Erlang: getting started

Johan Montelius

October 2, 2016

Introduction

This is an introduction to Erlang, or rather its a description of some tasks that you should complete in order to be up and running in Erlang.

1 Getting started

If you run on your own computer you need to install the Erlang development environment. If you're running a one of the KTH computers this is probably already done.

The first thing you need to do is download the Erlang SDK. This is found at www.erlang.org and is available both as a Windows binary and as source that you can make with the usual tools available on a Unix or even MacOs platform. If your using for example Ubuntu then the Erlang system is available in the repository.

The SDK includes a compiler, all the programming libraries and virtual machine. You can also download the Erlang manual in HTML. It does not include a development environment.

1.1 Erlang

When you have installed Erlang you should be able to start an Erlang shell. You do this either by starting Erlang found in the regular Windows program listing or by hand from a regular shell. Since you later will need to set runtime parameters you need to learn how to start Erlang by hand. Open a regular shell and type:

```
erl
```

This should start the Erlang shell and you should see something like this:

```
Erlang (BEAM) emulator version 5.6.1 [source] [smp:4] [async-threads:0] [hipe] [kern

Eshell V5.6.1  (abort with ^G)
1>
```

Type `help().` (note the dot) followed by return to see all the shell commands and `halt().` to exit the shell.

1.2 Emacs

As an development environment you need a text editor and what is better than Emacs. Download it from www.gnu.org/software/emacs/, available both for Unix, Mac and Windows. Ubuntu users will find it in the repository.

You need to add the code below to your `.emacs` file (provided that you have Erlang installed under `C:/Program Files/`). This will make sure that the Erlang mode is loaded as soon as you open a `.erl` file and that you can start an Erlang shell under Emacs etc. Change the `<Ver>` and `<ToolsVer>` to what is right in your system. On Linux it will look similar but the install directory is something like `/usr/local/lib/erlang/`.

Note! If you cut and past this text the '-character will not be a '-character, if you see what I mean. When you load the file in emacs you will have errors. If you do cut and paste then write in by hand "'erlang-start". This is true for all cut and paste from pdf documents, things might not be what they appear to be so be careful.

```
(setq load-path
  (cons
    "C:/Program Files/erl<Ver>/lib/tools-<ToolsVer>/emacs"
    load-path))
(setq erlang-root-dir
    "C:/Program Files/erl<Ver>")
(setq exec-path
  (cons
     "C:/Program Files/erl<Ver>/bin"
     exec-path))
(require 'erlang-start)
```

You will have to find your emacs home directory where the file should be placed. If you're on the KTH student computers then the home area is "h:" but you then need to set the "HOME" environment variable so that emacs finds its way.

If everything works you should be able to start an Erlang shell inside Emacs by `M-x run-erlang` (`M-x` is < *escape* > followed by x). A shell inside Emacs will allow you to quickly compile and run programs but when we experiment with distributed applications you need to run these in separate shells so make sure that you also know how start an Erlang shell manually.

1.3 Eclipse

If you prefer to use Eclipse you can install a Erlang plugin and do your development inside Eclipse. Fell free to use whatever environment you want.

2 Hello World

Open a file `hello.erl` and write the following:

```
-module(hello).

-export([world/0]).

world()->
  "Hello world!".
```

Now open a Erlang shell, compile and load the file with the command `c(hello).` and, call the function `hello:world()`.. Remember to end commands in the shell with a dot. If things work out you have successfully written, compiled and executed your first Erlang program.

Find the Erlang documentation and read the "Getting started" section.

3 Functional Programming

Here are some tasks that you can try to solve. They all use only the functional subset of Erlang.

3.1 Search trough a list

Given a list of atoms, for example

$$gurka, tomat, salad, potatis$$

, you should be able to determine if a particular atom is in the list. Implement the function `member/2` that takes a list a and an atom and returns `true` if the atom is in the list or `false` if the atom is not in the list.

Implement the function `count/2` that takes a list and an atom and returns a value that describes how many times the atom was found in the list.

3.2 Reverse a list

Implement a function that takes a list and returns a list with the elements in reversed order. Let's do it the simple way first, write a function that appends two list.

```
append(A, B) ->
   case A of
       [] ->
          ...;
```

```
        [H|Rest] ->
           [H | ...]
    end.
```

Then use the `append/2` function and write the function that reverses a list.

```
reverse(List) ->
     case List of
       [] ->
          ... ;
       [H|Rest] ->
          append(reverse(Rest), ...)
      end.
```

What is the complexity of this function? How many operations are performed when you reverse a list of n elements? Why is this function often referred to as *naive reverse*?

Write a new version of reverse but now with two arguments: the rest of the list to reverse and what we have seen so far in reverse order.

```
reverse(List, Sofar) ->
    case List of
        []  ->
            ...;
        [H|Rest] ->
            reverse(Rest, ...)
    end.
```

What is the complexity of this function?