


Lichess & Scala 3

 **thibault** 

12 Dec 2022

 **266****27,492** views[Software Development](#) [Lichess](#)

Lichess gets a big upgrade. It doesn't go as planned.

Lichess is a [100% open-source/free-software](#) chess website, used by millions of players to play billions of games.

I made the choice to write it using [the scala language](#) many years ago, and never looked back. It has the features that matter to me:

- Type safety: the compiler as a copilot
- Functional programming: functions as building blocks
- Performance and ecosystem: the JVM as a strong foundation

Enters Scala 3

Lichess being built on Scala 2, when Scala 3 was released last year, I was very excited to upgrade. Yet I chose to wait a full year for the language to stabilize, and for the library ecosystem to catch up.

Last month I decided the wait was over, and that it was time for Lichess to get a massive upgrade. Could I have waited for longer? Sure, but I didn't see a reason why, and let's be honest I was craving for the latest features.

Scala 3 new features

It's not so much an evolution of the language, but rather a complete overhaul, as the compiler was **rewritten from scratch**. Yet compatibility was preserved wherever possible, easing the migration.

Here are some of my favourite [Scala 3 features](#):

Opaque types

Stronger typing with zero runtime cost, what's not to love? Strings (like user IDs) and other primitives can be replaced with proper types that the compiler understands.

```
opaque type UserId = String
```

```
def find(id: UserId) // this function doesn't accept any String, just UserId values
```

I actually found and fixed some old obscure bugs that were due to using `Strings`, while changing them to `opaque types`.

Cleaner syntax

Significant indentation and optional braces make our code look like python, which is cute. Fortunately the comparison stops here ;)

```
object Chess:
  def turnColor(ply: Int) =
    if ply % 2 == 1 then White else Black
```

Improved [type inference](#)

We want types, not boilerplate. Sometimes it's best to let the compiler figure out what things are by itself. I've been able to remove a lot of type annotations during the Scala 3 migration, and it pleased my code-golf inclinations.

Better [contextual abstractions](#)

`implicit` has been replaced with specialized keywords `using`, `given`, and `extension`. It makes the code easier to understand, as the intent is a lot clearer than when using the generic `implicit` keyword.

Enumerations that look good

More of a code sugar thing, but I always love concision and expressiveness:

```
enum DrawReason:
  case MutualAgreement, FiftyMoves, ThreefoldRepetition, InsufficientMaterial
```

It doesn't look like much, but it comes with [batteries included](#).

New `export` keyword

It works like `import`, but to expose functions and values. It makes composition more concise.

```
// before:  
def rating = glicko.rating  
def deviation = glicko.deviation  
  
// after:  
export glicko.{ rating, deviation }
```

If better composition means less inheritance, then count me in.

New `inline` keyword

While the former `@inline` annotation was a best-effort thing, the new `inline` keyword **guarantees inlining during compilation**.

It's a powerful tool that should be handled with care.

And many more features

The list could go on and on; there is [so much more to Scala 3](#)! This post is already getting too long, so I'll cut down on the fanboism.

The migration

[Lila](#) is a big program, serving 2000 HTTP requests per second, playing 5000 chess moves per second, while doing [A LOT](#) of [OTHER THINGS](#) that I better not start enumerating here.

So yeah, migrating it was scary, and I fully expected a disaster of some sort. Let's see how it went.

Tooling

Fortunately [metals](#) and [bloop](#) are handling Scala 3 very well, which gave my [code editor](#) full language support. It was a very comfortable experience.

All we need now is for some brave soul to [improve Scala 3 support for treesitter](#), so that we can all enjoy proper code coloring for the new language syntax.

Updating my code

That was the easy and fun part, especially since [the compiler did most of the work](#) for me. I actually rewrote more code than I had too, because I couldn't resist converting some `implicit`s to `given` here and there, and using `opaque` types.

At some point I had to rewrite the Glicko2 rating system [from Java to Scala 3](#), as the compiler was complaining about having Java in my project. No-one noticed broken ratings, so I suppose it worked.

Third-party libraries

That's where things got a bit hairy. Lila is built on [Play Framework](#) which is not yet ported to Scala 3.

So I [forked it and butchered it](#) to remove everything we don't need - which is actually most of the framework.

Once Play was reduced to a handful of small libraries (HTTP/netty server, routing, and forms), it became very easy to migrate to Scala 3.

Most [other dependencies](#), such as the [MongoDB driver](#), the [template engine](#) or the [cute functional cats](#), were already upgraded to Scala 3. As for the libraries coming from the Java ecosystem, like our [redis driver](#), well, they just work as usual.

Going to production

When everything compiled, **I shipped it**. And to everyone's surprise, apart from a few bugs I had created while rewriting thousands of lines of code... it worked. It just did. No explosions, no obscure

bugs, no memory leak, no performance degradation. That was rather unexpected.

