Scala Center updates
Q3 2019 Advisory Board meeting

Scala Center team: Jorge Vicente Cantero, 100%; Ólafur Geirsson, 100% until July 31st; Julien Richard-Foy, 60%; Alexandre Archambault, 100%; ; Darja Jovanovic, 80%; Sébastien Doeraene, 100%

At a glance

- Dependency management (SCP-020)
- coursier
- MOOCs
- TASTy Reader for Scala 2 (SCP-018, start of the project)
- Metals
- Bloop
- Scala Libraries and Documentation
- Scala Compiler
- Scala.js
- Scala Days 2019 organisation
- SIP Meetings
- Communication and Community

Dependency management (SCP-020)

@alexarchambault, @sjrd

Most of the technical work for SCP-020 had been done in the previous cycles, but the documentation and global articulation of how the pieces contributed to addressing SCP-020 was previously not good enough. Therefore, we first recap everything related to that Recommendation.

Documentation of how sbt resolves dependencies

We published the following documentation about how coursier -- used by default in sbt as of v1.3.0 -- resolves dependencies: https://get-coursier.io/docs/other-version-handling. The documentation covers the precise algorithms, with examples.

Eugene Yokota from Lightbend then contrasted the above documentation with the old algorithms used by Ivy (which was used by default until sbt 1.2.x): "Dependency resolver semantics".

Better conflict manager

coursier, whether in sbt plugin form or integrated in sbt 1.3.x, has been enhanced to support the existing conflictManager setting of sbt. In particular, it supports the Strict mode which refuses to reconcile two incompatible versions.

Going further, coursier was enhanced with a low-level mechanism of Rules to implement more advanced resolution strategies. Recently these low-level mechanisms were exposed as a public API with a generic concept of Reconciliation. With sbt-coursier, they allow to specify, per organization/artifact pair (or patterns for organization/artifact pairs), how different versions should be reconciled. Reconciliation strategies have already been released for existing semantics:

- Default, for coursier's default strategy, as documented above
- Relaxed, matching sbt's old default strategy
- Strict, which works like Default except it refuses to replace a specific version by another one
- SemVer, which works like Default except it refuses to replace a specific version V.x by another one U.y if V and U are different (so that 1.2.5 can be replaced by 1.3.7 but not by 2.1.0)

Since Reconciliation strategies can be specified per artifact, they can encode once and for all the compatibility guarantees offered by individual libraries. An example would be cats-core, which guarantees SemVer compatibility, and can therefore be specified as follows (requires sbt-coursier):

```
versionReconciliation += "org.typelevel" %% "cats-core" % "semver"
```

Now, if the codebase transitively depends on two versions of cats-core that are SemVer-compatible, the most recent one will be used; but if it depends on two different major versions, the resolution will report a conflict.

Static analysis to check potential LinkageErrors

The Reconciliation strategies above are great if libraries provide clear guarantees about binary compatibility. If they do not, we can go one step further, and, given a particular resolution, statically analyze the classpath to check, ahead of time, for potential LinkageErrors that could happen at run-time.

The static analysis itself was already available as the missinglink tool, developed by Spotify. However, it was only available for Maven. We developed an sbt plugin, namely sbt-missinglink, to allow using it from sbt codebases. Using that plugin, checking a particular codebase for potential conflicts is as easy as a) adding the following sbt plugin:

```
addSbtPlugin("ch.epfl.scala" % "sbt-missinglink" % "0.1.0")
```

then running the sbt task

```
> missinglinkCheck
```

We will continue to develop sbt-missinglink with further settings to customize the behavior, notably to filter out certain conflicts (similar to what MiMa allows). Should the need arise, we could also advance missinglink itself.

A blog post documenting all this for the public will be released in the next few days. You may read the work-in-progress in this gist, although most points have been addressed hereinabove.

coursier

@alexarchambault

Native CLI

We managed to have the CLI of coursier run as a native executable, via GraalVM native image, running significantly faster. Various changes were needed to have each command work fine from the native executable (the `launch` commands starts a JVM to launch applications, the `bootstrap` and `install` commands needed not to rely on reflection to detect main classes in a classpath, etc.).

Native executables for Linux and OS X are now pushed upon release as GitHub release assets (as `cs-x86_64-apple-darwin` and `cs-x86_64-pc-linux` for OS X and Linux).

