LaTeX test doc for SphinxConJP 2014 Documentation

Release 0.1

Kenji Rikitake

October 24, 2014

Contents

1 Test Chapter 1 **1**
1.1 Why Erlang people need the Erlang Factory events 1
1.1.1 My impression at Erlang Factory SF Bay 2012 1
1.1.2 Talk Topics at Erlang Factory 1
1.1.3 My Talk Style 2
So I enjoyed it very much! 2

2 Test Chapter 2 **3**
2.1 Rewriting Loop with a Pair of Lists 3
2.1.1 Loop in Python 3
2.1.2 Loop in Erlang 3
2.2 A simple recursive loop 3
2.2.1 A ring buffer in Erlang 4

Chapter 1

Test Chapter 1

This is a text document for Kenji Rikitake's LaTeX presentation for SphinxCon JP 2014.

1.1 Why Erlang people need the Erlang Factory events

This section is quoted from my own blog article at the following URL, and modified by me:

http://concurrently-chaotic.blogspot.jp/2012/04/why-erlang-people-need-erlang-factory.html

1.1.1 My impression at Erlang Factory SF Bay 2012

Disclaimer: I am writing this blog article on my way home from SFO. The reduced air pressure in the passenger cabin might have affected the contents of this article.

I spent all three days of Erlang Factory SF Bay 2012 with two more days to cope with the jet lag. This was my third consecutive time to participate in the event as an invited speaker.

One of the most impressive things about this conference is that Francesco Cesarini, the leader of this event and CTO of the hosting company Erlang Solutions , carefully treats *all* delegates (which stands for the participants) equally with kindness and hospitality. Francesco's principles are well understood by all the other Erlang Solutions staff members as well.

I think Erlang Factory provides integrated feelings of satisfaction to all the delegated by letting them exchange the serious but friendly discussions. This is *not* something you can find over teleconferences or any forms of distant communication either synchronous or asynchronous. Erlang Factory is *not* a decision-making event; it is about sharing ideas and hanging out with each other, supporting both mental and emotional needs of the delegates.

1.1.2 Talk Topics at Erlang Factory

Erlang Factory conference accepts a large number of versatile topics including:

- Distributed databases
- Case studies in production systems
- Detailed debugging into the library and the host operating systems
- Crash courses for programming and testing the language system, and
- *Official announcements* from Ericsson's OTP Team, who owns the final responsibility of managing and directing the language system's future.

An opportunity to discuss various complex issues with the speakers *and* the audiences is itself rare and precious, especially when the issues share the same core interests: Erlang/OTP.

Erlang Factory also accepts non-mainstream topics as a talk proposal. In the past three years, I talked about RPC over SSH, Mersenne Twister implementation with the NIFs, and in this year 2012 it was about IPv6 readiness and programming tips of the language system. Those are not necessarily the mainstream topics in the Erlang/OTP community, though I find quite a few problem reports on the online discussion places, such as in the erlang-questions mailing list.

1.1.3 My Talk Style

My style of talk at the Erlang Factory is nothing extraordinary:

- First I analyze the problem and define the detailed problem domain to solve;
- then I write some example code to solve them and put them on the Web including GitHub and the other sharing places *before* finalizing the talk so that I could make it better with the wisdoms of others;
- I also experiment and evaluate the results while I'm finalizing the code and the presentation materials;
- Then at the Erlang Factory talks I discuss the conclusions found and propose the unresolved issues which may become another interesting talk topic.

The important points are sticking to the facts, no exaggeration, distinguishing assumptions from the facts, and listening to what the audiences ask and comment. Actually this is the same procedure I will take for the ACM Erlang Workshops, but for the Erlang Factory I try to make the presentation including more practical aspects and open questions.

I usually assign three months as a part-time project for an Erlang Factory presentation. The toughest part is to decide what to talk about. It's actually Francesco who suggested me to give a talk about IPv6 for this year 2012. I had some experience dealt with the technical issues of IPv6, so I decided to take the suggestion. The important point is that what you are going to talk is negotiable with the Erlang Factory staff.

So I enjoyed it very much!

This year 2012's events of Erlang Factory SF Bay Area were very intense, both in formal and casual senses. I've got an impression that it has become a truly international-class conference, while retaining the casual and friendly style of the past conferences.

I believe people go for conferences and events to meet people and share the feelings. It's a sort of festival, or *Matsuri* in Japanese, though also with the practical exchange of ideas. I strongly suggest all the Erlang/OTP enthusiasts to participate in this superb Matsuri.

Chapter 2

Test Chapter 2

This is a part of test document for Kenji Rikitake's LaTeX presentation for SphinxCon JP 2014.

