Scala, un langage à échelle variable

Didier Villevalois

LIF – Équipe MoVe

LIF Tech – Jeudi 15 Octobre 2015

Scala ?

Issu de la recherche à EPFL
(Ecole Polytechnique Fédérale de Lausane)

Sous l'impulsion de Martin Odersky
Co-auteur de Generic Java [BOSW98]

Gestation dès 2001
Public depuis 2004 [Oa04]

Scala pour "Scalability"

Scala ?

Issu de la recherche à EPFL
(Ecole Polytechnique Fédérale de Lausane)

Sous l'impulsion de Martin Odersky
Co-auteur de Generic Java [BOSW98]

Gestation dès 2001
Public depuis 2004 [Oa04]

Scala pour "Scalability"

Scala ?

Issu de la recherche à EPFL
(Ecole Polytechnique Fédérale de Lausane)

Sous l'impulsion de Martin Odersky
Co-auteur de Generic Java [BOSW98]

Gestation dès 2001
Public depuis 2004 [Oa04]

Scala pour "Scalability"

Scala ?

Issu de la recherche à EPFL
(Ecole Polytechnique Fédérale de Lausane)

Sous l'impulsion de Martin Odersky
Co-auteur de Generic Java [BOSW98]

Gestation dès 2001
Public depuis 2004 [Oa04]

Scala pour "Scalability"

Du procédural au fonctionnel

- Classes mutables (`class`) et immuables (`case class`)
- Variables mutables (`var`) et immuables (`val`)
- Collections mutables et immuables
- Boucles impératives (`while`) et compréhensions (`for`)

Du procédural au fonctionnel

Classes mutables (`class`) et immuables (`case class`)

Variables mutables (`var`) et immuables (`val`)

Collections mutables et immuables

Boucles impératives (`while`) et compréhensions (`for`)

Du procédural au fonctionnel

Classes mutables (`class`) et immuables (`case class`)

Variables mutables (`var`) et immuables (`val`)

Collections mutables et immuables

Boucles impératives (`while`) et compréhensions (`for`)

Du procédural au fonctionnel

Classes mutables (`class`) et immuables (`case class`)

Variables mutables (`var`) et immuables (`val`)

Collections mutables et immuables

Boucles impératives (`while`) et compréhensions (`for`)

Du script à l'application d'entreprise

Read-Eval-Print-Loop

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> candidates.size
res0: Int = 8

scala>
```

Exécution sur Java Virtual Machine

- Interface graphique, base de donnée, ...
- Serveur Web, Big Data, Cloud, ...

Du script à l'application d'entreprise

Read-Eval-Print-Loop

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> candidates.size
res0: Int = 8

scala>
```

Exécution sur Java Virtual Machine

- Interface graphique, base de donnée, ...
- Serveur Web, Big Data, Cloud, ...

Du script à l'application d'entreprise

Read-Eval-Print-Loop

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> candidates.size
res0: Int = 8

scala>
```

Exécution sur Java Virtual Machine

- Interface graphique, base de donnée, ...
- Serveur Web, Big Data, Cloud, ...

Du script à l'application d'entreprise

Read-Eval-Print-Loop

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> candidates.size
res0: Int = 8

scala>
```

Exécution sur Java Virtual Machine

- Interface graphique, base de donnée, ...
- Serveur Web, Big Data, Cloud, ...

Du script à l'application d'entreprise

Read-Eval-Print-Loop

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> candidates.size
res0: Int = 8

scala>
```

Exécution sur Java Virtual Machine

- Interface graphique, base de donnée, ...
- Serveur Web, Big Data, Cloud, ...

Du script à l'application d'entreprise

Read-Eval-Print-Loop

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> candidates.size
res0: Int = 8

scala>
```

Exécution sur Java Virtual Machine

Interface graphique, base de donnée, ...

Serveur Web, Big Data, Cloud, ...

Du script à l'application d'entreprise

Read-Eval-Print-Loop

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> candidates.size
res0: Int = 8

scala>
```

Exécution sur Java Virtual Machine

Interface graphique, base de donnée, ...
Serveur Web, Big Data, Cloud, ...

Du script à l'application d'entreprise

Read-Eval-Print-Loop

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> candidates.size
res0: Int = 8

scala>
```

Exécution sur Java Virtual Machine

- Interface graphique, base de donnée, ...
- Serveur Web, Big Data, Cloud, ...

