Scala

Gilles.Dubochet@epfl.ch, EPFL – Faculté IC – Laboratoire des méthodes de programmation, assistant

Scala is a programming language developed at EPFL. Today, it is amongst the 50 most popular languages, attracting over 100,000 visitors per month to its website. To explain its success: flawless compatibility with Java and superb scalability — thanks to programming language research.

Scala est un langage de programmation développé à l'EPFL, qui se trouve aujourd'hui parmi les 50 langages les plus populaires. Son site Web reçoit plus de 100'000 visites par mois. Les raisons de son succès? sa parfaite compatibilité avec Java et son extrême polyvalence — grâce à l'application de la recherche sur les langages.

La compétition pour devenir le langage de programmation le plus populaire est intense. Mais l'EPFL est bien placée dans cette course grâce à Scala, développé depuis 2001 au laboratoire des méthodes de programmation du professeur Martin Odersky. Il faut dire que, de tous les prétendants, Scala est l'un des plus évolués grâce à son ancrage dans la recherche. Toutefois, il a dès le début été conçu comme un langage pratique, par exemple en assurant une compatibilité parfaite avec l'ensemble des bibliothèques et outils Java existants. Ce choix — à la pointe des connaissances, mais sans rejeter ce qui existe — a permis à Scala de gagner une forte popularité, particulièrement depuis deux ans. Ainsi, le langage de l'EPFL est un prétendant crédible pour devenir un langage de premier plan. Aujourd'hui, il est déjà parmi les 50 langages les plus populaires selon l'index TIOBE[1]; il est utilisé par de grandes entreprises dans le Web social (**Twitter**, **LinkedIn**, **FourSquare**), dans l'édition (**The Guardian**, **Nature Publishing**), dans les jeux ou dans la finance; presque 20 livres de tous niveaux sur Scala sont disponibles ou annoncés; de plus en plus d'universités donnent des cours de Scala à leurs étudiants. Mais qu'est-ce qui explique ce succès?

Une question de niche

Pour commencer à comprendre cette réussite, il faut ausculter les tendances actuelles de la programmation. Avec l'avènement de Java, au début des années 2000, on avait pu croire à une *fin de l'histoire* des langages. À cette époque, Java disposait d'une popularité de presque 30 % (index TIOBE), 10 points au dessus de son plus proche rival. De plus en plus d'universités, dont la

[1] www.tiobe.com/content/paperinfo/tpci/

.. /.. Suite page 6

.. /.. Suite de la première page

section I&C de l'EPFL, focalisaient leur cursus de programmation sur Java. Il était naturel d'imaginer que ce langage deviendrait à terme hégémonique. Il n'en est rien !

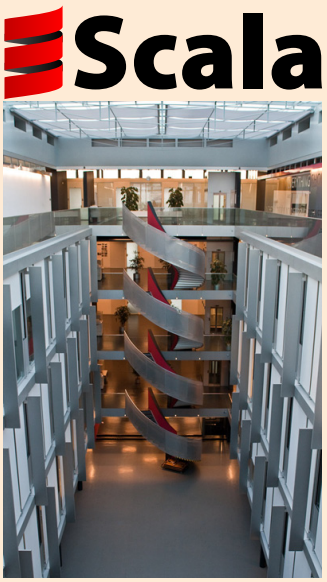
Au contraire, on observe aujourd'hui un écosystème de langages très varié. En particulier des langages dits *agiles* — par exemple PHP, Python ou Ruby —, dotés de systèmes de types dynamiques, sont en pleine progression depuis quelques années. Ces langages compensent une faible performance par un style de programmation décontracté, que les programmeurs apprécient et qui accélère l'écriture des logiciels.

Le langage Java, statiquement typé, est plus rigide que les langages agiles. Quoique disposant depuis quelques années de *types génériques*, la forme de programmation orientée objet qu'il propose reste relativement simple. Mais Java est une plate-forme autant qu'un langage, et cette plate-forme est remarquable. Au centre, la machine virtuelle (JVM) permet d'exécuter le même code binaire sur toutes sortes d'ordinateurs. Elle prend également en charge la gestion de la mémoire, une tâche difficile que les langages comme C++ délèguent au programmeur lui-même. Si la JVM originale souffrait d'une certaine lenteur, celle de 2010 est une bête de course qui, dans de bonnes conditions, interprète les programmes plus rapidement que l'ordinateur ne le ferait directement – en optimisant le code en fonction des performances mesurées. Quant à la bibliothèque standard, il s'agit d'une boite à outils extrêmement pratique pour le programmeur, complétée par une multitude de bibliothèques tierces, dont la compatibilité apportée par la JVM facilite le partage. Finalement, la rigidité du langage est en partie contournée en supportant le code par des logiciels d'intégration. Par exemple, la norme Java EE ne modifie pas le langage Java, mais exécute les programmes dans un conteneur qui fournit une série de services facilitant, entre autres, la programmation Web et la communication entre composants.

