SERLI

Play 2

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

- 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

Historique

- Annoncé en novembre 2011 à Devoxx
  - Guillaume Bort rejoint le board Typesafe
- Réécriture complète du framework ‘from scratch’
- La version 2.0 sort en Avril 2012
- Play devient la stack web de typesafe

Welcome Scala

- Framework entièrement réécrit en Scala
- Entièrement basé sur des notions d'asynchronisme et de non bloquant
- Fournit une API Java complète avec le même niveau de fonctionnalités que l'API Scala

SBT

- Simple Build Tool
- Outil de build standard de facto pour Scala
  - Gère le versioning des librairies
  - Gère les dépendencies
  - Gère la compilation et le packaging

Akka

- Librairie Scala très populaire
- Paradigme de programmation orienté ‘acteurs’
- Bien plus poussé que les acteurs Scala fournis par défaut
  - devient le standard dans Scala 2.10
- Enormes capacités pour traitements distribués et concurrents

Routes

```
GET    /clients/all              controllers.Clients.list()
GET    /clients/:id              controllers.Clients.show(id: Long)

GET    /files/*name              controllers.Application.download(name)

GET    /clients/$id<[0-9]+>      controllers.Clients.show(id: Long)


GET    /                         controllers.Application.show(page)


GET    /                         controllers.Application.show(page= home")
GET    /:page                    controllers.Application.show(page)

GET    /clients                  controllers.Clients.list(page:Int?=1)
```

Reverse Routing

```
package controllers
import play.api._
import play.api.mvc._
import javax.inject._

@Singleton
class Application extends Controller {
  def hello(name: String) = Action {
    Ok("Hello " + name + "!")
  }
  // Redirect to /hello/Bob
  def helloBob = Action {
    Redirect(routes.Application.hello("Bob"))
  }
}
# Hello action
GET    /hello/:name            controllers.Application.hello(name)
```

Contrôleurs

```
package controllers
import play.api.mvc._
import javax.inject._

@Singleton
class Application extends Controller {

  def index = Action {
    Ok("It works!")
  }
  def hello(name: String) = Action {
    Ok("Hello" + name + "!")
  }
}
```

Contrôleurs

```
package controllers

import play.api.mvc._
import javax.inject._

@Singleton
class Application extends Controller {
 def index = Action {
    Redirect( « http://www.google.fr » )
  }
}
```

Contrôleurs

```
package controllers

import play.api.mvc._
import javax.inject._

@Singleton
class Application extends Controller {
  def index = Action {
    NotFound
  }
  def error = Action {
    InternalServerError("Oops")
  }
}
```

Body parsers

```
package controllers

import play.api._
import play.api.mvc._
import javax.inject._
import play.api.mvc.BodyParsers.parse

@Singleton
class Application extends Controller {
  def save = Action(parse.text) { request =>
    Ok("Got: " + request.body)
  }
  def saveJson = Action(parse.json) { request =>
    Ok(request.json)
  }
}
```

Asynchrone

- Le framework est complètement asynchrone by design
- Possibilité de renvoyer des résultats asynchrones depuis les contrôleurs
  - utile pour les traitements long
  - ne bloque pas les ressources

Asynchrone

```
package controllers
import play.api.mvc._
import javax.inject._

@Singleton
class Application @Inject()()(implicit ec: ExecutionContext)
                                          extends Controller {
  def index = Action.async {
    val bob = Customer("Bob")
    val promiseOfOrders = Future { bob.orders() }
    promiseOfOrders.map { orders =>
      Ok(views.html.index(bob, orders))
    }
  }
}
```

Vues

- Vues également écrites en Scala
- Vues typesafe
  - il faut déclarer les paramètres de la vue
  - la vue est compilée

```
views/Application/index.scala.html
```

```
views.html.Application.index()
```

Vues

```
@(customer: Customer, orders: Seq[Order])

<h1>Welcome @customer.name!</h1>

<ul>
  @orders.map { order =>
    <li>@order.title</li>
  }
@for(order <- orders) {
    <li>@order.title</li>
}
</ul>
```

paramètres du templates (typés)

utilisation d'un paramètre

expressions scala

Contrôleurs + vues

```
package controllers

import play.api.mvc._
import javax.inject._

@Singleton
class Application extends Controller {

  def index = Action {
    val bob = Customer("Bob")
    Ok(views.html.index(bob, bob.orders()))
  }
}
```

Forms

```
import play.api.data._
import play.api.data.Forms._

case class UserData(name: String, age: Int)

val userForm = Form(
  mapping(
    "name" -> text,
    "age" -> number
  )(UserData.apply)(UserData.unapply)
)

val userData = userForm.bindFromRequest.get
```

