Scala Days

Gallia

A schema-aware Scala library for generic data transformation

Anthony Cros
Gallia Project

What: Gallia in a nutshell

Scala Days

Trivial example:

```
"""{"first": "anthony", "last": "cros", "age": 39, "fit": true}"""
  .read() // lazily + infers schema
    .nest("first", "last").under("name")
    .increment("age")
    .remove   ("fit")
  .display() // trigger meta then data transformation
```

prints **schema**:

```
<root>
  name
    first   _String
    last    _String
  age     _Int
```

prints **data**:

```
{ "name": {
      "first": "anthony",
      "last" : "cros"   },
  "age": 40 }
```

Gallia is "schema aware"

Scala Days

```
"""{"name": "anthony", "age": 39, "fit": true}"""
    .read(
        "name".string, "age".int, "fit".boolean)
        // or via: .read[Person]
      .toUpperCase("name")
      .increment  ("name") // nonsensical
      ~~.remove     ("fit")~~  // ignored
    .display()
    // Meta error
```

- Before processing data (potential schema inference notwithstanding)
- For any level of nesting/multiplicity
- For any operation where such checks can be realized
(won't catch IOOB errors for instance)

How do I get started?

Scala **Days**

```
mkdir /tmp/sd23 && cd /tmp/sd23

echo 'libraryDependencies +=
  "io.github.galliaproject" %% "gallia-core" % "0.4.0"' \
    > build.sbt # 2.12 or 2.13 only

sbt console
-----------------------------------------------------
scala> import gallia._
scala> """{"report": "TPS", "submitted": false}"""
             .read().flip("submitted").display()
```

Main goals

Scala Days

1. Practicality
2. Readability
3. Scalability (optionally)

When should I use it?

Scala Days

Python pandas: small dataframes

↖

Gallia: small <u>and</u> big data

↘

Apache Spark: big dataframes

not necessarily "**framed**" data (think nesting)

⇒ <u>One-stop shop paradigm</u> for data transformations*

Who's the team behind it?

(hopefully I'm standing somewhere down there)

FOR NOW!

Why not Case Classes + Collections?

1. May require **lots** of *case classes* (see DbNSFP example on github)
   - Input, output, and **intermediate** case classes
   - Lots of **boilerplate** code
   - Lots of **naming**: naming things judiciously is **hard**
2. Transformation code only commits to schema elements used
   - For instance: `val f: HeadS => HeadS = _.remove("name").flip("fit")`
   - Only "commits" to there being:
     - "name" field (whose type is irrelevant)
     - "fit"   field (boolean, because we flip it)
   - Great for **schema evolution** or **schema unification**
3. Data model may still be **changing**, or you may just be **poking around**

I/O Supported

Currently:

- {T,C}SV files: *plain*, *GZIP*, and *BZ2*
- JSON/jsonl : *plain*, *GZIP*, and *BZ2*
- Apache Avro and Parquet
- RDBMS: via JDBC
- Mongodb: read-only (for now)

In the future:

- Excel (yes, **lots** of research and clinical labs still rely on it)
- XML
- GraphQL, Sparql, Cypher (NEO4J), ..

Target selection

Scala Days

Choosing **what** to act on: its own meta-querying language

```
remove    (             "f1" )
remove    (             "parent" |> "child" )
remove    {             _.firstKey) }
remove    {             _.allBut ("f2", "f3") }
remove    {             _.findKey(_.endsWith("1")) }
remove    {             _.fullCustomKey { schema => [...] } }
transform { _.string("f1")          }.using([...])
transform { _.string(_.firstKey) }.using([...])
transform { _.string(_.allkeys  ) }.using([...])
```

Live coding session

Let's write some code!

More about scaling

(When Spark isn't available)

"Poor man's" scaling leveraging ***mergesort*** algorithm, think:

```
join -t$'\t' -o "1.1,2.6" -1 2 -2 7 \
  <(sort -t$'\t' -k2,2 \
    <(join -t$'\t' -o "1.8,2.7" -1 8 -2 8 \
      <(sort -t$'\t' -k8,8 <(tail -n+2 file1.tsv))   \
      <(sort -t$'\t' -k5,5 <(tail -n+2 file2.tsv)))) \
  <(sort -t$'\t' -k7,7 <(tail -n+2 file3.tsv))
```

Scaling: Poor man's version

Invoke via the use of `.iteratorMode`:

```
"/my/big/file/input.jsonl.gz2"
    .stream(
        _.iteratorMode.schema([...]))
      .decrement("fanciness")
      .groupBy([...])
      [...]
```

*CAVEAT: only works if **GNU sort** is available on your system (never tested with Windows/Mac)*

Scaling with Gallia in practice

Scala Days

See example of real world usage: *GeneMania* dataset processing

- **Poor man**/mergesort: GeneMania.scala
- **Gallia+Spark** equivalent: GeneManiaSparkDriver.scala

They share **the exact same code** for the transformations, and differ only in their I/O

More features

Scala Days

See this Towards Data Science article which introduce:

- **Union Types**: minimal support
- **Metaschema**:
    - As a result of supporting union types
    - Could dogfood schema transformations
- **Cotransformations**:
    - Via intermediate case classes representing a **subset** of schema
    - Great for encapsulation
    - For ***M-to-N*** field transformations where ***M>1*** and ***N>1***
    - Example: gist

So, why should I use Gallia?

Scala Days

1. Most common/useful data operations
2. Readable DSL
3. Scaling is not an afterthought
4. Meta-awareness
5. Can process individual entities
6. Can process deeply nested entities of any multiplicity
7. Provides flexible target selection
8. Execution DAG can be optimized easily thanks to Gallia's abstractions

Conclusion

Scala Days

- Seeking **feedback** to decide on future **directions**
- Welcome comparisons with **existing tools**
  - What would the complex biomedical data transformations (e.g. *dbNSFP*) look like with ***pandas***, ***Spark SQL***, etc?
  - Keeping in mind Gallia's **priorities**!
- Welcome **help** in general:
  - Terminology ("*deserialize1b*"...)
  - Documentation ("*deserialize1b*"???!!!)
  - Optimization
  - Semantic information?

contact.galliaproject@gmail.com

Scala Days 2023

Thanks!

G·A·L·L·I·A PROJECT

contact.galliaproject@gmail.com