On peut donc observer dans la situation actuelle une double tendance: D'un côté, une grande valeur d'usage dans les plates-formes (Java et .Net de Microsoft, en particulier) du fait de la concentration de bibliothèques et d'outils autour des machines virtuelles. De l'autre côté, un très grand attrait des programmeurs et des entreprises pour les langages agiles, plus décontractés, moins normatifs et généralement considérés comme plus productifs. Malheureusement, les deux tendances semblaient contradictoires, car les machines virtuelles imposent de fortes contraintes sur le code binaire qu'elles exécutent, ce qui est contraire aux besoins des langages agiles. D'ailleurs, les langages qui ont essayé avant Scala de résoudre cette contradiction l'ont fait au détriment de la performance ou de la compatibilité.

Scala prend acte de cette situation et tente de trouver le bon équilibre entre ces besoins contradictoires par un design et un compilateur à la pointe de la technologie. Pour commencer, la syntaxe de Scala est beaucoup plus flexible et tolérante que celle de Java — un exemple anecdotique, mais fort apprécié, est qu'elle rend les points-virgules optionnels. Mais plus fondamentalement, c'est le système de types qui offre la solution (voir encart page 8). Scala est en effet doté d'un système de types statiques, nécessaire pour satisfaire les exigences de la machine virtuelle. De l'autre côté, pour avoir un langage agile, le compilateur infère automatiquement la plupart des types, laissant ainsi le programmeur se concentrer sur ses algorithmes. Pour les utilisateurs plus avancés, Scala offre des types extrêmement polyvalents et expressifs qui permettent, suivant les besoins, soit une grande liberté de conception proche des langages dynamiquement typés soit au contraire une vérification rigoureuse des propriétés du programme.

Un langage de l'EPFL



Le compilateur Scala et sa bibliothèque standard sont des projets libres, portés par le laboratoire des méthodes de programmation de l'EPFL. Si le cœur du projet, et la majorité des développeurs se trouvent à Lausanne, la communauté Scala fait aujourd'hui le tour du monde. On trouve des contributeurs en Europe, bien sûr, mais également aux États-Unis, au Japon, ou encore au Brésil. En avril de cette année, plus de 150 personnes, chercheurs et utilisateurs de toutes nationalités se sont rendus aux premières **journées Scala** organisées au Polydôme de l'EPFL.

Quant à l'avenir industriel de Scala, il est assuré par **Scala Solutions**, une jeune pousse enracinée depuis quelques semaines au parc scientifique de l'EPFL, à quelques dizaines de mètres du laboratoire des méthodes de programmation. Cette entreprise, composée en partie d'anciens membres du laboratoire, fourni du conseil et des services autour de Scala, afin de transférer au mieux sa technologie vers les utilisateurs industriels.

Le lien du langage avec l'école est même présent dans le logo, directement inspiré du site de l'EPFL. **Scala**, qui veut dire **escalier** en italien, se devait de mettre en valeur l'architecture locale. Ce sont les escaliers en colimaçon du bâtiment BC qui ont servi de modèle pour le logo.

Apprendre Scala

Le point de départ pour apprendre Scala est le site Web du langage: www.scala-lang.org/. La page **learning Scala** contient une multitude de liens vers diverses ressources d'apprentissage sur le site et sur le Web.

