
    IO.puts "Accepting connections on port #{port}"
    {:ok, client} = :gen_tcp.accept(socket)
    {:ok, data} = :gen_tcp.recv(client, 0)
    IO.puts "Reading data: #{data}"
    :gen_tcp.send(client, data)
  end
end

EchoServer.accept(29297)
:timer.sleep(:infinity)
```

データを返す - 試す

```
% elixir -r 05-echo.exs
Accepting connections on port 29297
Reading data: hello~
----
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello~
hello~
hello again~
```

データを返す

- クライアントに返事がきた
- 繰り返せない

繰り返す - 実装

```
% diff 05-echo.exs 06-echo.exs
7,9c7,13
<     {:ok, data} = :gen_tcp.recv(client, 0)
<     IO.puts "Reading data: #{data}"
<     :gen_tcp.send(client, data)
---
>     serve(client)
>   end
>
>   defp serve(socket) do
>     {:ok, data} = :gen_tcp.recv(socket, 0)
>     :gen_tcp.send(socket, data)
>     serve(socket)
14d17
< :timer.sleep(:infinity)
```

繰り返す - 実装

```
# 06-echo.exs
defmodule EchoServer do
  def accept(port) do
    {:ok, socket} = :gen_tcp.listen(port,
      [:binary, packet: :line, active: :false, reuseaddr: true])
    IO.puts "Accepting connections on port #{port}"
    {:ok, client} = :gen_tcp.accept(socket)
    serve(client)
  end

  defp serve(socket) do
    {:ok, data} = :gen_tcp.recv(socket, 0)
    :gen_tcp.send(socket, data)
    serve(socket)
  end
end

EchoServer.accept(29297)
```

繰り返す - 試す

```
% elixir -r 06-echo.exs
Accepting connections on port 29297
** (MatchError) no match of right hand side value: {:error, :closed}
    06-echo.exs:11: EchoServer.serve/1
    (elixir) lib/code.ex:363: Code.require_file/2
    (elixir) lib/enum.ex:1088: Enum."-map/2-lists^map/1-0-"/2
----
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello~
hello~
hello again~
hello again~
^]
telnet> quit
Connection closed.
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
telnet: Unable to connect to remote host
```

繰り返す

- 繰り返せた
- 切ると再接続できない

再接続する - 実装

```
% diff 06-echo.exs 07-echo.exs
17c17,23
< EchoServer.accept(29297)
---
> import Supervisor.Spec
> children = [
>   worker(Task, [EchoServer, :accept, [29297]])
> ]
> opts = [strategy: :one_for_one, name: EchoServer.Supervisor]
> Supervisor.start_link(children, opts)
> :timer.sleep(:infinity)
```

再接続する - 実装

```elixir
# 07-echo.exs
defmodule EchoServer do
  def accept(port) do
    {:ok, socket} = :gen_tcp.listen(port,
      [:binary, packet: :line, active: :false, reuseaddr: true])
    IO.puts "Accepting connections on port #{port}"
    {:ok, client} = :gen_tcp.accept(socket)
    serve(client)
  end

  defp serve(socket) do
    {:ok, data} = :gen_tcp.recv(socket, 0)
    :gen_tcp.send(socket, data)
    serve(socket)
  end
end
```

再接続する - 実装(つづき)

```
# 07-echo.exs(つづき)
import Supervisor.Spec
children = [
  worker(Task, [EchoServer, :accept, [29297]])
]
opts = [strategy: :one_for_one, name: EchoServer.Supervisor]
Supervisor.start_link(children, opts)
:timer.sleep(:infinity)
```

再接続する - 試す

```
% elixir -r 07-echo.exs
Accepting connections on port 29297
Accepting connections on port 29297

20:11:30.728 [error] Task #PID<0.54.0> started from EchoServer.Supervisor terminating
** (MatchError) no match of right hand side value: {:error, :closed}
    07-echo.exs:11: EchoServer.serve/1
    (elixir) lib/task/supervised.ex:89: Task.Supervised.do_apply/2
    (stdlib) proc_lib.erl:240: :proc_lib.init_p_do_apply/3
Function: &EchoServer.accept/1
    Args: [29297]
```

再接続する - 試す(つづき)