Du script à l'application d'entreprise

Read-Eval-Print-Loop

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> candidates.size
res0: Int = 8

scala>
```

Exécution sur Java Virtual Machine

- Interface graphique, base de donnée, ...
- Serveur Web, Big Data, Cloud, ...

Syntaxe économique

Moins cérémonieuse que Java

Example (Java)

```
public class Point {
  private double x, y;

  public double distanceTo(Point o) {
    final double dx = o.x - x;
    final double dy = o.y - y;
    return sqrt(dx * dx, dy * dy);
  }

  // constructors, getters, setters...
}
```

Example (Scala)

```
class Point(x: Double, y: Double) {

  def distanceTo(o: Point): Double {
    val dx = o.x - x
    val dy = o.y - y
    sqrt(dx * dx, dy * dy)
  }
}
```

Inférence des point-virgules

Inférence de types

Syntaxe économique

Moins cérémonieuse que Java

Example (Java)

```
public class Point {
  private double x, y;

  public double distanceTo(Point o) {
    final double dx = o.x - x;
    final double dy = o.y - y;
    return sqrt(dx * dx, dy * dy);
  }

  // constructors, getters, setters...
}
```

Example (Scala)

```
class Point(x: Double, y: Double) {

  def distanceTo(o: Point): Double {
    val dx = o.x - x
    val dy = o.y - y
    sqrt(dx * dx, dy * dy)
  }
}
```

Inférence des point-virgules

Inférence de types

Syntaxe économique

Moins cérémonieuse que Java

Example (Java)

```
public class Point {
  private double x, y;

  public double distanceTo(Point o) {
    final double dx = o.x - x;
    final double dy = o.y - y;
    return sqrt(dx * dx, dy * dy);
  }

  // constructors, getters, setters...
}
```

Example (Scala)

```
class Point(x: Double, y: Double) {

  def distanceTo(o: Point): Double {
    val dx = o.x - x
    val dy = o.y - y
    sqrt(dx * dx, dy * dy)
  }
}
```

Inférence des point-virgules

Inférence de types

Orienté objet

- Classes
  - Contiennent des variables mutables
- Traits
  - Équivalent des interfaces Java
  - Peuvent contenir des méthodes implémentées
- Objects
  - Singletons
- Case classes et case objects
  - Entièrement immuables
- Value classes
  - Plus légères mais très limitées

Orienté objet

- Classes
  - Contiennent des variables mutables
- Traits
  - Équivalent des interfaces Java
  - Peuvent contenir des méthodes implémentées
- Objects
  - Singletons
- Case classes et case objects
  - Entièrement immuables
- Value classes
  - Plus légères mais très limitées

Orienté objet

Classes
- Contiennent des variables mutables

Traits
- Équivalent des interfaces Java
- Peuvent contenir des méthodes implémentées

Objects
- Singletons

Case classes et case objects
- Entièrement immuables

Value classes
- Plus légères mais très limitées

Orienté objet

Classes
- Contiennent des variables mutables

Traits
- Équivalent des interfaces Java
- Peuvent contenir des méthodes implémentées

Objects
- Singletons

Case classes et case objects
- Entièrement immuables

Value classes
- Plus légères mais très limitées

Orienté objet

Classes
- Contiennent des variables mutables

Traits
- Équivalent des interfaces Java
- Peuvent contenir des méthodes implémentées

Objects
- Singletons

Case classes et case objects
- Entièrement immuables

Value classes
- Plus légères mais très limitées

Fonctions d'ordre supérieur

Littéraux

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> val evens = candidates.filter(x => x % 2 == 0)
evens: List[Int] = List(2, 4, 6, 8)
```

En fait des objets

Example

```
scala> val evenPred = (x: Int) => x % 2 == 0
evenPred: Int => Boolean = <function1>

scala> typeOf[Int => Boolean] =:= typeOf[Function1[Int,Boolean]]
res1: Boolean = true
```

Fonctions d'ordre supérieur

Littéraux

Example

```
scala> val candidates = List(1, 2, 3, 4, 5, 6, 7, 8)
candidates: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8)

scala> val evens = candidates.filter(x => x % 2 == 0)
evens: List[Int] = List(2, 4, 6, 8)
```

En fait des objets

Example

```
scala> val evenPred = (x: Int) => x % 2 == 0
evenPred: Int => Boolean = <function1>

scala> typeOf[Int => Boolean] =:= typeOf[Function1[Int,Boolean]]
res1: Boolean = true
```