As the Windows support in GraalVM somewhat lags behind the one of Linux and OS X, we did not manage to generate native executables for Windows yet. As a workaround, we rely on jlink via sbt-native-packager to generate a launcher that can be run without requiring a JVM. (As the jlink archive ships with its own stripped down JVM.) These can be found in the GitHub release assets as `standalone-x86_64-pc-win32.zip`.

Overall, we now have launchers for the CLI of coursier that do not need a JVM to run, on most major platforms (Linux, OS X, Windows). This opens up the possibility of offering to install and manage JVMs ourselves. Our goal is not to be as featureful as SDKMAN or jabba, but still offer users to install the most common JVMs if needed.

Optimizations

We optimized various aspects of coursier, be it by needing less network round trips, or by lowering CPU utilization.

Network requests

Some Maven repositories provide MD5 and SHA-1 checksums via HTTP headers when downloading files. These are now used by coursier, avoiding to download checksum files altogether.

More not-found errors are now kept in cache. This speeds up the version listing and dependency string completion capabilities of coursier, that can rely on not-found errors rather than checking directory listings for the existence of some files beforehand.

CPU usage

We replaced some regexes processing POM files before parsing by hand-written logic, and added and now rely on optimized dependency sets during resolution (allowing to ignore dependencies whose transitive dependencies are all brought by another more general dependency).

Overall these optimizations bring some 30 to 50% speedup on some resolutions.

Miscellaneous

- support progress bars in the Windows console
- strict conflict manager from sbt
- fixes to support the `evicted` task of sbt
- take the `dependencyOverride` key into account (missing in some of the sbt integrations, not in the original sbt-coursier plugin)

MOOCs

@julienrf

The existing courses (on Coursera and edX) have all been updated to Scala 2.13 (or Scala 2.12 for the ones that use Spark) and sbt 1.x. We plan to deploy these updated versions during September.

We have been working on the “Functional Program Design” course to make its curriculum more streamlined (the current version is made of parts of previous versions of other courses, but everything is not consistent together). In particular, we have been replacing the content teaching `Future` with new content teaching implicits. (`Future` is covered both by our “Parallel Programming” and “Reactive Programming” courses)

We have also been working on the flagship course, “Functional Programming Principles”, to update its content for Dotty. In particular we have introduced enums early in the curriculum.

We plan to live-test the new content of these courses this semester with EPFL students before deploying it on our online learning platforms.

TASTy Reader for Scala 2 (SCP-018, start of the project)

@bishabosha

As the project started 2 weeks ago, we report on the general design direction at this point.

A key motivation for SCP-018 is to have a smooth story when migrating the ecosystem to Scala 3, so that we may finally unlock forwards binary compatibility. Ideally, projects will gradually migrate to the new compiler, and projects built with Scala 2 will still benefit from new updates from those that have migrated. One side of this story is already known: Scala 3 projects can depend on Scala 2 binaries, as `dotc` can unpickle `ScalaSignature` annotations. The reverse is not true at present. `dotc` does not generate `ScalaSignature` annotations, and `scalac` has no infrastructure to read TASTy.

To address this, we have started the project TASTy Reader For Scala 2, in which we add a frontend to parse TASTy files and enter signatures into the symbol table of `scalac`, allowing Scala 2 projects to depend on libraries compiled with Scala 3.

With both Scala 3 able to read Scala 2 signatures, and Scala 2 able to read Scala 3's TASTy, the ecosystem will be able to migrate from 2 to 3 one module at a time, in any order.

Metals

@olafurpg, VirtusLab

We released version 0.7.0 then version 0.7.2, with the following highlights:

- New tree view in VS Code
- Support for Scala 2.13.0 and 2.12.9
- Support for JDK 11
- Improved classpath indexing performance
- Bug fixes for importing builds in Gradle, Mill and sbt
- A lot of miscellaneous fixes

Bloop

@jvican

Preparation for v1.4.0, which we plan to release on September 26th. This version will feature:

- Automatic offloading of the compilation from sbt.
- Support for Semanticdb and Metals, to avoid Metals build tools integrations.
- Partial support for debugging.
- Support for Metals test/run.

