Elixir Lesson 2018/6/26

アジェンダ

- Elixirとは?
- Elixirの実行環境構築を作ろう
- IEx helperを使って、Hello Worldを実行しよう
- mixビルドツールを使ってみる
- 簡単なcliアプリケーションを作ろう
- こんなときはどうする(エラーが発生した場合)

Elixirとは(TODO: 日本語でOK)

`Elixir` とは

- `スケーラブル` かつ `メンテナンス性の高い` アプリケーションを構築する為に考案された `関数型プログラミング言語`
- Erlangで実装されていることから `Erlang が持つ特徴も備えています` ので

  レイテンシーが低い分散システム、耐障害性能の高いシステムを構築することが容易です。
- Webアプリケーション開発や組込みソフトウェアの分野を中心に利用されています。

特徴

- 関数型言語
  - コーディングスタイル
    - 素早く簡潔に、メンテナンスしやすいコードを書きやすい
  - パターンマッチング

        - destructoreソースコード上での、構造を持つデータの分解とアクセスを簡単に行うことができる
        - assertion
        - 意図した条件(パターン)でのみ実行するように制約をつけたり、条件を明示的に表明することができる

- スケーラビリティ

    - Elixirは個別のプロセス上で実行されるプログラム
        - プロセスはそれぞれ独立(isolated)しており、Elixir言語が標準提供する機能で、相互にメッセージの交換もできます
        - プロセスが独立していることで得られるメリットとしては
        - 垂直スケーリング(vertical scaling)
            - 1台のマシン上で、例えば数十万のプロセスを同時平行実行させることができます
            - プロセスのリソースはそれぞれ単独で管理・解放され
            - 1つ1つのマシンのリソースを十分に活用できます
            - システム全体が停止する障害を減少する効果が期待できます。
        - 水平スケーリング(horizontal scaling)
            - 同一ネットワーク上であれば異なるマシン上でそれぞれプロセスを実行させることができます。
            - 分散システムの基盤となる機能を提供します
            - 複数のマシンで実行するように設計しておくことで、マシンの数に応じたスケーリングも可能です

- 耐障害性

    - Supervisor機能
      本番稼働しているソフトウェアではあるため、

      ネットワークの障害、入出力のエラー、サードパーティリソース問題等が起こりうるため
      障害の発生を完璧に回避することは不可能です。

      Elixirは、そのためのSupervisor機能を標準機能として提供します

      Supervisor機能とは、プロセス監視・プロセス管理する機能です。

      Elixirは、Supervisor機能でプロセスの再始動を行い、初期状態(プロセスが正常動作する状態)に復元します。

環境構築

- Atomエディタインストール
- git clientのインストール
- パッケージ管理ソフト(homebrew)インストール
- elixirパッケージ インストール
    - Terminalを開く
    - Terminal上で以下のコマンドを実行する

```
$ brew update
$ brew install elixir
$ erlang --version
$ elixir --version
$ mix --version
$ iex --version
$ mix local.hex
```

iex helperを使って、Hello Worldを実行しよう

- 概要
    - Elixirのプログラムを書いてすぐ実行(REPL)ができる。
- 目的
    - iexの使い方を学習する
- 使ってみる

```
$ iex # Terminal上でiexと実行
```

```
iex(1)> message = "Hello World" # message変数を定義する
"Hello World"

iex(2)> message |> IO.puts # message変数をIO.putsという関数に渡して画面に出力する
Hello World
:ok
```

- 終了したい場合

```
Ctrl+Cを押して更にaを押すと終了」
```

mixビルドツールを使ってみる

- 概要

```
Elixirに標準で付属する
アプリケーションのコンパイル・実行、使用するライブラリの管理、テストなどのタスクを定義・管理するため
```

- 目的
    - mixの使い方を学ぶ
    - シンプルなcliアプリケーションを作りmixをつかってのアプリケーション実行方法を学習する
        - 簡単なElixirプログラムの書き方
    - 参考
        - ex_cli(サードパーティライブラリ)

- `インストール`

```
$ git clone https://github.com/Eigo-Mt-Fuji/elixir-ex-cli-sample.git
$ cd elixir-ex-cli-sample
```

- `使ってみる`

```
$ mix deps.get # パッケージのインストール
$ mix compile # コンパイル
$ mix run sample_cli.ex -vv hello 中尾さん --from 藤川 # 実行
```

簡単なcliアプリケーションを作ろう

- 目的
    - シンプルなcliアプリケーションを作り、簡単なElixirプログラムの書き方を学ぶ
    - 参考: ex_cli(サードパーティライブラリ)

- `MyApp.SampleCLI` というモジュールを定義

```
defmodule MyApp.SampleCLI do
```

- `2行目` は `ExCLI.DSL` をインポートして使用する

```
  use ExCLI.DSL
```

...

- `23行目付近`

```
if context.verbose > 0 do
        IO.puts("Running hello command")
      end
      if from = context[:from] do
        IO.write("#{from} says: ")
      end
      IO.puts("Hello #{context.name}!")
    end
  end
end
```

こんなときはどうする(エラーが発生した場合)

- `usr/local/lib is not writable` のエラーが発生した場合

```
# /usr/local/libディレクトリがない場合は作成
$ ls -lad /usr/local/lib
$ mkdir -p /usr/local/lib
# 所有者が実行ユーザと異なる場合所有者を変更
$ sudo chown -R $USER:admin /usr/local/lib
$ brew link erlang
$ brew link elixir
$ ls -la /usr/local/lib/
```