Fonctions d'ordre supérieur

Appel

Example

```
scala> val plus = (x: Int, y: Int) => x + y
plus: (Int, Int) => Int = <function2>
scala> plus(5, 4) == plus.apply(5, 4)
res0: Boolean = true
```

Currification

Example

```
scala> val curriedPlus = plus.curried
curriedPlus: Int => (Int => Int) = <function1>
scala> curriedPlus(5)(4) == plus(5, 4)
res1: Boolean = true
```

Fonctions d'ordre supérieur

Appel

Example

```
scala> val plus = (x: Int, y: Int) => x + y
plus: (Int, Int) => Int = <function2>
scala> plus(5, 4) == plus.apply(5, 4)
res0: Boolean = true
```

Currification

Example

```
scala> val curriedPlus = plus.curried
curriedPlus: Int => (Int => Int) = <function1>
scala> curriedPlus(5)(4) == plus(5, 4)
res1: Boolean = true
```

Non-strictness & Laziness

Strict par défaut

Paramètres explicitement non-stricts

Example

```
def if2[A](c: Boolean, tt: => A, ff: => A): A = if (c) tt else ff
```

Valeurs paresseuses

Example

```
def twiceOrZero(c: Boolean, i: => Int): Int = {
  lazy val j = i
  if (c) j * j else 0
}
```

Non-strictness & Laziness

Strict par défaut

Paramètres explicitement non-stricts

Example

```
def if2[A](c: Boolean, tt: => A, ff: => A): A = if (c) tt else ff
```

Valeurs paresseuses

Example

```
def twiceOrZero(c: Boolean, i: => Int): Int = {
  lazy val j = i
  if (c) j * j else 0
}
```

Non-strictness & Laziness

Strict par défaut

Paramètres explicitement non-stricts

Example

```
def if2[A](c: Boolean, tt: => A, ff: => A): A = if (c) tt else ff
```

Valeurs paresseuses

Example

```
def twiceOrZero(c: Boolean, i: => Int): Int = {
  lazy val j = i
  if (c) j * j else 0
}
```

Appariement de formes

Sur les case classes et objects

Example

```
sealed trait Tree
case class Sum(l: Tree, r: Tree) extends Tree
case class Var(n: String) extends Tree
case class Const(v: Int) extends Tree

object Tree {
  def eval(t: Tree, env: Environment): Int = t match {
    case Sum(l, r) => eval(l, env) + eval(r, env)
    case Var(n)  => env(n)
    case Const(v) => v
  }
}
```

Appariement de formes

Sur les case classes et objects

Example

```
sealed trait Tree
case class Sum(l: Tree, r: Tree) extends Tree
case class Var(n: String) extends Tree
case class Const(v: Int) extends Tree

object Tree {
  def eval(t: Tree, env: Environment): Int = t match {
    case Sum(l, r) => eval(l, env) + eval(r, env)
    case Var(n)  => env(n)
    case Const(v) => v
  }
}
```