2.1 Rewriting Loop with a Pair of Lists

Iteration loops are not necessarily a fundamental element of computer languages. Python is one of the popular languages which have the loop structure embedded. On the other hand, Erlang is *not*.

2.1.1 Loop in Python

Python loop is simple. Here's an example, of counting up a number, and choosing each member in a list, in Python 3:

```
#!/usr/bin/env python3

if __name__ == '__main__':
    i = 0
    while i < 10:
        print(i, end=" ")
        i += 1
    print(end="\n")

    l = ["one","two","three"]
    for i in l:
        print (i, end=" ")
    print(end="\n")
```

The execution result of this code is:

```
0 1 2 3 4 5 6 7 8 9
one two three
```

2.1.2 Loop in Erlang

2.2 A simple recursive loop

In Erlang, you need to make a recursive function to iterate:

```
-module(test).

-export([
```

```
        loop1/1,
        loop2/1,
        main/0
        ]).

-spec loop1(non_neg_integer()) -> ok.

loop1(N) ->
    loop1(N, 0).

-spec loop1(non_neg_integer(), non_neg_integer()) -> ok.

loop1(0, _) ->
    ok = io:format("~n");
loop1(N, V) ->
    ok = io:format("~B ", [V]),
    loop1(N - 1, V + 1).

-spec loop2(list(string())) -> ok.

loop2([]) ->
    ok = io:format("~n");
loop2(L) ->
    [H|T] = L,
    ok = io:format("~s ", [H]),
    loop2(T).

-spec main() -> ok.

main() ->
    loop1(10),
    loop2(["one", "two", "three"]).
```

The execution result of this code is:

```
Erlang/OTP 17 [erts-6.2] [source] [64-bit] [smp:8:8] [async-threads:10] [kernel-poll:false] [dtrace]

Eshell V6.2  (abort with ^G)
1> c(test).
{ok,test}
2> l(test).
{module,test}
3> test:main().
0 1 2 3 4 5 6 7 8 9
one two three
ok
```

2.2.1 A ring buffer in Erlang

Here, another example of loop using a ring buffer, which maintaines a fixed number of elements in the buffer structure, shifting each element one by one, with the new element added every time at the tail.

In Erlang, splitting a list into the single head element and the tail elements is a part of the fundamental operation and can be performed at a very low cost. Removing the first head element from a list, and adding another element on the head of the list is a low-cost operation. On the other hand, *appending an element to the tail* is relatively a higher cost operation in Erlang.

In the following example, a ring buffer list is split into two lists L and RL as follows:

- A list L is used so that the head value of the list L is taken and removed.
- Another list RL is used so that the appended value is added to the *head* of the list RL.

- When L becomes a null list, a reversed list of RL computed by `lists:reverse/1` is assigned as the new L, and a null list is assigned to the new RL.

Here is an example of the code:

```
%%% source code quoted from:
%%% https://gist.github.com/jj1bdx/cae6012d5d7c3a5d0a4d

-module(buftest).

-export([
        loop/1
    ]).

calc(H, H2) ->
    ok = io:format("H = ~p, H2 = ~p~n", [H, H2]),
    H2.

loop({[H], RL}) ->
    NL = lists:reverse(RL),
    loop({[H|NL], []});
loop({L, RL}) ->
    [H|L2] = L,
    [H2|L3] = L2,
    % here in calc/1 you can add an arbitrary value
    NH2 = calc(H, H2),
    NL2 = [NH2|L3],
    NRL = [H|RL],
    {NL2, NRL}.
```

And here's an example of the execution, which simply rotates each element in the list as a ring:

```
Erlang/OTP 17 [erts-6.2] [source] [64-bit] [smp:8:8] [async-threads:10] [kernel-poll:false] [dtrace]

Eshell V6.2  (abort with ^G)
1> l(buftest).
{module,buftest}
2> L1 = buftest:loop({[a,b,c],[]}).
H = a, H2 = b
{[b,c],[a]}
3> L2 = buftest:loop(L1).
H = b, H2 = c
{[c],[b,a]}
4> L3 = buftest:loop(L2).
H = c, H2 = a
{[a,b],[c]}
5> L4 = buftest:loop(L3).
H = a, H2 = b
{[b],[a,c]}
6> L5 = buftest:loop(L4).
H = b, H2 = c
{[c,a],[b]}
7> L6 = buftest:loop(L5).
H = c, H2 = a
{[a],[c,b]}
8> L7 = buftest:loop(L6).
H = a, H2 = b
{[b,c],[a]}
9> L8 = buftest:loop(L7).
H = b, H2 = c
{[c],[b,a]}
```