Forms

```
userForm.bindFromRequest.fold(
  formWithErrors => {
    BadRequest(views.html.user(formWithErrors))
  },
  userData => {
    val newUser = models.User(userData.name, userData.age)
    val id = models.User.create(newUser)
    Redirect(routes.Application.home(id))
  }
)
```

Forms

```
import play.api.data._
import play.api.data.Forms._

case class UserData(name: String, age: Int)

val userFormConstraints2 = Form(
  mapping(
    "name" -> nonEmptyText,
    "age" -> number(min = 0, max = 100)
  )(UserData.apply)(UserData.unapply)
)

val boundForm = userFormConstraints2.bind(Map("bob" -> "", "age" -> "25"))
boundForm.hasErrors must beTrue
```

Forms

```
def index = Action {
  Ok(views.html.user(userForm))
}
```

```
@import helper._

@helper.form(action = routes.Application.userPost()) {
  @helper.inputText(userForm("name"))
  @helper.inputText(userForm("age"))
}
```

Ebean vs. Anorm

- Deux philosophies d'accès aux données
  - EBean => Java
    - implémentation stateless de JPA
  - Anorm => Scala
    - Wrapper JDBC avec beaucoup d'aide pour mapper les resultsets

Anorm

- Anorm is Not an Object Relationnal Mapper
- Wrapper au dessus de JDBC
  - pas d'ORM
- API scala
  - plus de problèmes liés à la structure de l'API Java
  - pas d'exceptions à gérer

Anorm

```
import anorm._
import play.api.db._

class Application @Inject()(db: Database) extends Controller

db.withConnection { implicit c =>
  val result1: Boolean = SQL("Select 1").execute()
  val result2: Int =
    SQL("delete from City where id = 99").executeUpdate()
  val id: Option[Long] =
    SQL("insert into City(name, country) values ({name}, {country})")
     .on('name -> "Cambridge", 'country -> "New Zealand »)
     .executeInsert()
}
```

Anorm

```
val code: String = SQL(
  """
    select * from Country c
    join CountryLanguage l on l.CountryCode = c.Code
    where c.code = {countryCode}
  """)
  .on("countryCode" -> "FRA").as(SqlParser.str("code").single)
```

```
val lang = "French"
val population = 10000000
val margin = 500000

val code: String = SQL"""
  select * from Country c
    join CountryLanguage l on l.CountryCode = c.Code
    where l.Language = $lang and c.Population >= ${population - margin}
    order by c.Population desc limit 1"""
  .as(SqlParser.str("Country.code").single)
```

Anorm

```
import anorm.SqlParser._

case class Language(name: String, language: String, official: Boolean)

val languageParser = str("name") ~ str("language") ~ str("isOfficial") map {
  case name ~ language ~ "T" => Language(name, language, true)
  case name ~ language ~ "F" => Language(name, language, false)
}

def spokenLanguages(countryCode: String): List[Language] = {
  SQL(
    """
      select * from Country c
      join CountryLanguage l on l.CountryCode = c.Code
      where c.code = {code};
    """
  ).on("code" -> countryCode).as(languageParser.*)
}
```

Anorm

```
import anorm.SqlParser._

case class Language(name: String, language: String, official: Boolean)
val languageParser = get[String]("name") ~ get[String]("language") ~
                                          get[String]("isOfficial") map {
  case name ~ language ~ "T" => Language(name, language, true)
  case name ~ language ~ "F" => Language(name, language, false)
}
def spokenLanguages(countryCode: String): List[Language] = {
  SQL(
    """
      select * from Country c
      join CountryLanguage l on l.CountryCode = c.Code
      where c.code = {code};
    """
  ).on("code" -> countryCode).as(languageParser *)
}
```

Anorm: Types

↓JDBC / JVM→	BigDecimal1	BigInteger2	Boolean	Byte	DoubleFloat	Int	Long	Short
BigDecimal1	Yes	Yes	No	No	Yes	No	Yes	Yes	No
BigInteger2	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No
Boolean	No	No	Yes	Yes	No	No	Yes	Yes	Yes
Byte	Yes	No	No	Yes	Yes	Yes	No	No	Yes
Double	Yes	No	No	No	Yes	No	No	No	No
Float	Yes	No	No	No	Yes	Yes	No	No	No
Int	Yes	Yes	No	No	Yes	Yes	Yes	Yes	No
Long	Yes	Yes	No	No	No	No	Yes	Yes	No
Short	Yes	No	No	Yes	Yes	Yes	No	No	Yes