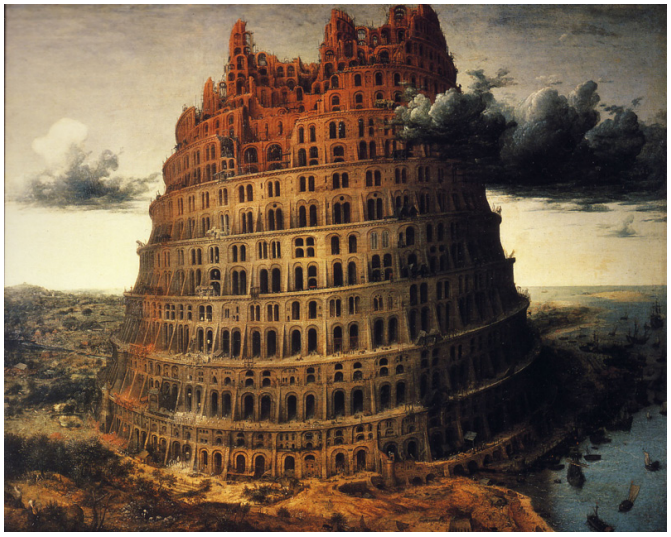
- Si vous êtes un professionnel de la programmation, la société Scala Solutions propose des cours d'introduction ou des cours avancés de Scala.
- Si vous êtes étudiants, inscrivez-vous au cours CS-205 de **programmation avancée** qui est donné en Scala par le laboratoire des méthodes de programmation.

Il existe plusieurs livres sur Scala (en anglais). Pour n'en citer que deux, dont celui de Martin Odersky:

- M. Odersky, L. Spoon,et B. Venners: **Programming in Scala**; Artima.
- A. Payne et D. Wampler:**Programming Scala**; O'Reilly.

La programmation extensible

Si le positionnement de Scala dans une niche particulièrement compétitive explique sans doute une partie de son succès, il faut également considérer ses qualités propres. Elles sont multiples, mais à la racine se trouve la polyvalence de Scala, sa capacité intrinsèque à s'adapter à de multiples tâches. Cette volonté de polyvalence a été présente dès le début de la conception de Scala. Le rapport technique de 2004 qui présente pour la première fois le langage, mentionne dans son introduction l'idée d'un langage extensible *(scalable)*, où **les mêmes concepts peuvent décrire des petits éléments aussi bien que des grands**. Pour comprendre ce que cela signifie, revenons encore une fois aux tendances de la programmation actuelle.



L'Histoire nous dit que la construction de la tour de Babel s'arrêta le jour où les hommes cessèrent de parler la même langue. Il se pourrait bien que l'on observe le même effet dans la construction de logiciels. Prenons l'exemple d'une application Web moderne: le nombre de langages utilisé est énorme. On peut y trouver du JavaScript pour le client, du Python pour *scripter* le serveur, du Java pour la logique métier, du SQL pour l'accès à la base de données, le tout maintenu par du XML. Chaque langage est choisi pour son efficacité dans une tâche spécifique, mais, pour communiquer, ces langages dépendent d'un **plus petit dénominateur commun**, souvent du XML, ou pire, de simples chaînes de caractères sans réelle structure. Cela complique le déploiement, rend les systèmes fragiles et est une grande source d'erreurs.

La conception extensible de Scala ne cherche pas à optimiser son utilisation dans un domaine, mais à le rendre aussi universel que possible. Par exemple, une application Web utilisant la technologie Lift — un logiciel d'intégration Web en Scala — peut être entièrement définie à l'aide d'un seul langage. En effet, les bibliothèques de Lift fournissent au programmeur des boites à outils spécifiques à chacune des tâches requises, qui sont toutes manipulées avec le même langage: Scala. Ainsi, l'obstacle de la communication disparaît, pour ainsi dire, car tous les éléments de l'application parlent la même langue.

La prise en compte simultanée de la recherche en programmation fonctionnelle et celle en programmation orientée objet, deux domaines qui évoluaient jusque-là dans une relative ignorance mutuelle, a permis un langage unique capable de résoudre un large spectre de problèmes. La programmation orientée objet avancée, utilisée par Scala, introduit des idées telles que les **membres abstraits de types**, les **annotations de type individuel** et la **composition par mixin**. Ensemble, ces techniques permettent de structurer un programme sous forme de services abstraits et modulaires, configurés librement selon les besoins spécifiques du programme. Cette modularité participe à l'extensibilité de Scala, d'autant plus que ces techniques remplissent leur rôle aussi bien pour de simples méthodes que pour des composants considérables. Quant à la programmation fonctionnelle, elle apporte en particulier les **fonctions de première classe** qui font du code une valeur qui peut se déplacer dans le programme. Cela favorise l'extensibilité en permettant de contrôler l'exécution du code suivant les besoins. Par ailleurs, les fonctions ouvrent la voie au **filtrage de motifs** *(pattern matching)* qui enrichit l'accès aux objets, abordés du point de vue des structures complexes qu'ils forment au lieu de les observer individuellement.