With Scala 3 working in production, I was free to rewrite the code even deeper, to incrementally make use of Scala 3 features.

JVM tuning

Until one morning, instead of deploying the changes from the day before, I let it run an extra 24h. Then we saw the JVM CPU usage rise to alarming heights, with unusual patterns. And no obvious culprit in the thread dumps...

I couldn't make sense of it, and eventually called for help - read all about it in [my previous blog post](#).

The avengers assembled and saved Lichess: it was just the JVM that needed some tuning. The HotSpot compiler was running out of code cache, and once we gave it some more, things suddenly went a lot better.

Ludicrous speed

As a result, Lichess is now faster than it ever was. The previous version running Scala 2 was also throttled by the lack of JVM tuning. The effects were not as spectacular, but still: we were basically running Lichess with the parking brake on.

Lichess is now running with Scala 3 and without the parking brake, and it's a lot faster. To be able to tell if Scala 3 itself is faster, we would have to rollback to Scala 2 and try it with the proper JVM tuning. I'm not willing to do that, sorry! Once you've tried Scala 3, there's no going back.

Last words

I've been both scared and excited about this migration, ever since Scala 3 was announced. Given the size of our codebase, I was expecting a disaster, but instead all we got was a few bumps on the road. And we're now smooth sailing, all modern and future proof, which feels kinda great.

It only took a month to fully migrate Lichess, from the first code change to being certain that it runs in production perfectly (current uptime: 7 days).

As with every refactoring or migration, the success and speed are largely due to static typing and compiler quality.

Verdict

10/10 would migrate again

Thanks

Massive thanks go to the Scala 3 team, who did an incredible job at this new version of the language. Changes are [pretty darn well documented](#) too.

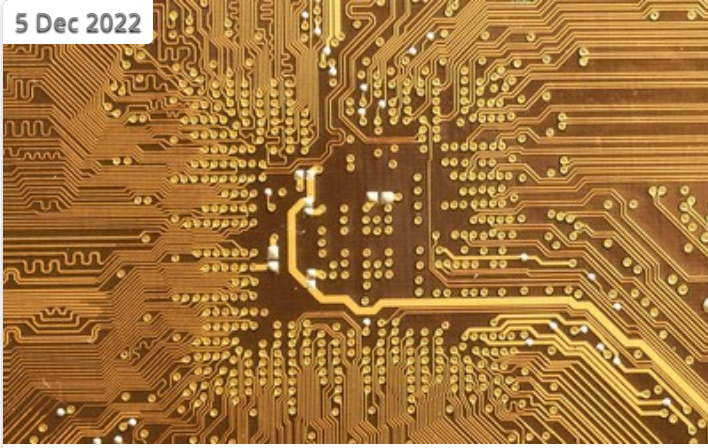
I'm also forever grateful to everyone who works on Scala tooling, and to the fantastic [community of developers](#) who helped me, guided me, and sometimes outright wrote the [complicated code](#) for me when I needed it.

Thank you to all Lichess players, and to everyone who [supports this beautiful project with a donation!](#)

 [DISCUSS THIS BLOG POST IN THE FORUM](#)

[More blog posts by thibault](#)

5 Dec 2022



Lichess on scala3 - help needed

A week after deploying Lichess rewritten with scala 3, I had to revert to the scala 2 version. I need...

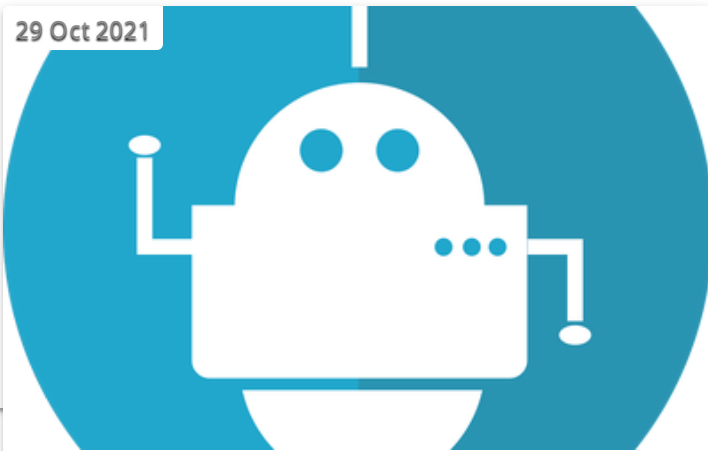
11 Aug 2022



Starting from scratch

If you had to start Lichess from scratch, what would you change?

29 Oct 2021



How to create a Lichess bot

Well, it depends. On you.

1 Oct 2021



How to ask technical questions

Simple guidelines to get quick and actionable answers