Système de typage

Typage statique

Basé sur Hindley/Milner

Paramétrés `List[String]`

Existentiels `List[T] forSome { type T }`

Higher-kinded `List`

Structural types `{ def close(): Unit }`

Path-dependent `trait Key { type Value }`

Système de typage

Typage statique

Basé sur Hindley/Milner

Paramétrés `List[String]`

Existentiels `List[T] forSome { type T }`

Higher-kinded `List`

Structural types `{ def close(): Unit }`

Path-dependent `trait Key { type Value }`

Système de typage

Typage statique

Basé sur Hindley/Milner

Paramétrés `List[String]`

Existentiels `List[T] forSome { type T }`

Higher-kinded `List`

Structural types `{ def close(): Unit }`

Path-dependent `trait Key { type Value }`

Système de typage

Typage statique

Basé sur Hindley/Milner

Paramétrés `List[String]`

Existentiels `List[T] forSome { type T }`

Higher-kinded `List`

Structural types `{ def close(): Unit }`

Path-dependent `trait Key { type Value }`

Système de typage

Typage statique

Basé sur Hindley/Milner

Paramétrés `List[String]`

Existentiels `List[T] forSome { type T }`

Higher-kinded `List`

Structural types `{ def close(): Unit }`

Path-dependent `trait Key { type Value }`

Système de typage

Typage statique

Basé sur Hindley/Milner

Paramétrés `List[String]`

Existentiels `List[T] forSome { type T }`

Higher-kinded `List`

Structural types `{ def close(): Unit }`

Path-dependent `trait Key { type Value }`

Système de typage

Typage statique

Basé sur Hindley/Milner

Paramétrés `List[String]`

Existentiels `List[T] forSome { type T }`

Higher-kinded `List`

Structural types `{ def close(): Unit }`

Path-dependent `trait Key { type Value }`

Implicits

Par scope une table d'instances indexée par type

Valeurs implicites `implicit val ctx: Context = ...`

`ctx` ajouté dans la table d'instances

Paramètres implicites `def f()(implicit ctx: Context): Unit = ...`

`ctx` résolu avec le scope de l'appelant

à la compilation

Utilisation des imports pour enrichir le scope

Implicits

Par scope une table d'instances indexée par type

Valeurs implicites `implicit val ctx: Context = ...`

`ctx` ajouté dans la table d'instances

Paramètres implicites `def f()(implicit ctx: Context): Unit = ...`

`ctx` résolu avec le scope de l'appelant

à la compilation

Utilisation des imports pour enrichir le scope

Implicits

Par scope une table d'instances indexée par type

Valeurs implicites `implicit val ctx: Context = ...`

- `ctx` ajouté dans la table d'instances

Paramètres implicites `def f()(implicit ctx: Context): Unit = ...`

- `ctx` résolu avec le scope de l'appelant
  - à la compilation

Utilisation des imports pour enrichir le scope

Implicits

Par scope une table d'instances indexée par type

Valeurs implicites `implicit val ctx: Context = ...`

`ctx` ajouté dans la table d'instances

Paramètres implicites `def f()(implicit ctx: Context): Unit = ...`

`ctx` résolu avec le scope de l'appelant

à la compilation

Utilisation des imports pour enrichir le scope

Implicits

Fonctions implicites

```
implicit def a2b(a: A): B = ...
```

Méthodes de `B` appelable sur un `A`

Classes implicites

```
implicit class RichA(a: A){ ... }
```

Sucre syntaxique pour

```
class RichA(a: A) { ... }
implicit def a2RichA(a: A): RichA = new RichA(a)
```

Implicits

Fonctions implicites

```
implicit def a2b(a: A): B = ...
```

Méthodes de `B` appelable sur un `A`

Classes implicites

```
implicit class RichA(a: A){ ... }
```

Sucre syntaxique pour

```
class RichA(a: A) { ... }
implicit def a2RichA(a: A): RichA = new RichA(a)
```

Méta-programmation

Macros

Plugins compilateur
Ex : Continuations délimitées

Macros + Implicits
Type Providers

Méta-programmation

Macros

Plugins compilateur
    Ex : Continuations délimitées

Macros + Implicits
    Type Providers

Méta-programmation

Macros

Plugins compilateur
    Ex : Continuations délimitées

Macros + Implicits
    Type Providers

Internal Domain Specific Languages

Example (Slick)

```
class Coffees(tag: Tag) extends Table[(String, Double)](tag, "COFFEES") {
  def name = column[String]("COF_NAME")
  def price = column[Double]("PRICE")
  def * = (name, price)
}
val coffees = TableQuery[Coffees]

val q = coffees.filter(_.price > 8.0).map(_.name)
```

Internal Domain Specific Languages

Example (Apache Camel)

```
"direct:b" ==> {
  when(_.in == "<hallo/>") {
    --> ("mock:b")
    to ("mock:c")
  } otherwise {
    to ("mock:e")
  }
  to ("mock:d")
}
```

Internal Domain Specific Languages

Example (Qee)

```
val leq1 = Rule((x) ⇒ (0 ≤ x) → true)
val leq2 = Rule((x) ⇒ (s(x) ≤ 0) → false)
val leq3 = Rule((x, y) ⇒ (s(x) ≤ s(y)) → (x ≤ y))

val plus1 = Rule((x) ⇒ (0 + x) → x)
val plus2 = Rule((x, y) ⇒ (s(x) + y) → s(x + y))

val rules = Set(leq1, leq2, leq3, plus1, plus2)
```

Exécution

Compilation pour la Java Virtual Machine

Interopérabilité avec les librairies Java

APIs Java pour librairies Scala souvent indispensables
- Car mapping Scala vers Java non-immédiat

