Control Systems Optimization

Igor Wojnicki

AGH – Univeristy of Science and Technology

2010

Outline

1 Erlang Error Handling

1 Erlang Error Handling

Runtime Errors

Runtime errors are exceptions thrown by the system.

- `function_clause` – none of existing function clauses matches the arguments.
- `case_clause` – as the above but for a case statement.
- `if_clause` – none of the existing expressions in an if statement matches true.
- `badmatch` – occurs when pattern matching fails.
- `badarg` – if a BIF is called with wrong arguments.
- `undef` – an undefined function is called.
- `badarith` – innapropriate argument for arithmetical operations.

function_clause

None of existing function clauses matches the arguments.

```
factorial(N) when N > 0 ->
  N * factorial(N - 1);
factorial(0) -> 1.
1> test:factorial(-1).
 ** exception error:
    no function clause matching test:factorial(-1)
```

case_clause

As the above but for a case statement.

```
test1(N) ->
  case N of
    -1 -> false;
     1 -> true
  end.

1> test:test1(0).
 ** exception error: no case clause matching 0
in function test:test1/1
```

if_clause

None of the existing expressions in an if statement matches true.

```
test2(N) ->
  if
    N < 0 -> false;
    N > 0 -> true
  end.

1> test:test2(0).
 ** exception error: no true branch found
    when evaluating an if expression
    in function foo:test2/1
```

badmatch

Occurs when pattern matching fails.

```
1> N=45.
45
2> {N,M}={23,45}.
 ** exception error: no match of right hand side value {23,45}
```

badarg

If a BIF is called with wrong arguments.

```
1> length(helloWorld).
 ** exception error: bad argument
in function length/1
called as length(helloWorld)
```

undef

An undefined function is called.

```
1> test:hello().
 ** exception error: undefined function test:hello/0
```

badarith

Inappropriate argument for arithmetical operations.

```
1> 1+a.
 ** exception error: bad argument in an arithmetic expression
in operator +/2
called as 1 + a
```

Handling Errors

- Do nothing let it fail. . . and have the process restarted.
- Catch the exception and handle it yourself.

Catching

- try...catch construct, a case expression on steroids.

```
try Exprs of
  Pattern1 [when Guard1] -> ExpressionBody1;
  Pattern2 [when Guard2] -> ExpressionBody2
catch
  [Class1:]ExceptionPattern1
    [when ExceptionGuardSeq1] -> ExceptionBody1;
  [ClassN:]ExceptionPatternN
    [when ExceptionGuardSeqN] -> ExceptionBodyN
end
```

- of clause can be omitted, if there is no test for the Exprs value needed.
- after can be omitted too.

Classes

If the class is omitted from the catch clause, `throw` is assumed.

- `error` – the most general class: Runtime Errors. Can be triggered on purpose with `erlang:error(Term)`
- `throw` – generated by explicit call to `throw/1`, discouraged.
- `exit` – raised by calling `exit/1` or by exit signal.

Exceptional Examples ;) 1

```
38> spawn(fun() -> math:sqrt(-1) end).

=ERROR REPORT==== 3-Jan-2011::17:12:46 ===
Error in process <0.92.0> with exit value:
               {badarith,[{math,sqrt,[-1]}]}

<0.92.0>
39> math:sqrt(-1).
 ** exception error: bad argument in an arithmetic expression
     in function  math:sqrt/1
        called as math:sqrt(-1)
```

Exceptional Examples ;) 2 I

```
generate_exception(1)->a;
generate_exception(2)->throw(a);
generate_exception(3)->exit(a);
generate_exception(4)->{'EXIT', a};
generate_exception(5)->erlang:error(a).
demo1() ->
  [catcher(I) || I <- [1,2,3,4,5]].
catcher(N) ->
  try generate_exception(N) of
    Val -> {N, normal, Val}
  catch
    throw:X -> {N, caught, thrown, X};
    exit:X -> {N, caught, exited, X};
    error:X -> {N, caught, error, X}
  end.
```

Exceptional Examples ;) 2 II

```
catcher(N) ->
  try generate_exception(N) of
    Val -> {N, normal, Val}
  catch
    throw:X -> {N, caught, thrown, X};
    exit:X -> {N, caught, exited, X};
    error:X -> {N, caught, error, X}
  end.

> try_test:demo1().
[{1,normal,a},
{2,caught,thrown,a},
{3,caught,exited,a},
{4,normal,{'EXIT',a}},
{5,caught,error,a}]
```

Programming Style with Exceptions

- f/1 returning information regarding an error:

```
...
case f(X) of
  {ok, Val} ->
    do_some_thing_with(Val);
  {error, Why} ->
    %% ... do something with the error ...
end,
...
```

- f/1 throwing an exception upon error:

```
...
{ok, Val} = f(X),
do_some_thing_with(Val);
...
```

Stack Traces

- a list of the functions on the stack to which the function will return if it returns.

```
demo3() ->
  try generate_exception(5)
  catch
    error:X ->
      {X, erlang:get_stacktrace()}
  end.

1> try_test:demo3().
{a,[{try_test,generate_exception,1},
{try_test,demo3,0},
{erl_eval,do_apply,5},
{shell,exprs,6},
{shell,eval_loop,3}]}
```

Stack Traces Another Example I

```
a(X) -> b(X),0.
b(X) -> 2/X.

11> a:a(2).
0
12> a:a(0).
 ** exception error: bad argument in an arithmetic expression
      in function  a:b/1
      in call from a:a/1
```

Stack Traces Another Example II

```
13> try a:a(0) catch X:Y -> {X,Y} end.
{error,badarith}
14> try a:a(0) catch X:Y -> {X,Y,erlang:get_stacktrace()} end.
{error,badarith,
       [{a,b,1},
        {a,a,1},
        {erl_eval,do_apply,5},
        {erl_eval,try_clauses,8},
        {shell,exprs,6},
        {shell,eval_exprs,6},
        {shell,eval_loop,3}]}
```