- A redesigned installation process for bloop:
    - No more runtime dependency on Python in bloop (the official facebook/nailgun script required Python).
    - Nailgun-based bloop client rewritten in Scala and available as both a GraalVM binary and library.
    - Improvements in the launcher to make integrations with bloop trivial for any tool.
- Build pipelining will be enabled by default. There will be a comprehensive performance analysis.

We also gave a talk at Scala World 2019 about bloop v1.3.2's build server semantics and how it improves the state-of-the-art of build tools: "Design challenges of Bloop: a fast, concurrent build server".

Scala Libraries and Documentation

@julienrf

We have been actively reviewing or contributing to various Scala repositories:

- Improved rendering of documentation (#1437, #1440),
- Reviewed Alvin Alexander tutorial #1469,
- Reviewed contributions to scala-collection-contrib (#4, #18, #35, #43)
- Reviewed contributions to scala-collection-compat (#238, #247)

Scala Compiler

@julienrf, @bishabosha

We have been collecting problematic cases related to the use of implicits, causing developer frustrations. We have been exploring ways to improve the feedback given by the compiler. You can see the summary of our experiments in the contributors discussion.

Scala.js

@sjrd

We added support for ECMAScript 2020's dynamic `import()` calls.

We discovered and implemented a new optimization for instance tests against classes, which applies for `isInstanceOf` calls and more importantly pattern matching. The new optimization brings speedups of up to 40% for applications that intensively use pattern matching.

Finally, we rewrote most of the `java.util` collections so that they do not depend on Scala collections. The new implementations are faster, as they do not pile up logic to adapt the

semantics of Scala collections to those of the JDK, but instead implement the correct semantics from the start.

All those improvements will ship with Scala.js 0.6.29, which will be released within the next couple of weeks.

Scala Days 2019 organisation

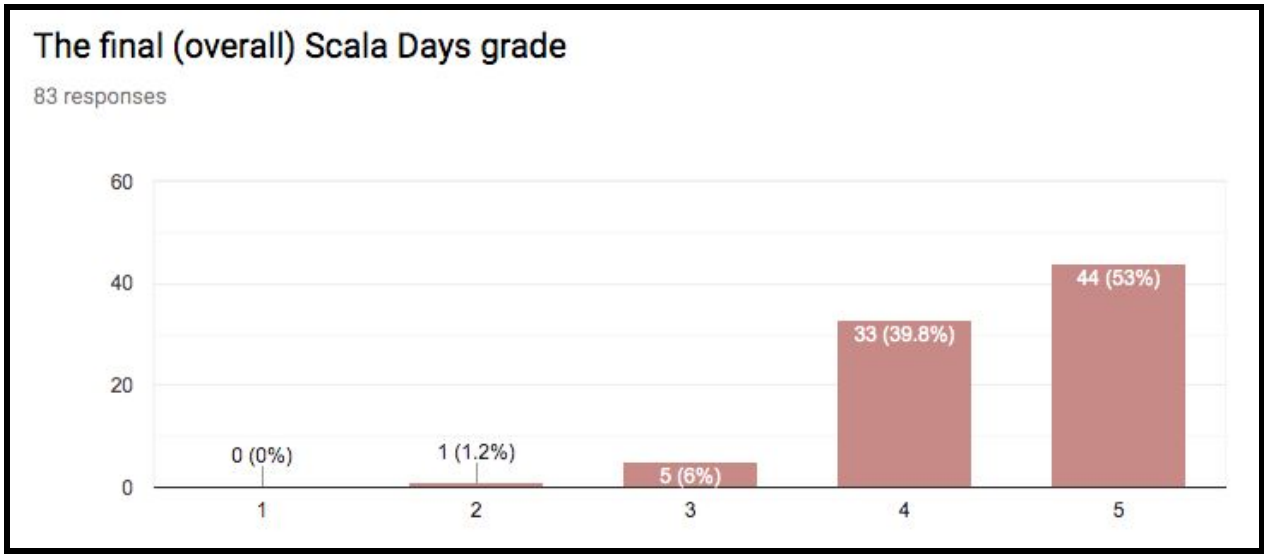
@darjutak

We finalized the organization and execution of Scala Days 2019 as planned in the report of June 2019.

