

Play Framework !



- Annoncé en novembre 2011 à Devovx
 - 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

Historique



- Framework entièrement ré-écrit 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

Welcome Scala



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

- 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

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)

Routes



```
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)
```

Reverse Routing



```
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")
  }
}
```

Contrôleurs



```
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)
  }
}
```

Body parsers



- 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))
    }
  }
}
```

Asynchrone



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

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

paramètres du templates (typés)

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

utilisation d'un paramètre

```
<ul>
```

```
  @orders.map { order =>
```

```
    <li>@order.title</li>
```

```
  }
```

```
  @for(order <- orders) {
```

```
    <li>@order.title</li>
```

```
  }
```

```
</ul>
```

expressions scala

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()))
  }
}
```

Contrôleurs + vues




```
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" ) )  
}
```

Forms



```
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"
    )
  )
)
```

Play 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)
```

Play 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
```

Play Json



```
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 _)
```

```
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 _)
```

Play 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())
}
```

Play Json



```
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()
```

Web services



```
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")
```

Cache



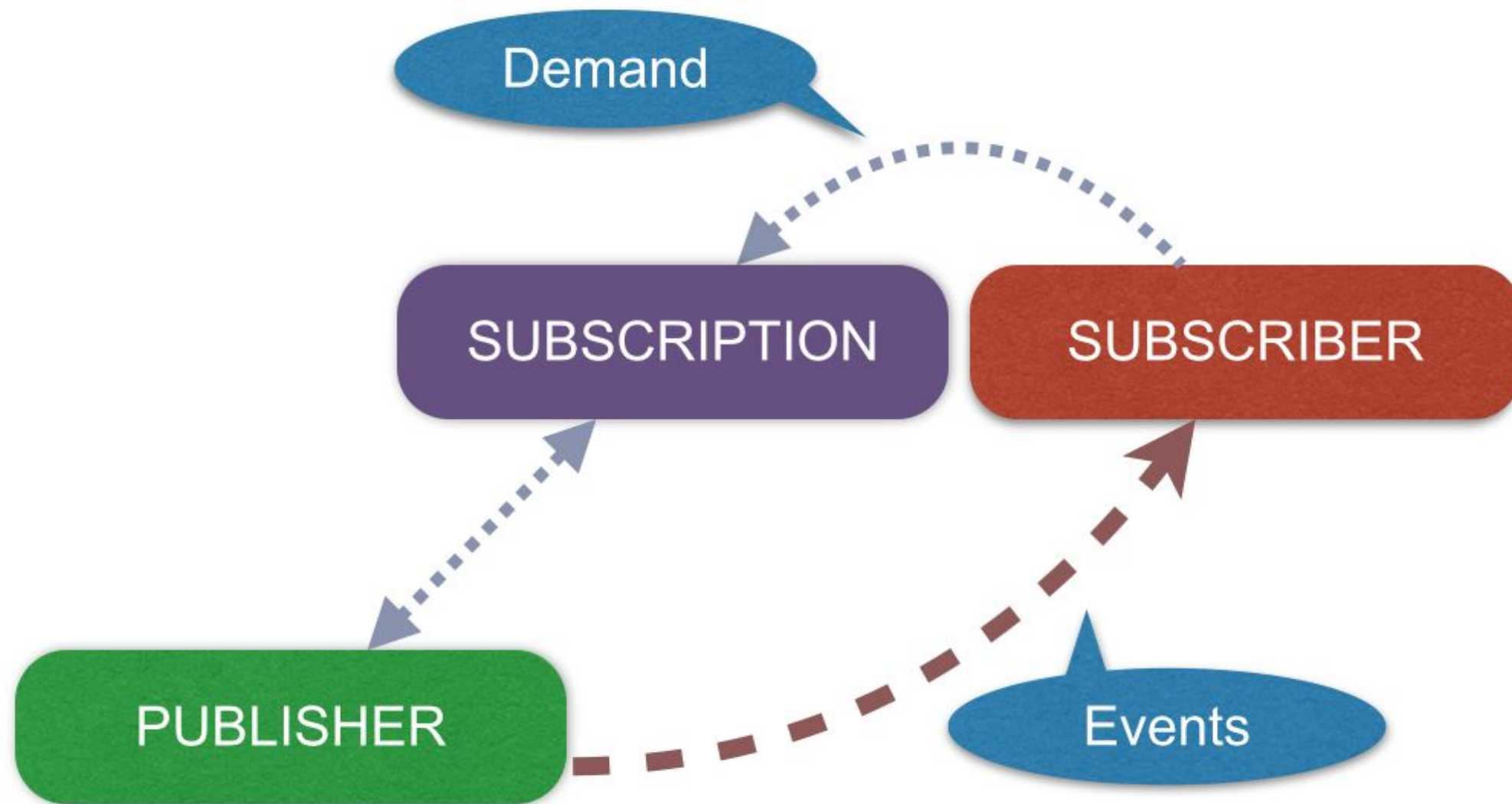
Reactive Streams is an initiative to provide a standard for asynchronous stream processing with non-blocking back pressure.

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

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

Reactive Streams





```
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>
```

Exemple



```
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);  
}
```

Server Sent Events



```
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)
  }
}
```

WebSockets

