SpaceFox

Zeste de savoir

Le langage Ceylon passe en v1.1

12 août 2019

Table des matières

Un peu moins d'un an après la sortie de la v1.0 , Ceylon gagne en maturité en passant en v1.1 . *« Voilà qui me fait une belle jambe, parce que je n'ai pas la moindre idée de ce qu'est "Ceylon" »*, me direz-vous avec perspicacité.

Eh bien, voilà une excellente occasion de le découvrir ensemble !

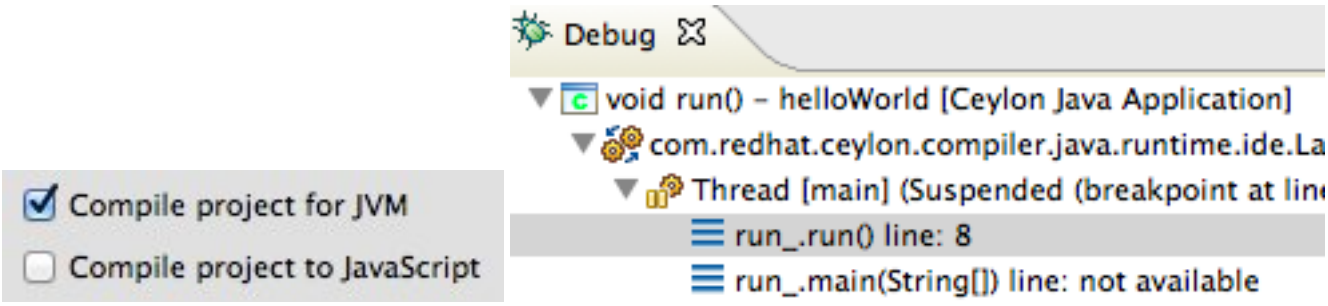
1. Présentation du langage de programmation Ceylon

Vers 2009, Gavin King, créateur de Hibernate et de JBoss Seam, se fit la réflexion que Java avait quand même quelques inconvénients, qu'il avançait peu[1] et qu'on pourrait sans doute faire un langage bien plus moderne.

Ainsi donc il s'allia à RedHat pour créer ce qui allait devenir Ceylon, un langage dont les concepts fondamentaux sont les suivants, si j'en crois les fonctionnalités clefs ainsi que l'introduction rapide :

1.1. Un langage multi-plateforme

Non seulement Ceylon est conçu pour être compilé pour la JVMet Javascript, mais en plus peut interagir nativement avec Java et Javascript.



Pourquoi un tel choix ? Pour plusieurs raisons :

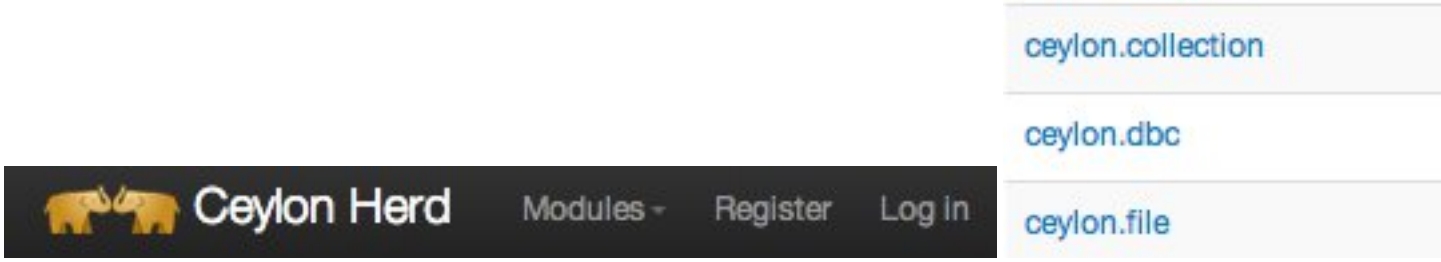
— Pour tourner sur n'importe quel OS possédant une machine virtuelle Java 7+ ou JavaScript.
— Pour faciliter l'interopérabilité avec les énormes bases de code existant dans ces langages.
— Parce que compiler en JavaScript permet d'avoir du code exécutable dans les navigateurs – et on a pas trop le choix du langage.
— Parce que compiler pour la JVMpermet de profiter de la performance, de la stabilité et de la fiabilité des JVMs et des outils associés.
— Parce que Ceylon a été explicitement conçu comme un successeur de Java.

Exemples d'interopérabilité (tiré du site officiel, comme tous les exemples de cet article) :

Contenu masqué n°1

1.2. Fourni avec un SDK complet

Parce qu'un langage sans SDK ne sert pas à grand-chose, Celyon est fourni avec un SDK qui se veut complet et modulaire.