Librairie standard Scala obligatoire

Possibilité développement Android
- Élagage code inutilisé indispensable (ProGuard)

Exécution

Compilation pour la Java Virtual Machine

Interopérabilité avec les librairies Java

APIs Java pour librairies Scala souvent indispensables
- Car mapping Scala vers Java non-immédiat

Librairie standard Scala obligatoire

Possibilité développement Android
- Élagage code inutilisé indispensable (ProGuard)

Exécution

Compilation pour la Java Virtual Machine

Interopérabilité avec les librairies Java

APIs Java pour librairies Scala souvent indispensables
- Car mapping Scala vers Java non-immédiat

Librairie standard Scala obligatoire

Possibilité développement Android
- Élagage code inutilisé indispensable (ProGuard)

Exécution

Compilation pour la Java Virtual Machine

Interopérabilité avec les librairies Java

APIs Java pour librairies Scala souvent indispensables
- Car mapping Scala vers Java non-immédiat

Librairie standard Scala obligatoire

Possibilité développement Android
- Élagage code inutilisé indispensable (ProGuard)

Exécution

Compilation pour la Java Virtual Machine

Interopérabilité avec les librairies Java

APIs Java pour librairies Scala souvent indispensables
- Car mapping Scala vers Java non-immédiat

Librairie standard Scala obligatoire

Possibilité développement Android
- Élagage code inutilisé indispensable (ProGuard)

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Librairie standard

Collections

- Un jeu d'une dizaine de traits de base
- Implémentations mutables et immuables
- Un grand nombre de combinateurs
- Opérations parallèles avec les même combinateurs
- Vues paresseuses

Réflection

- Éléments de code source
- Miroirs dans le système de typage
- Utilisable à la compilation et à l'exécution

Compilateur alternatif: ScalaJS

Compilateur Scala vers JavaScript

Support de Scala complet (y compris les macros)

Interopérabilité avec code JavaScript

Typage static pour plusieurs librairie JS
DOM, jQuery, PouchDB, Google Maps, CreateJS, ...

Beaucoup de librairie cross-compilent

Les plus sophistiquées sont portées
Akka, Scalaz, Shapeless, ...

Compilateur alternatif: ScalaJS

Compilateur Scala vers JavaScript

Support de Scala complet (y compris les macros)

Interopérabilité avec code JavaScript

Typage static pour plusieurs librairie JS
- DOM, jQuery, PouchDB, Google Maps, CreateJS, ...

Beaucoup de librairie cross-compilent

Les plus sophistiquées sont portées
- Akka, Scalaz, Shapeless, ...

Compilateur alternatif: ScalaJS

Compilateur Scala vers JavaScript

Support de Scala complet (y compris les macros)

Interopérabilité avec code JavaScript

Typage static pour plusieurs librairie JS
    DOM, jQuery, PouchDB, Google Maps, CreateJS, ...

Beaucoup de librairie cross-compilent

Les plus sophistiquées sont portées
    Akka, Scalaz, Shapeless, ...

Compilateur alternatif: ScalaJS

Compilateur Scala vers JavaScript

Support de Scala complet (y compris les macros)

Interopérabilité avec code JavaScript

Typage static pour plusieurs librairie JS
    DOM, jQuery, PouchDB, Google Maps, CreateJS, ...

Beaucoup de librairie cross-compilent

Les plus sophistiquées sont portées
    Akka, Scalaz, Shapeless, ...

Compilateur alternatif: ScalaJS

Compilateur Scala vers JavaScript

Support de Scala complet (y compris les macros)

Interopérabilité avec code JavaScript

Typage static pour plusieurs librairie JS
    DOM, jQuery, PouchDB, Google Maps, CreateJS, ...

Beaucoup de librairie cross-compilent

Les plus sophistiquées sont portées
    Akka, Scalaz, Shapeless, ...

Compilateur alternatif: ScalaJS

Compilateur Scala vers JavaScript

Support de Scala complet (y compris les macros)

Interopérabilité avec code JavaScript

Typage static pour plusieurs librairie JS
    DOM, jQuery, PouchDB, Google Maps, CreateJS, ...

Beaucoup de librairie cross-compilent

Les plus sophistiquées sont portées
    Akka, Scalaz, Shapeless, ...

Dans l'industrie

LinkedIn, EDF Trading, Twitter, Novell, the Guardian, New York Times, Xebia, Xerox, FourSquare, Sony, Siemens, Thatcham, OPower, GridGain, AppJet, Reaktor, VMWare, Foursquare ...

