Cats Effect - IO Monad

E2E Application flow with IO

Agenda

- Effects & Side Effects
- Referential Transparency
- IO Monad from Cats Effect vs Future
- Http4s & Doobie
- E2E Application flow with IO from Cats Effect

Effects & Side Effects

- What are effects?
  - **Option[A]**: May or may not produce a value A
  - **Either[A, B]**: Either produces a value A or a value B
  - **List[A]**: Produces Zero, One or Many elements of type A
  - **IO[A]**: Produces a value A, fails or never terminates

- An **effect** is what the monad handles

Effects & Side Effects

- What are side effects?
    - **println(“Hello”)**: writes to the console **immediately**
    - **scala.io.StdIn.readLine()**: reads from the console **immediately**
    - **System.nanoTime()**: retrieves current time from the JVM **immediately**
    - **Future(deleteRecordsFromDb)**: deletes records from DB **immediately**

- Everything that is not reading the arguments and returning a result is a **Side Effect**
- **Side effects** are potential bugs

Referential Transparency

- Are these two blocks are the same?

```
val expr = 123
(expr, expr)
```

```
(123, 123)
```

Referential Transparency

- Are these two blocks are the same?

```
val expr = println(“Hello!”)
(expr, expr)
```

```
(println(“Hello!”), println(“Hello!”))
```

Referential Transparency

- Are these two blocks are the same?

```
import scala.concurrent.Future
import scala.concurrent.ExecutionContext.Implicits.global

val expr = Future(println(“Hello!”))
(expr, expr)
```

```
(Future(println(“Hello!”)), Future(println(“Hello!”)))
```

Referential Transparency

- Are these two blocks are the same?

```
import cats.effect.IO

val expr = IO(println(“Hello!”))
(expr, expr)
```

```
(IO(println(“Hello!”)), IO(println(“Hello!”)))
```

REFERENTIAL TRANSPARENCY (RT)

IO[A]

Represents the **intention** to perform a side effect

RT means that instead of function call you can simply put value and the program won’t break

‘’*To understand a program part (a function) you need no longer account for the possible **histories** of executions that can lead to that program part”*
© Martin Odersky

IO[A]

IO[A] describes a computation that will:

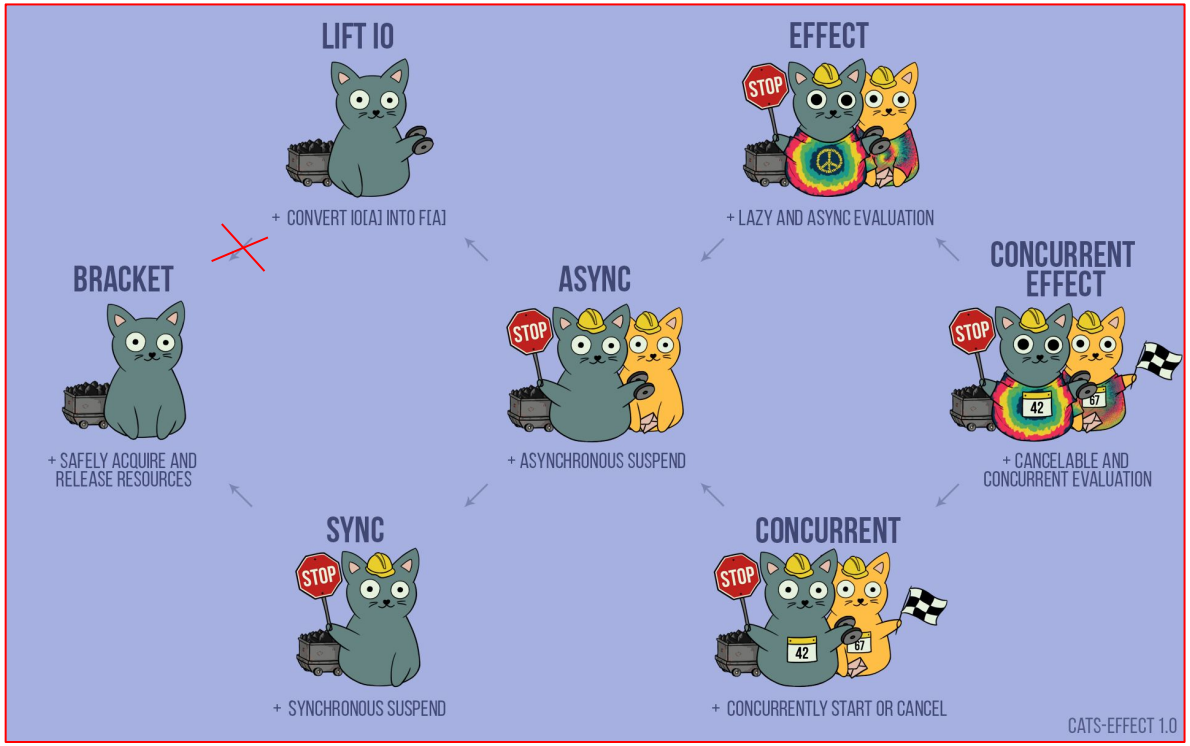
- Eventually produce a value of A, or
- Fail with a throwable, or
- Never complete

