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
- Conversions.scala
- Actor.scala
- Door.scala
- It.scala
- Cupcakes.scala
- Kvs.scala
- Q&A

What is Scala 3?

- Next generation
- 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)
```

- **Significant Indentation**
  - No more curly braces (not really)
- Compiler switches allow to go back and forth

Select.scala

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

```
given Select[List[Int], Int] { ??? }
given [A]: Select[List[A], A] { ??? }
```

- **Extension Methods**

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

- **Main Functions**

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

- **Given Imports**

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

Select2.scala

- On top of the ability to select something, abstract how the selection is done.
- **Opaque Type Aliases**

```
opaque type RandomInt = (Int, Int)
object RandomInt
  def apply(start: Int, end: Int): RandomInt = (start, end)
```

- **Given Parameters**

```
def (x: A) select (selector: B)(given Selector[B, C]): Option[Out]
```

Conversions.scala

- The new and safer way to define implicit conversion
- **Implicit Conversions**

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

Actor.scala

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

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

Door.scala

- Translated from an Idris example
- **Enumerations**

```
enum DoorState
  case IsOpen()
  case IsClosed()
```

- **Dependent Function Types**

```
(c: DoorCommand[S]) => c.NextState
```

- **Toplevel Definitions**
  - `type`, `val`, `var`, and `def` can be defined without the enclosing `trait`, `class`, or `object`

It.scala

- Mimics Kotlin's "it"
- **Implicit Function Types** (as parameter)

```
def it[A](given p: GivenParameter[A]): A = p
```

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"

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

- **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")
```

Kvs3.scala

- Abstract effects
  - Identity effect
  - Network IO effect
- **Alias Givens**

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

NetIO.scala

- Socket IO effect
- **Creator Applications**

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

Serde.scala

- Serialization / deserialization

Q&A

That's all and thank you for your attention