

SERLi

Introduction au web



Mathieu ANCELIN
@TrevorReznik



Mathieu ANCELIN

- Développeur @SERLI
- Scala, Java, web & OSS
 - ReactiveCouchbase, Weld-OSGi, Weld, etc ...
 - Poitou-Charentes JUG
- Membre de l'expert group CDI 1.1 (JSR-346)
- Membre de l'expert group OSGi Enterprise
- @TrevorReznik





SERLI

- Société de conseil et d'ingénierie du SI
- 75 personnes
- 80% de business Java
- Contribution à des projets OSS
- 10% de la force de travail sur l'OSS
- Membre de l'EG JSR-346
- Membre de l'OSGi Alliance
- www.serli.com @SerliFr



SERLI



```
var param1 = "param1"  
val param2 = "param2"
```

```
def doSomething(param: String = "default") {  
    println(s"value = $param")  
}
```

```
def process(s: String): String = s.toLowerCase
```

```
list.forEach { item =>  
    item.toUpperCase  
}
```

```
list.forEach(_.toLowerCase)  
list.forEach(process)
```




String interpolation

```
val name = "World"
```

```
println(s"hello $name!")
```

```
println(s"hello ${name.toUpperCase}!")
```



Traits

```
trait AuthService {  
    def isConnected(username: String): Boolean  
  
    def hash(username: String) = MD5.sign(username)  
}  
  
class MyService extends AuthService {  
    override def isConnected(username: String): Boolean = true  
}
```



Classes

```
class Post(id: String, title: String) {  
    val innerContext = "Nouveau post"  
    def display() = s"$title \n $innerContext"  
}
```



Case class

```
case class User(id: String, name: String, email: String, age: Int)
```

```
object User {  
  def findByEmail(email: String) = ???  
  def birthday(u: User) = u.copy(age = u.age + 1)  
}
```

```
object UserController {  
  def getUrl(email: String) = User.findByEmail(email)  
}
```




Fonctions

```
def doSomething(param: String) {  
  println(s"value = $param")  
}
```

```
def process(s: String): String = s.toLowerCase
```

```
list.forEach { item =>  
  item.toUpperCase  
}
```

```
list.forEach(_.toLowerCase)  
list.forEach(process)
```

```
def doSomethingElse(param: String): String = {  
  s"value = $param"  
}
```

```
def doSomethingElse(param: String): String = s"value = $param"
```



Fonctions

```
object Hello {  
  def apply(name: String) = println(s"Hello $name!")  
  
  def doSomething(f: String => String) = f("Hello")  
  
  def /?\(name: String) = apply(name)  
}
```

```
Hello("World")  
Hello.doSomething(p => p.toUpperCase)  
Hello.doSomething { p =>  
  p.toLowerCase  
}
```

```
Hello./?\("World")  
Hello /?\ "World"
```



Curryfication

```
def add(v1: Int)(v2: Int) = v1 + v2
```

```
val value1 = add(2)(2) // 4
```

```
val add4 = add(4)_
```

```
val val2 = add4(2) // 6
```



Pattern matching

```
val param = ???
```

```
param match {  
  case "Hello"  
  case 1  
  case t: Task  
  case t: Task if t.id == 1  
  case "a" :: "b" :: tail  
  case head :: "b" :: "c" :: tail  
  case User(id, name, email, age)  
  case User(_, _, _, age) if age > 18  
  case user @ User(_, _, _, age) if age > 18  
  case t @ Task(_, _, User(_, _, _, age)) if age < 100  
}  
  
=> println("Yeah !")  
=> println("Yeah !")  
=> println(t)  
=> println(t)  
=> println(tail)  
=> println(head)  
=> println(s"$id : $name : $email : $age")  
=> println(age)  
=> println(user)  
=> println(t)
```

```
Hello.doSomething {  
  case "Hello" => println("Hello World")  
  case _ => println("Yeah !")  
}
```



implicit

```
def adder(a: Int)(implicit b: Int) = a + b
```

```
adder(2)(2) // 4
```

```
implicit val implicitParam = 6
```

```
adder(2) // 8
```


