Erlang Superpowers

Jack Moffitt
@metajack
jack@metajack.im
http://metajack.im

Syntax

Prolog derived

Atoms

atom

foo_bar

jack@metajack.im

Variables

Var

FooBar

_

Lists

```
[1, 2, apple]
```

Tuples

```
{1, 2, apple}
```

Strings

```
"abc"
"abc" = [97, 98, 99].
```

Binaries

```
<<"something">>
<<Data:160/big-unsigned-integer>>
```

Functions

```
foo(Arg1, Arg2) ->
    io:format("hi!"),
    ok.
```

Functions

```
foo(0) ->
  abc;
foo(1) ->
  def;
foo(_) ->
  oops.
```

Messages

```
Pid ! some_message.
name ! {hello, 1}.
receive
    {hello, N} ->
        N;
    _ ->
        0
end.
```

Overview

Functional

Pattern matching

```
Foo = bar.

[Head | Tail] = foo().

{[H | _] = List} = bar().
```

```
handle_call({submit_word, Word},
            {Player, _},
            State = #game{
              words=Words,
              scores=[Score1, Score2],
              player_data=[
                #player{rating=Rating1} = PlayerData1,
                #player{rating=Rating2} = PlayerData2]}) ->

    %% ...

    {reply, ok, NewState}.
```

Binary matching

IPv4 Packet

```
<<?IP_VERSION:4, HLen:4, SrvcType:8, TotLen:16,
ID:16, Flgs:3, FragOff:13, TTL:8, Proto:8, HdrChkSum:16,
SrcIP:32, DestIP:32, RestDgram/binary>> = Packet.
```

MP3 Frame Header

```
decode_header(<<2#11111111111:11,B:2,C:2,_D:1,E:4,F:2,G:1,Bits:9>>) ->

    %% header code
```

Light-weight processes

Actor model

Distributed

Fault tolerance

OTP library

History

1982–1985

Exploration begins at Ericsson Computer Science Lab

1985–1988

Birth of Erlang

1989

JAM speeds up Erlang

1989-1993

Erlang matures and is demonstrated

1995

Collapse of AXE-N

1998

The bi-polar year

Erlang in action

Snack Words

Protocol design

Length-prefixed JSON

2 bytes for packet length (L)

L bytes of raw JSON data

In Erlang:

gen_tcp option

{packet, N}

N=1,2, or 4

Receive:

```
{tcp, socket, Data}
```

Decode:

```
mochijson2:decode(Data)
```

Encode:

```
Data = mochijson2:encode(Json)
```

Send:

```
gen_tcp:send(Socket, Data)
```

In Objective-C:

```
- (void)sendAvailableData {
      NSUInteger sentBytes, maxLen;
      do {
            if (outBuffer == nil) {
                  if ([sendQueue count] > 0) {
                        id jsonObj = [sendQueue objectAtIndex:0];
                        [sendQueue removeObjectAtIndex:0];
                        NSString *jsonString = [jsonObj JSONRepresentation];

                        NSLog(@"SENDING: %@", jsonString);

                        NSData *data = [jsonString dataUsingEncoding:NSUTF8StringEncoding];
                        NSUInteger len = htons([data length]);

                        [outBuffer release];
                        outBuffer = [[NSMutableData dataWithCapacity:[data length] + 2] retain];
                        [outBuffer appendBytes:&len length:2];
                        [outBuffer appendData:data];
                        outPos = 0;
                  } else {
                        // no more data to send
                        break;
                  }
            }

            maxLen = ([outBuffer length] - outPos) < 4096 ? ([outBuffer length] - outPos) : 4096;
            sentBytes = [outStream write:([outBuffer mutableBytes] + outPos) maxLength:maxLen];
            outPos += sentBytes;

            if ((outPos + 1) >= [outBuffer length]) {
                  [outBuffer release];
                  outBuffer = nil;
                  outPos = 0;
            }
      } while (sentBytes >= maxLen);
}
```

JSON payload:

```
["play"]
["submit", {"word": "erlang"}]
```

In Erlang:

```
idle({data, [<<"play">> | _]},
     StateData = #sd{player=Player}) ->

%% handle command
```

In Objective-C:

```
- (id)addHandlerForCommand:(NSString *)command
     notifyObject:(id)object
     selector:(SEL)selector
     repeat:(BOOL)repeat;


- (id)addHandlerForCommand:(NSString *)command
   withBlock:(LLSWClientBlock)block
     repeat:(BOOL)repeat;
```

Game logic

Players

Each player is a finite state machine

States

starting, authing, idle, waiting, playing

```
authing({data, [<<"auth">>, Props]}, State) ->

      %% authenticate

waiting({game_start, Game, Letters, Time, Players},
        State) ->

      %% send notification to client

playing({data, [<<"submit">>, Props]},
        StateData = #sd{game=Game}) ->

      %% handle word
```

Games

Each game must track and communicate moves, time, and scoring

Games receive:

timer events - `tick`

moves - `{submit_word, Word}`

Games send:

```
{game_start, ...}
{clock, TimeLeft}
{word_found, Word, Who, ...}
{game_end, Winner, ...}
```