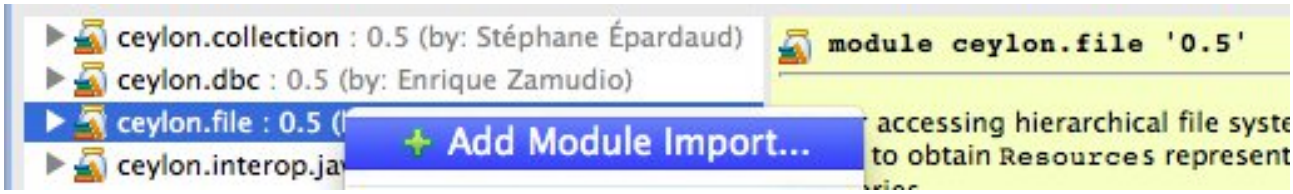


« Se veut » parce que pour l'instant, on peut remarquer deux inconvénients à ce SDK :

1. Il manque encore certaines fonctionnalités (par exemple, le module de logs n'est arrivé qu'avec la v1.1).
2. La majorité de ce SDK n'est disponible qu'avec Java, pas avec Javascript.

1.3. Modularité

Le code est organisé en package et en module, et est compilé sous forme de modules.



Qu'est-ce que cela implique concrètement ?

— Qu'il est possible de déclarer des modules *directement avec leurs dépendances*, ce qui évite de maintenir un système de dépendances externes :

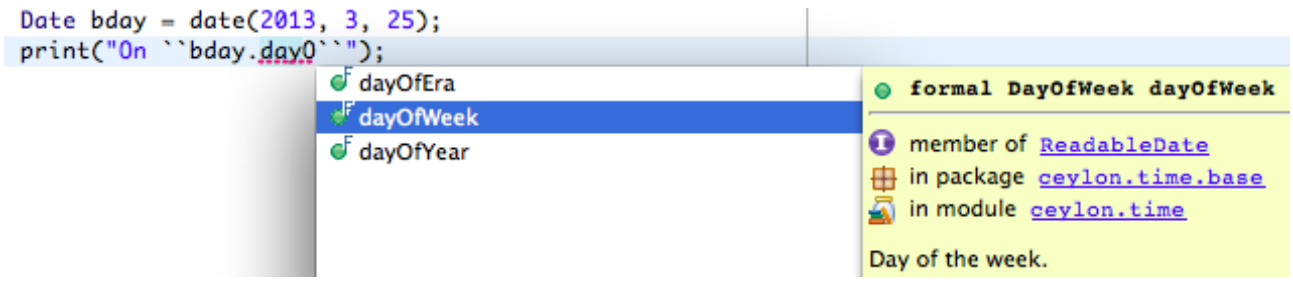
```
1 module org.jboss.example "1.0.0" {
2     import ceylon.math "1.1.0";
3     import ceylon.file "1.1.1";
4 }
```

— Qu'en cas de compilation vers Java, tout le module est compilé dans un seul fichier, avec des métadonnées compatibles avec les conteneurs OSGi et Vert.x. Mieux, ces modules sont chargés dynamiquement à l'exécution.
— Qu'en cas de compilation vers JavaScript, le compilateur génère des modules au format CommonJS donc compatibles avec node.js et require.js.

Outre le fait que ce système de modules est un serpent de mer en Java (il est prévu pour Java ~~7~~ ~~8~~ 9), il offre souplesse, performance et compatibilité avec les systèmes existants. Et arrive avec un dépôt de modules centralisé, Ceylon Herd . Que demande le peuple ?[2]

1.4. Outils de développements

Comment imaginer un langage moderne sans IDE du même acabit ? RedHat fournit bien sûr des outils en ligne de commande , mais aussi un module Eclipse pour développer en Ceylon – lequel module est d'ailleurs développé par David Festal de la société SERLI , développeur et société françaises.



1.5. Un modèle 100% défensif

C'est pour moi une *killer-feature* de Ceylon et je m'étonne qu'elle ne soit pas plus mise en avant. Ceylon se protège au maximum contre les surprises, ce qui implique entre autres que :

— Par défaut, *tous les éléments sont privés*. Il faut déclarer explicitement si on veut les voir ailleurs que dans leur propre bloc.
— Par défaut, *toutes les valeurs sont immutables*. Si mon élément peut être modifié, il doit être explicitement déclaré comme `variable` : `variable Integer count;`
— Par défaut, *une valeur ne peut pas être* `null`. Si une variable peut être nullable, ceci doit être déclaré explicitement. Adieu, `NullPointerException` !

