Erlang Cheat Sheet

alpha-0.2 2010/4/30
By @kuenishi.

Basics

Integers

```
1, -34, 16#DEADBEEF, 2#101, 32#23, $A
```

transparent for Bignums: no limit.

Floats

```
23.234, -4.435, 2.34E-10.
```

Atoms

```
foo, foo_bar, 'Good speed\n', '42', smCamel, 'you"ll be someone\\\\'
```

single-quoting makes atom. No more than 1M atoms in one node. `list_to_atom/1, atom_to_list/1` is available.

Tuples

```
{foo, hoge_huga}, {name, "foo"}, {value, 23, {do, it}}, {}
```

`append/2, erlang:append_element/2` is available.

Lists

```
[foo, hoge_huga], [{name, "foo"}], [[value, 23], {do, it}}], []
```

see `lists(3)` for more good handlings. String is list of chars, just an array of ascii-ranged integer. list comprehensions is as follows:

```
[ Expr(X) || X <- ListX, SomeQualifier]
```

Variables

```
Foo, Hoge_Variable, SomeCamel
```

Variables start with Capitalized letter. Those complex structures can be arbitrarily combined.

Pattern Matching of case

```
A=12, [B|C]=[1,2,3], {ok,V}=acall(),

case acall() of
  {ok, V}-> some; {error,R}-> hoge;
  {list,[H|L]}-> other, _-> error end.
```

Functions and guards

```
some_fun(ok)-> somehoge;
some_fun(error)-> hogehoge;
some_fun(_)-> other.

other_fun(List) when is_list(List)-> hoge;
other_fun(Int) when 0 <= Int, Int < 23->
foo;

F = fun(X) when is_integer(X)-> integer;
       (X) when is_float(X)-> float end.
```

sequence of guards are separated by ',' for "and", ';' for "or". following tests are available: `is_atom/1, is_binary/1, is_bitstring/1, is_float/1, is_function/1, is_function/2, is_integer/1, is_list/1, is_number/1, is_pid/1, is_port/1, is_record/2, is_record/3, is_reference/1, is_tuple/1, >, <, =:=, ==, =/=, >=, <=, andalso, orelse, abs/1, bit_size/1, byte_size/1, element/2, float/1, hd/1, length/1, node/0, node/1, round/1, self/0, size/1, tl/1, trunc/1, tuple_size/1`

Function Calls

```
some_module:other_fun(Argv, hoge),
apply(some_module, other_fun, [Argv, hoge])
```

Starting the System

Just type `erl` to start erlang. Useful options:

-make	build all .erl files, seeing Emakefile
-man *name*	see manpage for name
-name *name*	start erts with longname '*name*'
-pa, -pz	add load list
+K true	enable Kernel polling

see `erl -man erl` for more.

Shell

`b().`	see all bound variables
`f().`	forget all bound variables
`f(X).`	forget X.
`i().`	see all process infos.

Traversing Lists, Lists and Accumlators,

```
sum([])-> 0;
sum([H|L]) when is_integer(H)-> H+sum(L).
```

or same result:

```
lists:foldl(fun(N,S)->N+S end, 0, IntList).
```

Record

```
-record(point, {x=0, y=0, z=0}).
R = #record{}, %{point, 0, 0, 0}
S = #record{x=1}, %{point, 1, 0, 0}
T = R#record{y=1}, %{point, 1, 1, 0}
R#record.z --> 0.
```

Record is a syntax sugar of tuple.

Messaging

Sending - usually send self pid for recving reply:

```
Destination ! {self(), any_term, Payload}
```

Receiving - usually does pattern matching and timeout, with tail recursion for looping:

```
receive
  {From, any_term, Payload} when is_pid(From)->
     do_some_right_thing(), From ! {reply, ok};
  {From, _, _}-> %unknown protocol
     its_error(), From ! {reply, error};
  after 1024-> % timeout after 1024 ms
     do_other_things()
end.
```

Any notifications such as callbacks, signals, driver-call replies, socket event listening have messaging-interface, program can wait for different-kind multiple events easily, such as "socket-readable events, server control commands and timer events" at one `receive` sentence.