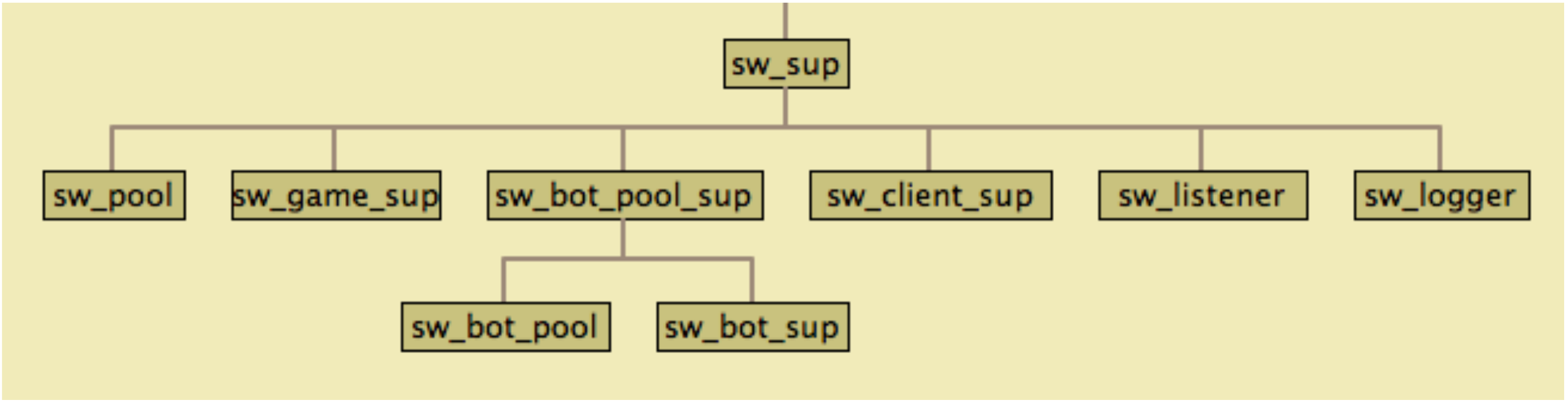
Server design

OTP application

Parts:

players, games, waiting pool, socket listener, bots

Supervision tree:



Process math:

1000 players

500 games

4 service processes

5 supervisors

= **1509 processes**

Persistence

A small database API stores data in Mnesia

Code Tour

```
handle_info({tcp, Socket, Data}, StateName, StateData) ->
    %% set the socket so we can receive another data packet
    ok = inet:setopts(Socket, [{active, once}]),

    %% verify that Data is a JSON object
    case catch mochijson2:decode(Data) of
        {'EXIT', _} ->
            {stop, invalid_json, StateData};
        [_Cmd] = Json ->
            ?MODULE:StateName({data, Json}, StateData);
        [_Cmd, _Props] = Json ->
            ?MODULE:StateName({data, Json}, StateData);
        _ ->
            {stop, bad_request, StateData}
end;
```

```
- (void)sendAvailableData {
    NSUInteger sentBytes, maxLen;
    do {
        if (outBuffer == nil) {
            if ([sendQueue count] > 0) {
                id jsonObj = [sendQueue objectAtIndex:0];
                [sendQueue removeObjectAtIndex:0];
                NSString *jsonString = [jsonObj JSONRepresentation];

                NSLog(@"SENDING: %@", jsonString);

                NSData *data = [jsonString dataUsingEncoding:NSUTF8StringEncoding];
                NSUInteger len = htons([data length]);

                [outBuffer release];
                outBuffer = [[NSMutableData dataWithCapacity:[data length] + 2] retain];
                [outBuffer appendBytes:&len length:2];
                [outBuffer appendData:data];
                outPos = 0;
            } else {
                // no more data to send
                break;
            }
        }

        maxLen = ([outBuffer length] - outPos) < 4096 ? ([outBuffer length] - outPos) : 4096;
        sentBytes = [outStream write:([outBuffer mutableBytes] + outPos) maxLength:maxLen];
        outPos += sentBytes;

        if ((outPos + 1) >= [outBuffer length]) {
            [outBuffer release];
            outBuffer = nil;
            outPos = 0;
        }
    } while (sentBytes >= maxLen);
}
```

```
idle({data, [<<"play">> | _]}, StateData = #sd{player=Player}) ->
    case sw_pool:join_pool(Player) of
        {ok, joined} ->
            send([play_ok], StateData),
            {next_state, waiting, StateData, ?PING_INTERVAL};
        {error, _} ->
            send([play_fail, {struct,
                              [{reason, <<"Internal error">>}]}],
                 StateData),
            {next_state, idle, StateData, ?PING_INTERVAL}
end;
```

```
-export([start/0,
         stop/1,
         get_player/3,
         create_player/2,
         create_player/5,
         save_player/2,
         move_player/3,
         authenticate/2,
         pick_puzzle/1,
         get_words/2,
         ...]).
```

Code Size:

sw_client.erl - 439 lines

sw_game.erl - 259 lines

elo.erl - 177 lines

sw_db_mnesia.erl - 566 lines

All code - 2416 lines

All tests - 857 lines

Performance

Load tested to 2000 simultaneous players on my laptop.
Runs on the cheapest Linode VPS.

Scaling

Changing persistence

Horizontal scaling

Thanks!

jack@metajack.im

@metajack

http://metajack.im

http://snackwords.com