SERLi

Json

```
import play.api.libs.json._

val json: JsValue = Json.parse("""
{
  "name" : "Watership Down",
  "location" : {
    "lat" : 51.235685,
    "long" : -1.309197
  },
  "residents" : [ {
    "name" : "Fiver",
    "age" : 4,
    "role" : null
  }, {
    "name" : "Bigwig",
    "age" : 6,
    "role" : "Owsla"
  } ]
}
""")
```

```
import play.api.libs.json._

val json: JsValue = Json.obj(
  "name" -> "Watership Down",
  "location" -> Json.obj(
     "lat" -> 51.235685, "long" -> -1.309197),
  "residents" -> Json.arr(
    Json.obj(
      "name" -> "Fiver",
      "age" -> 4,
      "role" -> JsNull
    ),
    Json.obj(
      "name" -> "Bigwig",
      "age" -> 6,
      "role" -> "Owsla"
    )
  )
)
```

Json

```
case class Location(lat: Double, long: Double)
case class Resident(name: String, age: Int, role: Option[String])
case class Place(name: String, location: Location, residents: Seq[Resident])
```

```
implicit val locationWrites = new Writes[Location] {
  def writes(location: Location) = Json.obj(
    "lat" -> location.lat,
    "long" -> location.long
  )
}

implicit val residentWrites = new Writes[Resident] {
  def writes(resident: Resident) = Json.obj(
    "name" -> resident.name,
    "age" -> resident.age,
    "role" -> resident.role
  )
}
```

```
implicit val placeWrites = new Writes[Place] {
  def writes(place: Place) = Json.obj(
    "name" -> place.name,
    "location" -> place.location,
    "residents" -> place.residents)
}

val place = Place(
  "Watership Down",
  Location(51.235685, -1.309197),
  Seq(
    Resident("Fiver", 4, None),
    Resident("Bigwig", 6, Some("Owsla"))
  )
)

val json = Json.toJson(place)
```

Json

```
val name = (json \ "name").as[String]
// "Watership Down"

val names = (json \\ "name").map(_.as[String])
// Seq("Watership Down", "Fiver", "Bigwig")

val nameOption = (json \ "name").asOpt[String]
// Some("Watership Down")

val bogusOption = (json \ "bogus").asOpt[String]
// None
```

Json

```scala
import play.api.libs.json._
import play.api.libs.functional.syntax._

implicit val locationReads: Reads[Location] = (
  (JsPath \ "lat").read[Double] and
  (JsPath \ "long").read[Double]
)(Location.apply _)

implicit val residentReads: Reads[Resident] = (
  (JsPath \ "name").read[String] and
  (JsPath \ "age").read[Int] and
  (JsPath \ "role").readNullable[String]
)(Resident.apply _)
```

```scala
import play.api.libs.json._
import play.api.libs.functional.syntax._

implicit val placeReads: Reads[Place] = (
  (JsPath \ "name").read[String] and
  (JsPath \ "location").read[Location] and
  (JsPath \ "residents").read[Seq[Resident]]
)(Place.apply _)
```

Json

```
val json = { ... }

val placeResult: JsResult[Place] = json.validate[Place]
placeResult match {
  case JsSuccess(place, _) => println("Place: " + place)
  case JsError(e) => println("Errors: " + JsError.toFlatJson(e).toString())
}
```

Web services

```
import play.api.libs.ws._
import scala.concurrent.Future

class Application @Inject()(ws: WSClient) extends Controller

val futureResponse: Future[WSResponse] = ws.url("http://www.google.fr/q")
    .withHeaders("Accept" -> "application/json")
    .withRequestTimeout(10000)
    .withQueryString("search" -> "play")
      .get()
```