Typesafe Reactive Platform: Play, Akka, Spark

Plus présent dans l'industrie que Haskell

Dans l'industrie

LinkedIn, EDF Trading, Twitter, Novell, the Guardian, New York Times, Xebia, Xerox, FourSquare, Sony, Siemens, Thatcham, OPower, GridGain, AppJet, Reaktor, VMWare, Foursquare ...

Typesafe Reactive Platform: Play, Akka, Spark

Plus présent dans l'industrie que Haskell

Dans l'industrie

LinkedIn, EDF Trading, Twitter, Novell, the Guardian, New York Times, Xebia, Xerox, FourSquare, Sony, Siemens, Thatcham, OPower, GridGain, AppJet, Reaktor, VMWare, Foursquare ...

Typesafe Reactive Platform: Play, Akka, Spark

Plus présent dans l'industrie que Haskell

Dans la recherche

Principalement sur les langages de programmation

EPFL, University of Edinburgh, University of Glasgow, University of Athens, Universität Bremen, Technischen Universität Darmstadt, Chalmers University of Technology, ...

Dans la recherche

Principalement sur les langages de programmation

EPFL, University of Edinburgh, University of Glasgow, University of Athens, Universität Bremen, Technischen Universität Darmstadt, Chalmers University of Technology, ...

Dans l'éducation

EPFL, Stanford University, Oxford University, Georgia Institute of Technology, ...

Moins enseigné que Haskell

Dans l'éducation

EPFL, Stanford University, Oxford University, Georgia Institute of Technology, ...

Moins enseigné que Haskell

Scala 2.12

Prochaine version majeure

Profite des nouveautés apportées à la JVM pour Java 8

- Génération de clôture comme Java 8
  - Pas de classe anonyme pour chaque lambda
  - Réduction de la taille du code généré
- Utilisation des default methods pour les traits

Nouvelle génération de code

- Plus efficace
- Meilleures optimisations de code
- Meilleure détecteur de code mort

Scala 2.12

Prochaine version majeure

Profite des nouveautés apportées à la JVM pour Java 8

- Génération de clôture comme Java 8
  - Pas de classe anonyme pour chaque lambda
  - Réduction de la taille du code généré
- Utilisation des default methods pour les traits

Nouvelle génération de code

- Plus efficace
- Meilleures optimisations de code
- Meilleure détecteur de code mort

Scala 2.12

Prochaine version majeure

Profite des nouveautés apportées à la JVM pour Java 8

- Génération de clôture comme Java 8
  - Pas de classe anonyme pour chaque lambda
  - Réduction de la taille du code généré
- Utilisation des default methods pour les traits

Nouvelle génération de code

- Plus efficace
- Meilleures optimisations de code
- Meilleure détecteur de code mort

Scala 2.12

Prochaine version majeure

Profite des nouveautés apportées à la JVM pour Java 8

- Génération de clôture comme Java 8
  - Pas de classe anonyme pour chaque lambda
  - Réduction de la taille du code généré
- Utilisation des default methods pour les traits

Nouvelle génération de code

- Plus efficace
- Meilleures optimisations de code
- Meilleure détecteur de code mort

Scala 2.12

Prochaine version majeure

Profite des nouveautés apportées à la JVM pour Java 8

- Génération de clôture comme Java 8
  - Pas de classe anonyme pour chaque lambda
  - Réduction de la taille du code généré
- Utilisation des default methods pour les traits

Nouvelle génération de code

- Plus efficace
- Meilleures optimisations de code
- Meilleure détecteur de code mort

Scala 2.12

Prochaine version majeure

Profite des nouveautés apportées à la JVM pour Java 8

- Génération de clôture comme Java 8
  - Pas de classe anonyme pour chaque lambda
  - Réduction de la taille du code généré
- Utilisation des default methods pour les traits

Nouvelle génération de code

- Plus efficace
- Meilleures optimisations de code
- Meilleure détecteur de code mort

Scala 2.12

Prochaine version majeure

Profite des nouveautés apportées à la JVM pour Java 8

- Génération de clôture comme Java 8
  - Pas de classe anonyme pour chaque lambda
  - Réduction de la taille du code généré
- Utilisation des default methods pour les traits

Nouvelle génération de code

- Plus efficace
- Meilleures optimisations de code
- Meilleure détecteur de code mort