Reserved words

```
after and andalso band begin bnot bor bsl bsr
bxor case catch cond div end fun if let not of
or orelse query receive rem try when xor
```

exceptions: try...catch

exceptions has 3 classes. they can be caught separately.

```
try
  some_sequence()
catch
  error:E -> handle_error(E);
  exit:E  -> handle_exit(E);
  throw:T -> handle_throw(T)
end.
```

error Reasons

class error is given as `{Reason, Stack}`. throws as follows:

Reason	Type of error
`badarg`	The argument is of wrong data type, or badly formed.
`badarith`	Too much or less arguments
`{badmatch, V}`	no matching expression for V
`function_clause`	no matching function
`{case_clause, V}`	no matching case clause for V
`{try_clause, V}`	bad catch match expression for V
`undef`	no function found
`{badfun, F}`	something wrong in function F
`{badarity, F}`	wrong number of arguments for F
`timeout_value`	bad timeout value for `after`
`noproc`	the process you're trying to link to does not exist
`{nocatch, V}`	no catch clause for throw
`system_limit`	system limit.

Linking and trap_exit

```
A=self(),
B=spawn_link(Mod, Fun, Argv),
process_flag(trap_exit, true),
exit(Reason).
```

Then process B will receive message `{'EXIT', A, Reason}`. Exit signal will propagate to all linked processes. The recipient will also exists unless it handles the signal. `register/1` is also available. or you can write as follows:

```
case catch somefun(X) of
  {'Exit', Reason}-> ...;
  _-> other
end
```

fork-execing unix program

so easy:

```
Port=open_port({spawn, "someprogram"},[]),
Port ! {msg, "like a normal byte stream"}.
```

Of course binary streaming is also available.

Reference

```
Ref=erlang:make_ref(),
```

System Infos

```
erlang:memory(), processes(), process_info
(Pid), Module:module_info()
```

term infos can be obtained by using `erts_debug:size/1`, `erts_debug:flat_size/1`.

reference: see www.erlang.org/course/course.html for basics

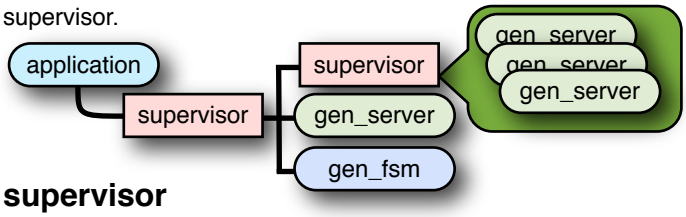
erlang.el

Emacs' erlang-mode file is in `lib/tools-x.y.z/emacs/erlang.el`. After installation into your emacs, commands will be shown by `M-x tempo-[tab]`.

OTP

supervision tree

Supervision tree is a process hierarchy for watching processes. `application` is for root node, `supervisor` for middle nodes, `gen_server/gen_fsm` for leaf nodes. If application name is hoge, `hoge.app` file is needed in the path. In `Mod:start/1`, kick supervisor.



supervisor

Supervisor's 4 restart strategies:

`one_for_one`	when one child dies, only that process will be restarted
`one_for_all`	when one child dies, all processes will be restarted
`rest_for_one`	processes 'rest' in start order will be terminated before other processes be terminated.
`simple_one_for_one`	same as `one_for_one` and additional feature: dynamically adding child.

child_spec() = {Id,StartFunc,Restart,Shutdown,Type,Modules}
supervisor:start_link(SupName, Module, Args)-> {ok, Pid}
supervisor:start_child(SupRef, ChildSpec)-> {ok, ChildPid}

gen_server

is GENeric Server - users 'inherit' this behaviour by overriding following functions, mainly:

`init/1`	when one child dies, only that process will be restarted
`handle_call/3`	when one child dies, all processes will be restarted
`handle_cast/2`	processes 'rest' in start order will be terminated before other processes be terminated.
`handle_info/3`	same as `one_for_one` and additional feature: dynamically adding child.

Then process B will receive message `{'EXIT', A, Reason}`. Exit signal will propagate to all linked processes. The recipient will also exists unless it handles the signal. `register/1` is also available. or you can write as follows:

gen_fsm
gen_event

error_logger, reltool, systool, .rel
failover, takeover

see www.erlang.org/doc/design_principles/users_guide.html for otp guide

Frequent Module usages
lists, proplists, dict, gen_tcp, gen_udp, file io
erlang:decode_packet, eunit vs common_test
Drivers, edoc, dets/ets/mnesia
Plloaso mako nuto uf cakso dodtos

see also: Bit String Pattern Matching and Bit String