```
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello~
hello~
hello again~
hello again~
^]
telnet> quit
Connection closed.
/Users/d-kitamura% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello once again~
hello once again~
----
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hi!
```

再接続する

- 再接続できた
- 複数接続できない

複数接続する - 実装

```
% diff 07-echo.exs 08-echo.exs
5a6,9
>     loop_acceptor(socket)
>   end
>
>   defp loop_acceptor(socket) do
7c11,12
<     serve(client)
---
>     Task.start_link(fn -> serve(client) end)
>     loop_acceptor(socket)
```

複数接続する - 実装

```
# 08-echo.exs
defmodule EchoServer do
  def accept(port) do
    {:ok, socket} = :gen_tcp.listen(port,
      [:binary, packet: :line, active: :false, reuseaddr: true])
    IO.puts "Accepting connections on port #{port}"
    loop_acceptor(socket)
  end

  defp loop_acceptor(socket) do
    {:ok, client} = :gen_tcp.accept(socket)
    Task.start_link(fn -> serve(client) end)
    loop_acceptor(socket)
  end

  defp serve(socket) do
    {:ok, data} = :gen_tcp.recv(socket, 0)
    :gen_tcp.send(socket, data)
    serve(socket)
  end
end
```

複数接続する - 実装(つづき)

```
# 08-echo.exs(つづき)
import Supervisor.Spec
children = [
  worker(Task, [EchoServer, :accept, [29297]])
]
opts = [strategy: :one_for_one, name: EchoServer.Supervisor]
Supervisor.start_link(children, opts)
:timer.sleep(:infinity)
```

複数接続する - 試す

```
% elixir -r 08-echo.exs
Accepting connections on port 29297
Accepting connections on port 29297

20:35:09.005 [error] Task #PID<0.55.0> started from #PID<0.54.0> terminating
** (MatchError) no match of right hand side value: {:error, :closed}
    08-echo.exs:16: EchoServer.serve/1
    (elixir) lib/task/supervised.ex:89: Task.Supervised.do_apply/2
    (stdlib) proc_lib.erl:240: :proc_lib.init_p_do_apply/3
Function: #Function<0.97146601/0 in EchoServer.loop_acceptor/1>
    Args: []
Accepting connections on port 29297

20:35:43.462 [error] Task #PID<0.58.0> started from #PID<0.56.0> terminating
** (MatchError) no match of right hand side value: {:error, :closed}
    08-echo.exs:16: EchoServer.serve/1
    (elixir) lib/task/supervised.ex:89: Task.Supervised.do_apply/2
    (stdlib) proc_lib.erl:240: :proc_lib.init_p_do_apply/3
Function: #Function<0.97146601/0 in EchoServer.loop_acceptor/1>
    Args: []
```

複数接続する - 試す(つづき)

```
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello~
hello~
hello again~
hello again~
^]
telnet> quit
Connection closed.
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello once again~
hello once again~
Connection closed by foreign host.
%
----
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hi!
hi!
^]
telnet> quit
Connection closed.
```

複数接続する

- 複数接続できた
- 影響を分離できない(あるクライアントでエラーになると，全てのクライアントに影響がある)

影響を分離する - 実装

```
% diff 08-echo.exs 09-echo.exs
11c11,12
<     Task.start_link(fn -> serve(client) end)
---
>     {:ok, pid} = Task.Supervisor.start_child(EchoServer.TaskSupervisor, fn -> serve(client) end)
>     :ok = :gen_tcp.controlling_process(client, pid)
23a25
>   supervisor(Task.Supervisor, [[name: EchoServer.TaskSupervisor]]),
```

影響を分離する - 実装

```elixir
# 09-echo.exs
defmodule EchoServer do
  def accept(port) do
    {:ok, socket} = :gen_tcp.listen(port,
      [:binary, packet: :line, active: :false, reuseaddr: true])
    IO.puts "Accepting connections on port #{port}"
    loop_acceptor(socket)
  end

  defp loop_acceptor(socket) do
    {:ok, client} = :gen_tcp.accept(socket)
    {:ok, pid} = Task.Supervisor.start_child(EchoServer.TaskSupervisor, fn -> serve(client) end)
    :ok = :gen_tcp.controlling_process(client, pid)
    loop_acceptor(socket)
  end

  defp serve(socket) do
    {:ok, data} = :gen_tcp.recv(socket, 0)
    :gen_tcp.send(socket, data)
    serve(socket)
  end
end
```

影響を分離する - 実装(つづき)

