Logger sentry

Trouble

- the error information from Logger is **dispersed** among every Elixir node
- it's **difficult** to check all error conveniently
- we need **aggregate** all error information through sentry dashboard

Solutions

- manually
- hack Logger

manually

```
error_msg = "some error information from tubitv elixir pro
Logger.error(error_msg)
Sentry.capture_exception(error_msg, [level: :error, stackt
```

advantage

- straightforward
- ...

disadvantage

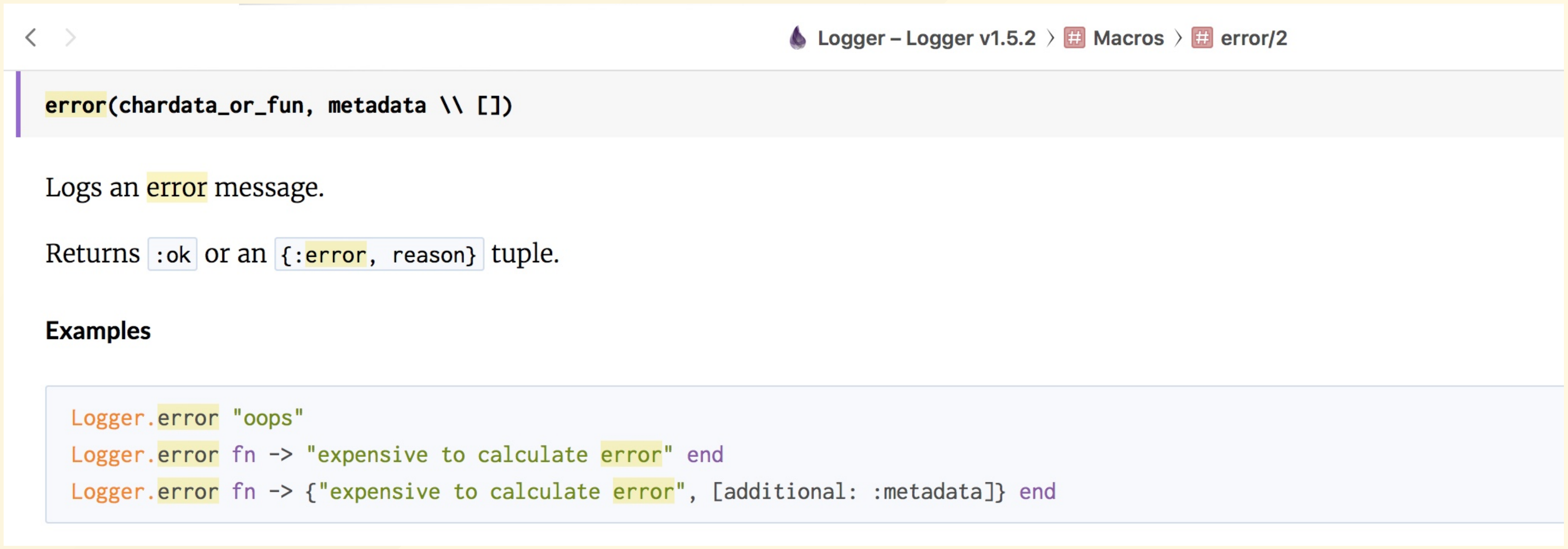
- ugly
- not graceful

hack Logger

- hack Logger.error
- console backend
- event model
- init event handler
- send msg to event server
- create one new backend
- the code for sentry backend

hack Logger.error

Logger.error will output the information to Elixir console, and it's api definition is:



Logger – Logger v1.5.2 › Macros › error/2

```
error(chardata_or_fun, metadata \\ [])
```

Logs an error message.

Returns `:ok` or an `{:error, reason}` tuple.

Examples

```
Logger.error "oops"
Logger.error fn -> "expensive to calculate error" end
Logger.error fn -> {"expensive to calculate error", [additional: :metadata]} end
```

add callback to error function? NO, can't !

console backend

output information to Elixir console, just because console backend.

code base for console backend.

key point:

- init
- log event

init

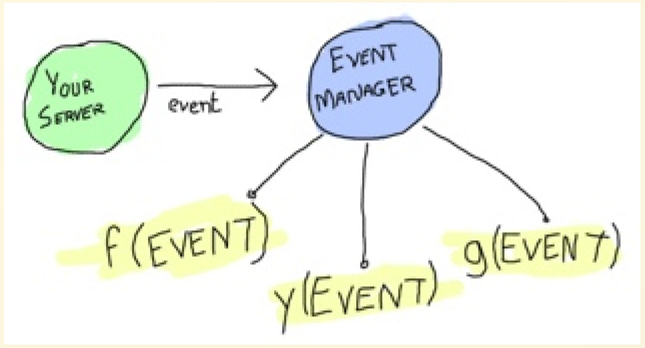
```
  defp init(config, state) do
    level = Keyword.get(config, :level)
    device = Keyword.get(config, :device, :user) # key of
    format = Logger.Formatter.compile Keyword.get(config,
    colors = configure_colors(config)
    metadata = Keyword.get(config, :metadata, []) |> confi
    max_buffer = Keyword.get(config, :max_buffer, 32)

    %{state | format: format, metadata: metadata,
              level: level, colors: colors, device: device
  end
```