Cache

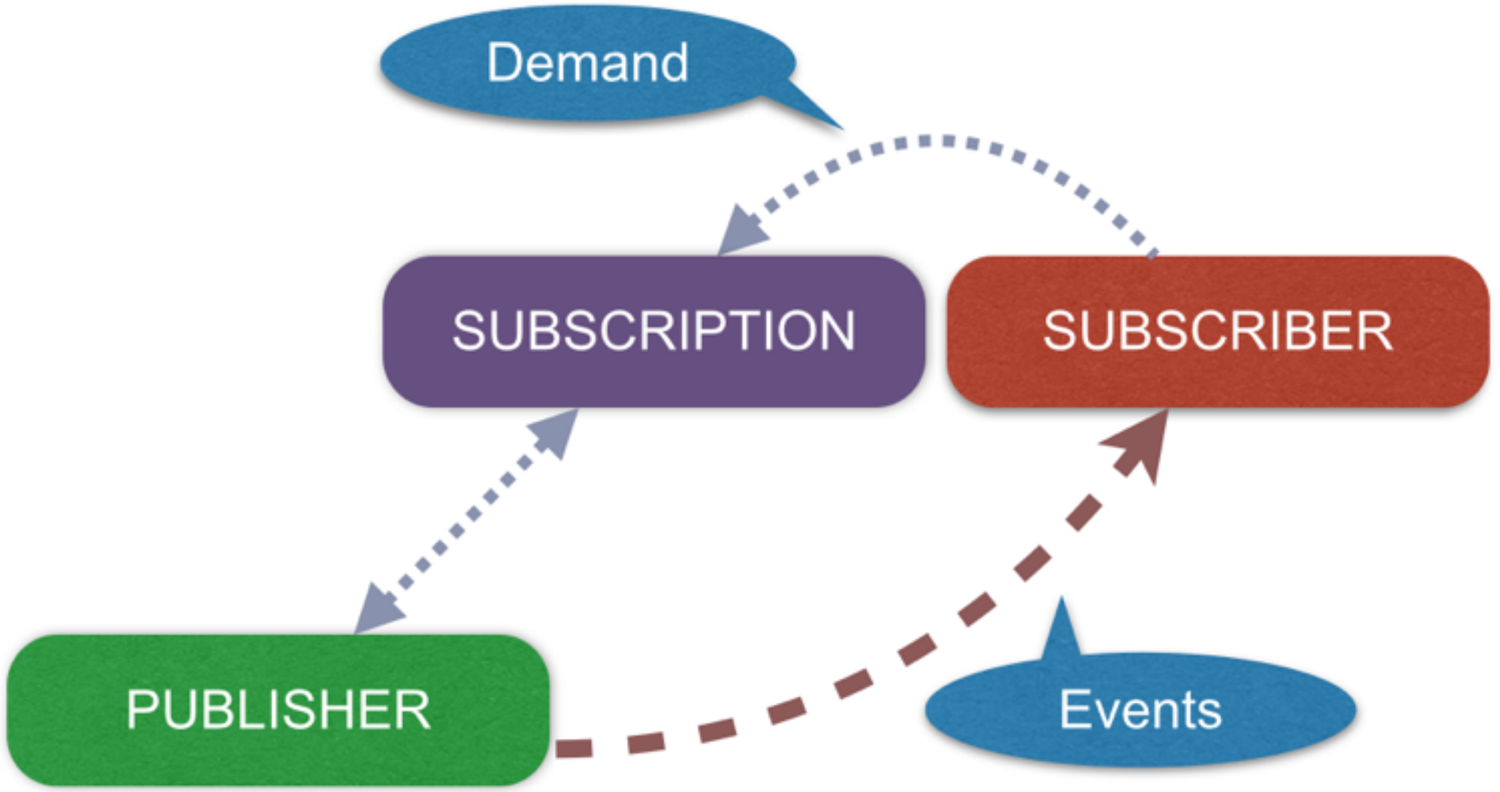
```
class Application @Inject()(cache: CacheApi) extends Controller

cache.set("item.key", connectedUser)
val maybeUser: Option[User] = cache.getAs[User]("item.key")
val user: User = cache.getOrElse[User]("item.key") {
  User.findById(connectedUser)
}
cache.remove("item.key")
```

Reactive Streams

Reactive Streams is an initiative to provide a standard for asynchronous stream processing with non-blocking back pressure.

This encompasses efforts aimed at runtime environments (JVM and JavaScript) as well as network protocols.

http://www.reactive-streams.org/


Demand
SUBSCRIPTION
SUBSCRIBER
PUBLISHER
Events

Exemple

```
def comet = Action {
    val events = Source(Seq("kiki", "foo", "bar"))
    Ok.chunked(events via
        Comet.string("parent.cometMessage")).as(ContentType.HTML)
}
```

```
<script type="text/javascript">
  var cometMessage = function(event) {
    $('#messages').append('Received: ' + event)
  }
</script>
<div id="messages"></div>
<iframe src="/comet"></iframe>
```

Server Sent Events

```
def feed = Action {
    val events = Source("kiki", "foo", "bar")
    Ok.chunked(events via EventSourced.flow)
                                .as("text/event-stream")
}
```

```
var feed = new EventSource('/feed');
feed.onmessage = function (e) {
    var data = JSON.parse(e.data);
    console.log(data);
}
```

WebSockets

```
import play.api.mvc._
import play.api.libs.streams._

class App @Inject() (implicit system: ActorSystem, materializer: Materializer)
                                                        extends Controller {

 def socket = WebSocket.accept[String, String] { request =>
    ActorFlow.actorRef(out => MyWebSocketActor.props(out))
  }
}
object MyWebSocketActor {
  def props(out: ActorRef) = Props(new MyWebSocketActor(out))
}
class MyWebSocketActor(out: ActorRef) extends Actor {
  def receive = {
    case msg: String =>
      out ! ("I received your message: " + msg)
  }
}
```

This is the end …

des questions ?