Scala 2.12

Prochaine version majeure

Profite des nouveautés apportées à la JVM pour Java 8

- Génération de clôture comme Java 8
  - Pas de classe anonyme pour chaque lambda
  - Réduction de la taille du code généré
- Utilisation des default methods pour les traits

Nouvelle génération de code

- Plus efficace
- Meilleures optimisations de code
- Meilleure détecteur de code mort

Dotty

Nouveau support pour l'expérimentation

Future implémentation de Scala

Système de typage entièrement refondu

- Plus petit ensemble de fondamentaux
- Permettant d'exprimer le système de typage de Scala
- DOT, a calculus for dependent object types [ARO14]

Octobre 2015: Dotty compiles itself

Dotty

Nouveau support pour l'expérimentation

Future implémentation de Scala

Système de typage entièrement refondu

- Plus petit ensemble de fondamentaux
- Permettant d'exprimer le système de typage de Scala
- DOT, a calculus for dependent object types [ARO14]

Octobre 2015: Dotty compiles itself

Dotty

Nouveau support pour l'expérimentation

Future implémentation de Scala

Système de typage entièrement refondu

- Plus petit ensemble de fondamentaux
- Permettant d'exprimer le système de typage de Scala
- DOT, a calculus for dependent object types [ARO14]

Octobre 2015: Dotty compiles itself

Dotty

Nouveau support pour l'expérimentation

Future implémentation de Scala

Système de typage entièrement refondu

- Plus petit ensemble de fondamentaux
- Permettant d'exprimer le système de typage de Scala
- DOT, a calculus for dependent object types [ARO14]

Octobre 2015: Dotty compiles itself

Dotty

Nouveau support pour l'expérimentation

Future implémentation de Scala

Système de typage entièrement refondu

- Plus petit ensemble de fondamentaux
- Permettant d'exprimer le système de typage de Scala
- DOT, a calculus for dependent object types [ARO14]

Octobre 2015: Dotty compiles itself

Dotty

Nouveau support pour l'expérimentation

Future implémentation de Scala

Système de typage entièrement refondu

- Plus petit ensemble de fondamentaux
- Permettant d'exprimer le système de typage de Scala
- DOT, a calculus for dependent object types [ARO14]

Octobre 2015: Dotty compiles itself

Dotty

Nouveau support pour l'expérimentation

Future implémentation de Scala

Système de typage entièrement refondu

- Plus petit ensemble de fondamentaux
- Permettant d'exprimer le système de typage de Scala
- DOT, a calculus for dependent object types [ARO14]

Octobre 2015: Dotty compiles itself

Conclusion

Multi-paradigme: orienté objet, procédural, fonctionnel

Apprentissage graduel possible

Interopérabilité avec Java

Forte demande en développeurs

Bientôt dans les universités françaises ?

Conclusion

Multi-paradigme: orienté objet, procédural, fonctionnel

Apprentissage graduel possible

Interopérabilité avec Java

Forte demande en développeurs

Bientôt dans les universités françaises ?

Conclusion

Multi-paradigme: orienté objet, procédural, fonctionnel

Apprentissage graduel possible

Interopérabilité avec Java

Forte demande en développeurs

Bientôt dans les universités françaises ?

Conclusion

Multi-paradigme: orienté objet, procédural, fonctionnel

Apprentissage graduel possible

Interopérabilité avec Java

Forte demande en développeurs

Bientôt dans les universités françaises ?

Conclusion

Multi-paradigme: orienté objet, procédural, fonctionnel

Apprentissage graduel possible

Interopérabilité avec Java

Forte demande en développeurs

Bientôt dans les universités françaises ?

Pointeurs

www.scala-lang.org
www.typesafe.com

[ARO14] Nada Amin, Tiark Rompf, and Martin Odersky. Foundations of Path-dependent Types. In *Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA '14, pages 233–249, New York, NY, USA, 2014. ACM.

[BOSW98] Gilad Bracha, Martin Odersky, David Stoutamire, and Philip Wadler. Making the Future Safe for the Past: Adding Genericity to the Java Programming Language. In *Proceedings of the 13th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '98, pages 183–200, New York, NY, USA, 1998. ACM.

[Oa04] Martin Odersky and al. An Overview of the Scala Programming Language. Technical Report IC/2004/64, EPFL Lausanne, Switzerland, 2004.