Zero-effort Structural Logging

London Scala Users Group, Scala Community Day, 14/12/2019

Septimal Mind Ltd
team@7mind.io

What would be the outcome of the following code?

```
1 val id = "john@doe.com"
2 val balance = 265
3 logger.info(s"User id=$id, balance=$balance")
```

```
User id=john@doe.com, balance=265
```

There may be a bit different code as well:

```
1 val id = "+13023072835"
2 val balance = 42
3 logger.info(s"User id=$id, balance=$balance")
```

```
User id=+13023072835, balance=42
```

let's assume we've collected many logs and now we want

let's assume we've collected many of these logs...

...and we want to filter logs lines for users...

...who registered with their email

```
grep -E -o "[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+[A-Za-z]2,6" file.txt
```

Oops...

Can we do it better?

We need structural logs

```
{
"id": "john@doe.com",
"id.type": "email",
"balance":"42"
}
```

Almost all the structural logging libraries are very inconvenient...

```
1  object Example {
2    val LOG = FluentLoggerFactory
3      .getLogger("fluentd.test")
4
5    // ...
6
7    val data =
8        new HashMap[String, String]()
9    data.put("id", userId)
10   data.put("balance", userBalance.toString)
11   LOG.log("user balance", data)
12 }
```

But the code...

```
1 val user = "JohnDoe"
2 logger.debug(s"Received a message from $user")
```

. . . is always structured!

```
Expr(Apply(Select(
  Apply(
    Select(Select(Ident("scala"), scala.StringContext),
      TermName("apply"))
      , List(Literal(Constant("Received a message from "))
          , Literal(Constant(""))
        )
  ),
  TermName("s")
  )
, List(Ident(TermName("user")))
))
```

LOGSTAGE

First-class logging framework for Scala

So, we made a library ...

which can extract structure ...

. . . so you may have text for humans

. . . and nice JSON for robots

```scala
object LoggerTest {
  def main(args: Array[String]): Unit = {
    import logstage.circe._
    val logger = IzLogger(sink = ConsoleSink.json(prettyPrint = true))
    val id = "user@gmail.com"
    val balance = 42
    logger.info(s"User with $id has balance $balance")
  }
}
```

ProfilesTest

LoggerTest

```
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/Contents/Home/bin/java ...
{
  "event" : {
    "balance" : 42,
    "id" : "user@gmail.com"
  },
  "meta" : {
    "class" : "1f8a415c",
    "logger" : "leaderboard.LoggerTest.main",
    "line" : 106,
    "file" : "tests.scala",
    "level" : "info",
    "timestamp" : 1576282110093,
    "datetime" : "2019-12-14T00:08:30.093Z[UTC]",
    "thread" : {
      "id" : 1,
```

Batteries included

1. `LogF[F[_]]`
2. `LogBIO[F[_, _]]`
3. Better alternative for MDC
4. `slf4j` adapter
5. etc, etc...

Dependency Injection for Scala done right

London Scala Users Group, Scala Community Day, 14/12/2019

Septimal Mind Ltd
team@7mind.io

Enigneers crave for good modularity...

...There were at least 4 talks at FS2019 on this subject

What we want from a good module system?

1. FP Support: ZIO envs, Effects, Resources...
2. Speed
3. Reliability: should fail fast, no reflection
4. Transparency: introspections
5. Scalability: should work for big apps

What we have?

1. TF/implicits — does not scale
2. Cake pattern — does not scale
3. Guice, Spring? — no, please!
4. Airframe? — no support for effects and resources
5. Macwire? — no support for effects and resources

Are we out of options?

distage

is a Dependency Injection mechanism for Scala

But it lacks negative traits of a typical DI thingy...
...and has many unique properties.
...and it's blazing fast.

You may call it a module system for Scala...
...with automatic dependency-pruning solver

What is Dependency Pruning in `distage`?

1. Your application and tests may specify their dependencies semi-dynamically
2. All the unnecessary components will not be even attempted to instantiate
3. Technically it's very similar to Garbage Collection in JVM

`distage` performs optimal ahead of time planning.
It doesn't do any unnecessary job.

With distage you can do this:

```
class JustATest[F[+_, +_]] {
  "service" must {
    "do something" in {
      (users: UserService[F], accounts: AccountingService[F]) =>
      for {
        user  <- users.create()
        balance <- accounts.getBalance(user)
      } yield {
        assert(balance == 0)
      }
    }
  }
}

object JustATestZioProd extends JustATest[zio.IO] with Prod
object JustATestZioDummy extends JustATest[zio.IO] with Dummy
```

And this:

```
laptop# docker run -ti company/product --run-all-services
prod01# docker run -ti company/product :analytics
prod02# docker run -ti company/product :accounting :users
```

Why it's a good idea to try `distage` now

1. Supports FP, wires Resources, Effects, ZIO Environments...
2. Transparent and predictable
3. Supports Roles and Dependency Pruning
   - May **drastically** simplify development and operations workflows
   - Helps you to write best tests possible without boilerplate
4. Promotes good practices and SOLID
5. Most of the job is done in compile-time
6. Safe
7. Blazing fast
8. no `scala-reflect` nor Java reflection, ScalaJS support is coming
9. is non-invasive.
   - You can add it to your project keeping business logic intact...
   - ...and remove it.
10. is not an ad-hoc thing, it has strong theory behind.

`distage-framework` is the most productive way to write maintainable pure functional applications with ZIO.
(And any other monad)

`distage-testkit` is the best way to make your tests performant and reliable.

`distage` is adopted by several different companies and tested by two years of production usage.

Thank you for your attention

docs: `https://izumi.7mind.io`

Septimal Mind is an Irish software consultancy and R&D Company

We're looking for clients, contributors, adopters and colleagues ;)

Contact: team@7mind.io

Github: `https://github.com/7mind`

Slides: `https://github.com/7mind/slides`

Follow us on Twitter

`https://twitter.com/shirshovp`

`https://twitter.com/kai_nyasha`