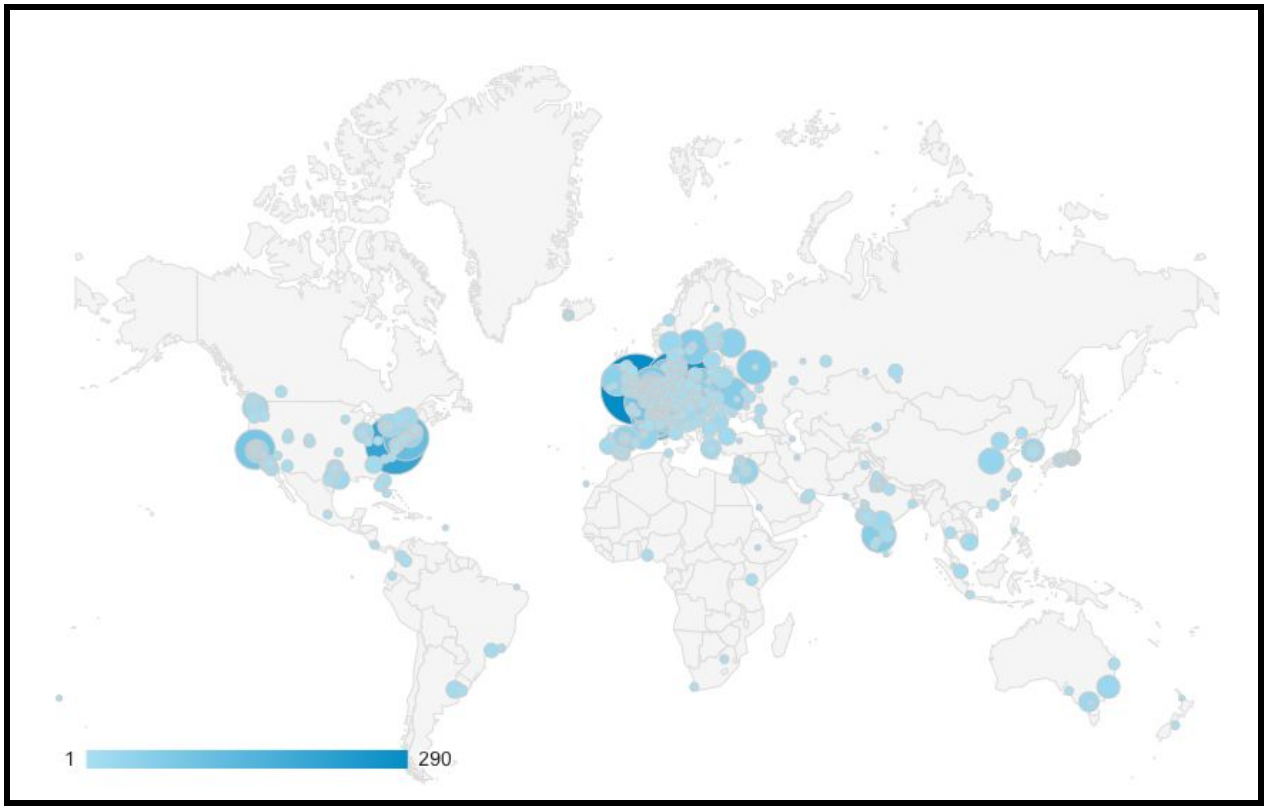
Beside the main goal of celebrating Scala's many anniversaries, bringing the community together for the conference as well as for the surrounding events, we as the Scala Center succeeded in:

- Bringing visibility to the Scala Center and its projects
- Increasing our presence in Switzerland (2 sprees Dec-May 2019, signed 7 Swiss company sponsors out of 22 potential)
- Establishing relationships with potential international supporters (sponsors, Advisory Board members)
- Bringing together the sub-community of contributors and organisers and establishing stronger relationships with them

We got an excellent feedback, based on this form. For example:



The final report (with financial data, audience, etc.) is ongoing. In the meantime, here is an extract of the communication outreach (cities) based on the Scala Days website analytics (Jan-June 2019):



SIP Meetings

@darjutak

A SIP meeting took place on June 8th, in person, at EPFL. We will restart the monthly meetings as of September.

Read the minutes here, taken by Dale Wijnand.

Communication and Community

@darjutak

- Moderator training proposal draft, adding for discussion (here)
- Announcement of team changes and other Scala Center updates (here)
- Help in organising the 3rd Contributors Summit at Scala Sphere, Krakow, Poland. We will be participating and leading discussions in October.