Un petit exemple avec ce dernier point : `String name = ...` doit être défini et donc ne peut pas contenir `null`. Si j'ai besoin que mon nom puisse être indéfini, je dois le déclarer ainsi :

```
1 String? name = ...
```

Ce qui me permet d'écrire :

```
1 void hello(String? name) {
2     if (exists name) {
3         //name is of type String here
4         print("Hello, ``name``!");
5     }
6     else {
7         print("Hello, world!");
8     }
9 }
```

1.6. Un système de type puissant

Le système de type gère l'union et l'intersection de types, les types énumérés et les alias de types. Mieux : Ceylon est capable d'inférence de type, laquelle inférence gère toutes ces subtilités !

Bien sûr, comme tout système puissant, il permet de faire des choses difficiles à comprendre. C'est donc à utiliser uniquement lorsque cela permet de rendre le code plus clair et plus efficace.

Alors là, je sens que j'ai perdu la plupart des lecteurs, donc je vais vous sortir des explications, des exemples, et des titres de niveau 3.

1.6.1. Union et intersection de type

Une union de types, c'est quand je veux dire qu'un élément est du type A *ou* du type B (*ou* inclusif). Exemple : mon élément est une personne ou une organisation :

```
Person | Organization personOrOrganization = ... ;
```

Une intersection de types, c'est l'autre opération logique : je veux dire qu'un élément est du type A *et* du type B en même temps. Exemple : mon élément est imprimable, a une taille et est persistant :

```
Printable & Sized & Persistent printableSizedPersistent = ... ;
```

1.6.2. Types énumérés

Les types énumérés me permettent de définir une série de sous-types et de faire des traitements différenciés sur chacun des sous-types, tout en permettant au compilateur de signaler qu'un des traitements est incomplet. L'exemple détaillé est un peu long alors je vous le mets en secret.

Contenu masqué n°2

1.6.3. Alias de types

Il s'agit simplement de renommer un type : `interface Strings => List<String>;`

1.6.4. Inférence de type

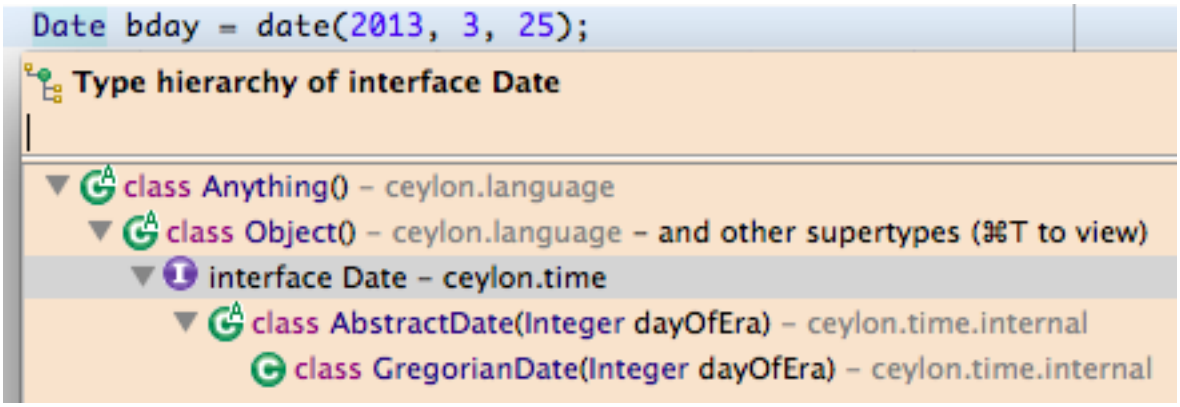
En gros, il s'agit de permettre au compilateur de *deviner* le type réellement utilisé en fonction des informations dont il dispose. L'avantage pour Ceylon, c'est que couplé aux éléments précédents, il permet des inférences plutôt précises. Jugez plutôt : face à ceci :

```
value numbers = HashMap { "one"->1.0, "zero"->0.0, 1->1.0, 0->0.0
    };
```

Le compilateur va déduire que le type de `numbers` est `HashMap<String|Integer,Float>` (un tableau associatif qui associe une chaîne de caractères ou un entier à un flottant).

1.7. Programmation orientée objet avancée

Non seulement en Ceylon *tout* est objet (y compris les nombres, la valeur `null`, les fonctions et les classes) ; mais en plus le langage supporte les fonctions d'ordre supérieur et les listes en compréhensions. Il supporte en plus une version restreinte de l'héritage multiple appelée *mixin inheritance*[3] – qui ressemble fortement à ce qui a été mis en place dans Java 8 d'ailleurs.



Là encore, on rentre dans la technique, alors voici quelques explications :

1.7.1. Les fonctions d'ordre supérieur

Une fonction d'ordre supérieur, en informatique, c'est une fonction qui prend une ou plusieurs fonctions en entrée ; ou (inclusif) qui renvoie une fonction.

Les détails nous entraîneraient très loin, aussi je vous propose l'article Wikipédia sur le sujet et l'implémentation Ceylon de la chose .

