Scala 3, Here I Come!

張瑋修 Walter Chang

@weihsiu / weihsiu@gmail.com

Scala Taiwan

Scala Taiwan gitter channel

Scala Taiwan FB group

Scala Taiwan meetup

Agenda

- What is Scala 3?
- New syntax
- Select.scala
- Calc.scala
- Actor.scala
- Conversions.scala
- Cupcakes.scala
- Kvs.scala
- Q&A

What is Scala 3?

- Next generation of Scala
- Coming out Fall 2020
- Many new features and cleanups
- Source code is mostly backward compatible with Scala 2
- Libraries from both Scala 2/3 can be used interchangeably
- VSCode with Dotty Language Server plugin

New syntax

- Optional
- **New Control Syntax**

```
if x < 0 then y else z
while x > 0 do ???
for x <- xs do println(x)
```

- **Optional Braces**
  - No more curly braces (not really)
  - Just indent the block that normally goes inside curly braces
- Compiler switches allow to go back and forth

Select.scala

- The ability to select something
- **Anonymous Given Instances**

```scala
given Select[List[Int], Int] { ??? } // implicit val in Scala 2
given [A]: Select[List[A], A] { ??? } // implicit def in Scala 2
```

- **Extension Methods**

```scala
def (x: A) select (selector: B): Option[Out]
```

- **Main Functions**

```scala
@main def testSelect() = ???
```

- **Given Imports**

```scala
import hereicome.Select.given
```

Calc.scala

- Typed length calculations
- **Opaque Types Aliases**

```
opaque type Millimeters = Double
```

- **Given Instances with Collective Parameters**

```
given (n: Long) {
  def millimeters: Millimeters = n
  def centimeters: Centimeters = n
}
```

- **Inline Definitions**

```
inline def +[B : Units](y: B): Meters = x.toMeters + y.toMeters
```

Actor.scala

- A minimum actor implementation taken shamelessly from @li_haoyi
- **Union Types**

```
new Actor[Int | String] { ??? }
```

Conversions.scala

- A new way to define implicit conversion
- **Implicit Conversions**

```
given [A, B]: Conversion[Either[A, B], A | B] = ???
// abstract class Conversion[-T, +U] extends Function1[T, U]
```

Cupcakes.scala

- A minimum DI framework
- New take on the infamous Cake Pattern in Scala 2 ;)
- **Implicit Function Types** (as return value)

```
def write(data: String): (given FileService) => Unit
```

- **summon** aka. "the" or a better "implicitly"

```
summon[FileService].write(data)
```

- **Trait Parameters**

```
trait LogService(val prefix: String)
```

Cupcakes.scala (Cont.)

- Intersection Types

```
type AllServices = FileService & DatabaseService & NetworkService & LogService
```

- Export Clauses

```
export ctx._
```

Kvs1.scala

- A simple key-value store
- Typeclass Oriented Programming (TOP)
  - A better way to organize program
  - A solution to the Expression problem
- **Parameter Untupling**

```
ps.foreach((k, v) => x.del(k))
```

Kvs2.scala

- Typed key-value store
- **Named Type Arguments**

```
simpleKvs.getT[V = String]("hello")
```

Serde.scala

- Serialization / deserialization

Kvs3.scala

- Abstract effects
  - Identity effect
  - Network IO effect
- Kvs network client
- **Alias Givens**

```
given ioContextShift: ContextShift[IO] = IO.contextShift(executorService)
```

Protocol.scala

- Enums for network protocol Command and Reply
- **Enums**

```scala
enum Command
  case Put, Get, Del
```

NetIO.scala

- Socket IO effect
    - Synchronous IO (Blocker thread pool)
- **Creator Applications**

```
DataInputStream(socket.getInputStream)
```

KvsServer.scala

- Kvs network server
- Concurrency safe (Fiber, MVar)

Q&A

That's all and thank you for your attention