log event

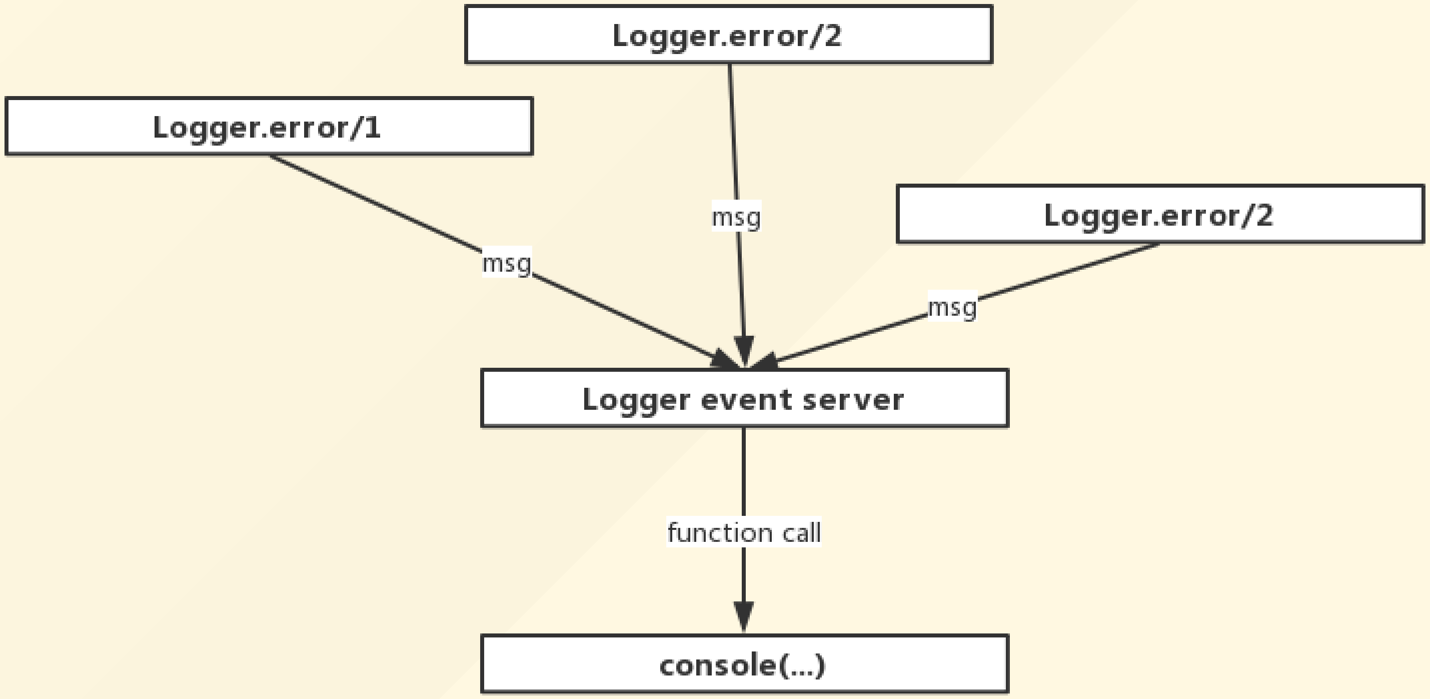
```
send(device, {:io_request, self(), ref, {:put_chars, :unic
```

But who initialized the console backend ? How come the event ?

event model



There is Logger event server in Elixir system.



init event handler

```
def init({mod, handler, args}) do
  case :gen_event.delete_handler(mod, handler, :ok) do
    {:error, :module_not_found} ->
      res = :gen_event.add_sup_handler(mod, handler, arg
      do_init(res, mod, handler)
    _ ->
      init({mod, handler, args})
  end
end
```

code base: [init event handler through watcher]()

It used gen event model, `gen_event` behaviour will call `Logger.Backends.Console.init/1` when execute `add_sup_handler` function.

send msg to event server

The `Logger.error/1,2` is Macro, it will call

```
   defp notify(:sync, msg),  do: :gen_event.sync_notify(Log
   defp notify(:async, msg), do: :gen_event.notify(Logger,
```

finally, and the code is here.

create one new backend

So, maybe we could create one new backend for our business requirements. And the Elixir allow this:

the document for backends in Elixir.