```
# 09-echo.exs(つづき)
import Supervisor.Spec
children = [
  supervisor(Task.Supervisor, [[name: EchoServer.TaskSupervisor]]),
  worker(Task, [EchoServer, :accept, [29297]])
]
opts = [strategy: :one_for_one, name: EchoServer.Supervisor]
Supervisor.start_link(children, opts)
:timer.sleep(:infinity)
```

影響を分離する - 試す

```
% elixir -r 09-echo.exs
Accepting connections on port 29297

20:58:48.775 [error] Task #PID<0.56.0> started from #PID<0.55.0> terminating
** (MatchError) no match of right hand side value: {:error, :closed}
    09-echo.exs:17: EchoServer.serve/1
    (elixir) lib/task/supervised.ex:89: Task.Supervised.do_apply/2
    (stdlib) proc_lib.erl:240: :proc_lib.init_p_do_apply/3
Function: #Function<0.126567687/0 in EchoServer.loop_acceptor/1>
    Args: []

20:59:14.307 [error] Task #PID<0.58.0> started from #PID<0.55.0> terminating
** (MatchError) no match of right hand side value: {:error, :closed}
    09-echo.exs:17: EchoServer.serve/1
    (elixir) lib/task/supervised.ex:89: Task.Supervised.do_apply/2
    (stdlib) proc_lib.erl:240: :proc_lib.init_p_do_apply/3
Function: #Function<0.126567687/0 in EchoServer.loop_acceptor/1>
    Args: []
```

影響を分離する - 試す(つづき)

```
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello~
hello~
hello again~
hello again~
^]
telnet> quit
Connection closed.
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hello once again~
hello once again~
is connected?
is connected?
----
% telnet 127.0.0.1 29297
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hi!
hi!
^]
telnet> quit
Connection closed.
```

影響を分離する

- 影響を分離できた(あるクライアントでエラーになっても，他のクライアントに影響がない)
- **エコーサーバー完成**

エコーサーバー完成形

```
defmodule EchoServer do
  def accept(port) do
    {:ok, socket} = :gen_tcp.listen(port,
      [:binary, packet: :line, active: :false, reuseaddr: true])
    IO.puts "Accepting connections on port #{port}"
    loop_acceptor(socket)
  end

  defp loop_acceptor(socket) do
    {:ok, client} = :gen_tcp.accept(socket)
    {:ok, pid} = Task.Supervisor.start_child(EchoServer.TaskSupervisor, fn -> serve(client) end)
    :ok = :gen_tcp.controlling_process(client, pid)
    loop_acceptor(socket)
  end

  defp serve(socket) do
    {:ok, data} = :gen_tcp.recv(socket, 0)
    :gen_tcp.send(socket, data)
    serve(socket)
  end
end

import Supervisor.Spec
children = [
  supervisor(Task.Supervisor, [[name: EchoServer.TaskSupervisor]]),
  worker(Task, [EchoServer, :accept, [29297]])
]
opts = [strategy: :one_for_one, name: EchoServer.Supervisor]
Supervisor.start_link(children, opts)
:timer.sleep(:infinity)
```

(参考)

コードはGetting StartedのTask and gen_tcpから持ってきた
http://elixir-lang.org/getting-started/mix-otp/task-and-gen-tcp.html

今日やってみたこと

- ElixirやErlangVMのおはなし
- Stackを普通に動かす/監視付きで動かす
- EchoServerを作る

どうやって勉強したらいい？

- https://elixirschool.com/jp/ 手始めに全体像見るのにいいかも
- http://elixir-lang.org/getting-started/introduction.html 全部やったらそれなりにわかる

どうやって勉強したらいい？

- https://pragprog.com/book/elixir12/programming-elixir-1-2 良書（英語）
- 日本語翻訳版が夏に出るみたい https://twitter.com/ohrdev/status/736449636534079488
- サッポロビームに参加してみる

質問？

- どこで役にたちそう？
- メタプログラミングできるの？
- ライブラリは？
- 依存性解決ツールとか