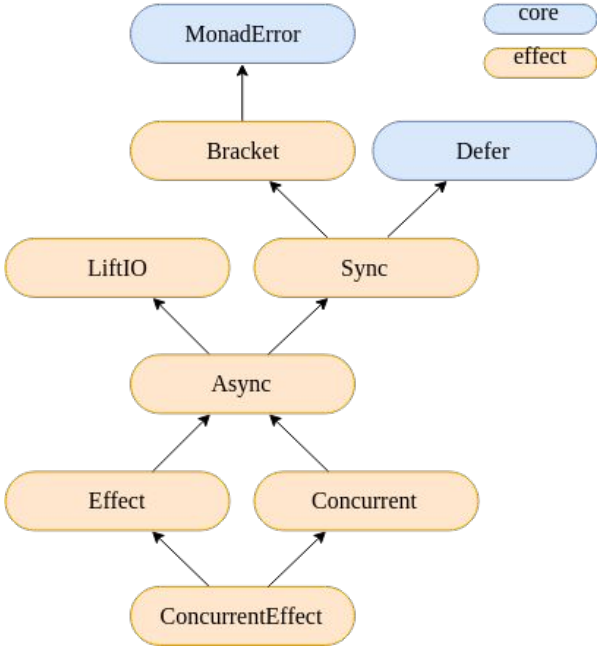
Cats-effect: https://typelevel.org/cats-effect/

implementation of **IO[A]**: https://typelevel.org/cats-effect/docs/2.x/datatypes/io

All the capabilities that IO exposes are described by **type classes**, allowing more generic code and multi-library compatibility

Cats-effect core typeclasses





IO vs Future - Overview

Actions:

- **IO** - a value that **describes** an action (possibly asynchronous)
- **Future** - a **handle** to the result of an already-running action (possibly asynchronous)

Performance:

- **IO** is optimized for throughput
  - Thread shift on demand
  - Has utilities for introducing manual shifts for fairness
  - Benchmarks faster for most workloads

- **Future** is optimized for fairness
  - thread shift every single map/flatMap (hence implicit *EC* - Execution Context)
  - Can only be configured using a specialized *EC* argument

IO vs Future - Overview

Cancellation:

- **Future[A]** can’t be cancelled - once constructed, it can’t be stopped
  - Wasted resources
- **IO[A]** can be concurrently forked, and then either *joined* or *canceled*
  - There are high-level constructs around this in cats-effect
  - More sophisticated abstractions build on top can be found in other libraries

Libraries for e2e application flow

- **http4s** - http layer (https://http4s.org/)
- **doobie** - persistence layer (https://tpolecat.github.io/doobie/)

- **cats core** - abstractions for FP (https://github.com/typelevel/cats)
- **cats effects -** pure asynchronous runtime (https://typelevel.org/cats-effect/)

- **circe** - json conversion (https://circe.github.io/circe/)
- **chimney** - dto mapping (https://github.com/scalalandio/chimney)

Repo with code

https://github.com/gleb-streltsov-4by/roulette