La contribution de Scala à la recherche dans les langages vient de la mise en commun de concepts et de techniques jusque-là séparés. Il parvient à unifier des éléments similaires, et à reconnaître ceux qui diffèrent et leur rôle dans l'ensemble. Le résultat de cette recherche est ce que l'on pourrait appeler un langage universel. Ce n'est plus un simple langage d'usage général, comme Java, car il permet par son extensibilité de séparer les tâches et les domaines, et de les résoudre de la façon qui leur convient le mieux. En d'autres termes, la capacité du langage à entourer intimement le code avec des librairies est telle que chaque tâche peut être menée à bien dans un milieu qui lui convient spécialement. Certains langages agiles comme Python sont également presque universels, mais Scala crée cette capacité dans le cadre beaucoup plus exigeant d'un système de types statique et de la JVM.

Un exemple de programme Scala

On passe à ce programme une série de couleurs, en toutes lettres. Le programme retourne le code hexadécimal correspondant à chaque couleur qu'il connaît.

```
object ColorCode {

  val colors = Map (
    "rouge"     -> 0xFF0000,
    "turquoise" -> 0x00FFFF,
    "noir"      -> 0x000000,
    "orange"    -> 0xFF8040,
    "brun"      -> 0x804000
  )

  def main(args: Array[String]) {
    for (name <- args) println (
      (colors get name) match {
        case Some(code) =>
          name + " est encod√© par: " + code
        case None =>
          "couleur inconnue: " + name
      }
    )
  }

}
```

[2] Le principal auteur de la bibliothèque d'acteurs est Philippe Haller.

Un langage pour la programmation distribuée

Un des exemples les plus convaincants de l'extensibilité du langage a été l'implantation d'une infrastructure de programmation distribuée de premier plan pour Scala[2]. En effet, avec la multiplication des cœurs dans les processeurs, la programmation **dans le nuage** et les fermes de serveurs, il n'est aujourd'hui plus possible de programmer sans prendre en compte la distribution des calculs. Plusieurs modèles de programmation distribuée existent, l'un des plus prometteurs étant la **programmation par acteurs**. Dans ce modèle, certains objets deviennent des acteurs qui disposent de leur propre fil d'exécution *(thread)* et qui communiquent entre eux à l'aide de messages.

Traditionnellement, l'implantation d'un nouveau modèle de programmation aussi fondamental que les acteurs aurait requis le développement d'un nouveau langage afin de ne pas en compromettre l'ergonomie ou la performance. Pas en Scala: quoique son système d'acteurs soit largement considéré comme le standard actuel, tant du point de vue des performances que de celui de l'ergonomie, il a été entièrement défini à l'aide d'une bibliothèque. Le langage n'a pas été modifié. Sa capacité d'usage universel a permis au concepteur de la bibliothèque de contrôler l'exécution de façon à en assurer la performance, et de maîtriser le milieu de programmation de façon à en assurer le confort.

Aujourd'hui, d'autres implantations d'acteurs, comme *Akka* ou *Lift*, sont disponibles pour Scala. *Lift* fournit des acteurs simplifiés aux performances plus élevées. *Akka* est un environnement plus large dont les acteurs ne sont qu'une partie. Cette diversité démontre la flexibilité d'un langage universel qui ne force pas l'utilisation de certains modèles de programmation, mais permet au contraire d'étendre ceux-ci en fonction des besoins, sans changer le langage. ■

Qu'est-ce qu'un système de types ?

Un système de types statique est un composant du compilateur qui vérifie que les programmes sont cohérents. Cela permet d'éviter un grand nombre d'erreurs avant même que le programme ne soit exécuté. Pour cela, le compilateur donne à chaque élément du programme un type, qui est une approximation de sa valeur réelle. Par exemple, **Fleurette** aura le type vache, ce qui permettra au compilateur d'éviter qu'on ne tente de la tondre, mais permettra qu'on la traie.

Les types peuvent être plus ou moins puissants dans leur capacité descriptive, c'est-à-dire dans la façon dont ils approximent les vraies valeurs. Par exemple, avant les types génériques en Java, il n'était pas possible de différencier une liste de poules d'une liste de vaches, car les deux étaient simplement considérées comme des listes sans autres qualifications.

Par ailleurs, le système de types peut disposer d'opérateurs plus ou moins puissants pour agir sur les types. Par exemple, en Scala, il est possible de joindre plusieurs types dans un *mixin*, ce type composé disposant de toutes les propriétés des types originaux. En joignant les types d'un cheval, d'une carriole et d'un cocher, on peut obtenir un type représentant un attelage qui permettra de hennir (du cheval) aussi bien que de fouetter (du cocher).