Structures du SDK





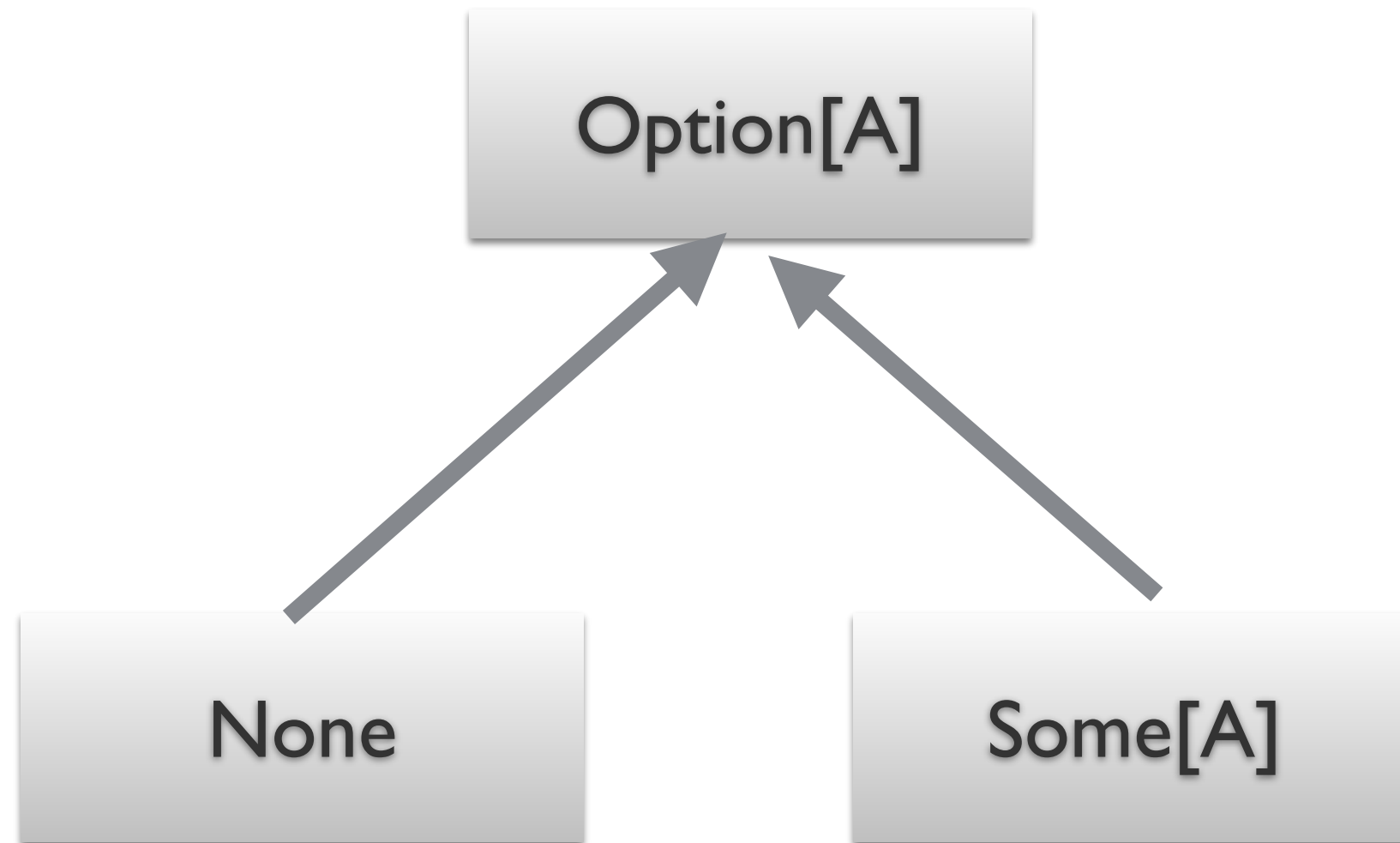
Collections

```
val coll: Seq[String] =  
    Seq("a", "b", "c", "d", "e", "f", "g")
```

```
val str: String = coll  
    .filter(_.length == 1)  
    .map(_.toUpperCase)  
    .reduce(_ + _)           // ABCDEFG
```



Option





Option

```
Option[A] map = (f: A => B): Option[B]
```

```
Option[A] flatMap = (f: A => Option[B]): Option[B]
```

```
Option[A] filter = (f: A => Boolean): Option[A]
```



Option

```
process(Option("Hello"))  
process(None)
```

```
def process(opt: Option[String]) = {  
  opt match {  
    case Some(str) => println(str)  
    case None => println("No value here")  
  }  
  opt.map(_.toUpperCase)  
    .filter(_.length >= 5)  
    .map(_ + "World")  
    .getOrElse(":(") // Some("HELLO World")  
}
```




Try

```
import scala.util._

val proc1 = Try(process(Option("yeah")))
val proc2 = Try {
  throw new RuntimeException("Error !!!!")
}

def handleTry(t: Try[Option[String]]) = {
  t match {
    case Success(Some(str)) => println(str)
    case Success(None) => println("Success but nothing found")
    case Failure(err) => println(err.getMessage)
  }
}

handleTry(proc1)
handleTry(proc2)
```



Future

```
implicit val ec = ExecutionContext.fromExecutor(Executors.newFixedThreadPool(5))

val future1 = Future {
  Thread.sleep(10000)
  "I'm done 1!"
}
val future2 = Future {
  Thread.sleep(20000)
  "I'm done 2!"
}

future1.map { message =>
  println(message)
  message
}.onComplete {
  case Success(message) => println(s"success of : $message")
  case Failure(err) => println(s"error : ${err.getMessage}")
}
```



Future

```
implicit val ec = ExecutionContext.fromExecutor(Executors.newFixedThreadPool(5))
```

```
val future1 = Future {  
  Thread.sleep(10000)  
  "I'm done 1!"  
}
```

```
val future2 = Future {  
  Thread.sleep(20000)  
  "I'm done 2!"  
}
```

```
future1.flatMap { message1 =>  
  future2.map { message2 =>  
    s"messages : $message1, $message2"  
  }  
}.onComplete {  
  case Success(message) => println(message) // messages: I'm done 1!, I'm done 2!  
  case Failure(err) => println(err.getMessage)  
}
```

This is the end ...

des questions ?