> `Logger` supports different backends where log messages are written to.
>
> Developers may also implement their own backends, an option that
> is explored in more detail below.

the code for sentry backend

the code is very simple.

The relationship for Logger and error_logger

Logger is event server in Elixir, it will send event message to Logger event server when we use Logger. `level`/1,2 .

`error_logger` is event server in Erlang, we can ignore it when we write Elixir code. But some error information will send to `error_logger` event server.

- gen process crash info inside Erlang system
- ...

handler for error_logger in Logger

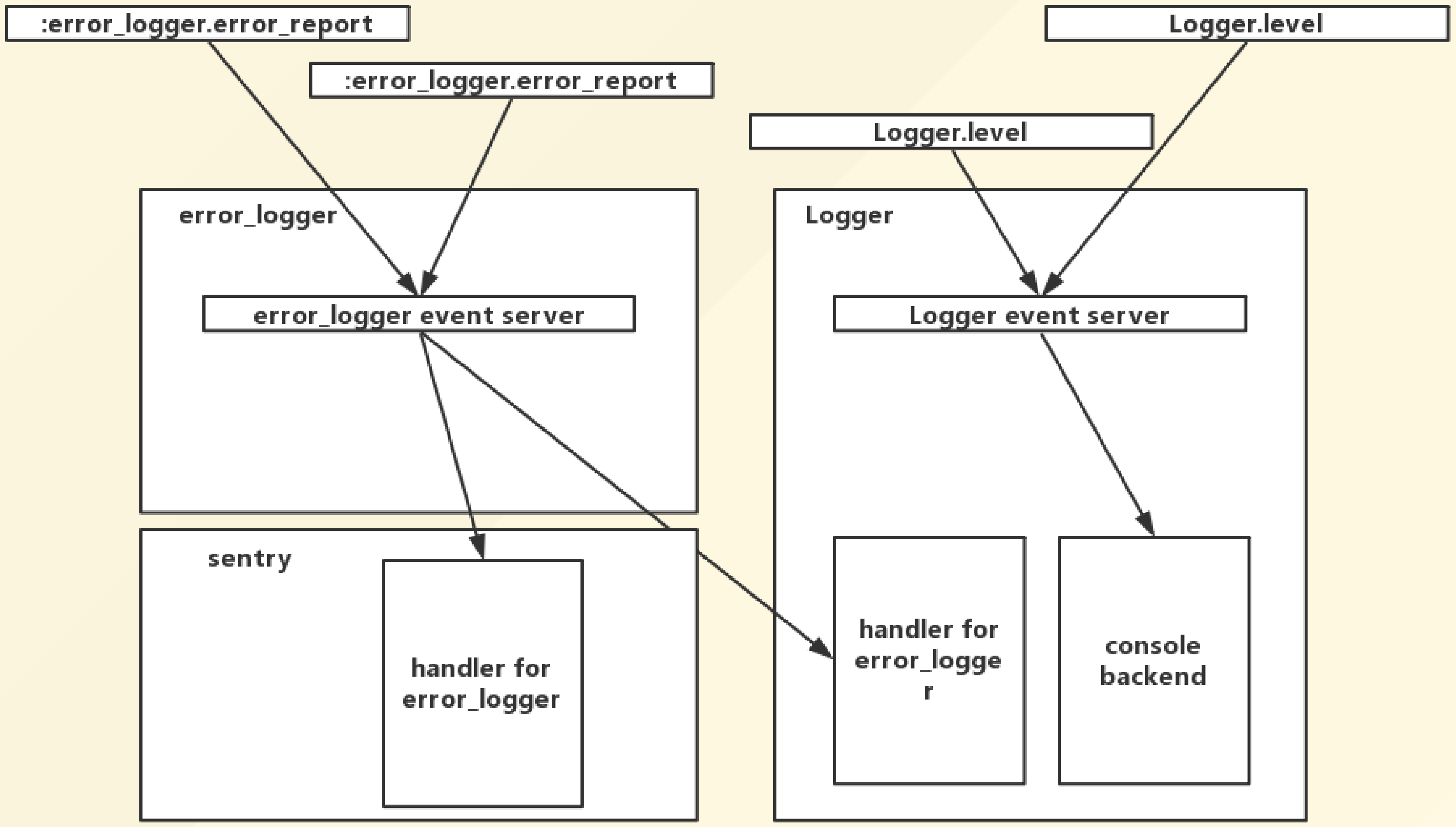
in Logger, there is one handler for `error_logger`.

handler for error_logger in sentry

in Sentry, there also has one handler for `error_logger`.

It will handle error message from `error_logger` event server, and push event to datadog dashboard in more friendly (to sentry dashboard) format.

relationship graph



Q&A