1.7.2. Les listes en compréhension

Une liste en compréhension est une liste dont le contenu est défini par le filtrage d'une autre liste. Par exemple, renvoyer la liste des adultes peut s'écrire :

```
1 [ for (p in people) if (p.age>=18) p ]
```

Là aussi les explications peuvent être longues, c'est pourquoi je vous renvoie à l'article Wikipédia et à l'implémentation en Ceylon .

1.8. … et bien d'autres choses !

Je pourrais encore parler par exemple de bien d'autres choses, comme :

- Une syntaxe déclative pour les structures complexes – allez voir l'exemple, c'est encore le plus simple,
- Les tuples , outil pratique s'il en est,
- La réduction automatique de type , qui permet de tester le type d'un objet et de le convertir dans ce type en une seule opération,
- Les *generics* qui ne perdent pas leur type à la compilation , à faire baver tout développeur Java et la gestion de leur variance ,
- Les inévitables annotations (il y a *toujours* des annotations, mais qui s'en sert vraiment ?),
- La métaprogrammation ,
- Des attributs polymorphiques et opérateurs polymorphiques ,
- …

Bref, vous l'avez compris : si vous désirez en savoir plus, vous devriez lire les fonctionnalités clefs de Ceylon , puis l'introduction à Ceylon . Enfin, lancez-vous dedans grâce au tour de Ceylon , qui suppose que vous avez quand même une connaissance minimale de la programmation orientée objet.

2. Ceylon, un projet 100% libre

Une des caractéristiques de Ceylon, c'est que le projet dans son intégralité est libre !

Pour commencer, le langage est intégralement spécifié , et la spécification est elle-même libre. Vous voulez créer un langage dérivé ? Allez-y !

Ensuite, l'implémentation standard est libre elle aussi. Tout comme le SDK. Et l'IDE. Et Herd, le gestionnaire de module. Et les sites qui y sont liés. Et la doc. Et tous les exemples. En fait, absolument tout est disponible sur Github !

La licence exacte dépend de ce dont vous parlez, la page de licence détaille ça mieux que moi .

3. Les nouveautés de la v1.1

Ah oui, parce que c'était le sujet de l'article à la base. Enfin, je pense que vous avez fini par comprendre que le *véritable* sujet ce cet article c'était de vous faire une présentation de Ceylon, mais la v1.1 m'a fourni un beau prétexte, alors voici quand même les nouveautés qu'elle apporte !

- Des améliorations de performance. En particulier des temps de compilation, parce que le compilateur n'était pas spécialement nerveux.
- La gestion des modules OSGi et l'intégration à Vert.x, qui permettent de créer des modules utilisables sur des plate-formes existantes et connues.

— Des améliorations du SDK, avec l'arrivée de `ceylon.promise`, `ceylon.locale` et `ceylon.logging`. En voyant que les deux derniers n'existaient pas en v1.0, on se dit que le langage est *vraiment* jeune.
— Un module de formatage de code pour l'IDE.
— Plein de petits bugs.

Ceux que ça intéresse iront consulter la liste complète . Plus pour savoir ce qu'on peut corriger et améliorer dans un langage neuf que pour le contenu de la liste elle-même, j'imagine (et c'est d'ailleurs assez intéressant comme exercice).

4. Ceylon aujourd'hui

Voilà pour le tour d'horizon de Ceylon. Et si on prenait un peu de recul face à cette présentation quelque peu commerciale, et qu'on regardait ce qu'il en est aujourd'hui ?

4.1. Accueil

Ceylon a reçu un accueil assez mitigé. Parmi ceux qui ont un avis sur la question, on constate en gros deux positions :

— Ceux qui considèrent que Ceylon est un excellent langage qui peut remplacer Java un peu partout, et donc est probablement le futur de Java s'il arrive à se lancer. Notez qu'on a déjà dit ça de Scala et de Groovy à leurs époques, pour ne citer que les deux plus connus, et pourtant Java est toujours bien vivant (même si ces deux langages ont trouvé leur public).
— Ceux qui considèrent que Ceylon est un n$^{\text{ième}}$ concurrent inutile à Scala ou Groovy et que les efforts sur Ceylon devraient plutôt porter sur ces langages. Mais c'est oublier les spécificités de Ceylon (support de la JVMet de JavaScript, approche défensive, ...).

Comme d'habitude, la vérité est sans doute quelque part au milieu... Si on essaie de faire le bilan, que trouve-t-on ?

4.2. Avantages

— L'approche défensive.
— Le côté très carré qui va avec : le compilateur peut sortir un max d'erreurs.
— Les variables qui ne peuvent pas être `null` par mégarde.
— Le système de type, que je trouve un peu déroutant au premier abord mais qui semble ultra-pratique.
— 100% libre !

4.3. Inconvénients

— Le manque total de références. C'est **le** gros problème de Ceylon : on a pas encore de projet digne de ce nom en référence sur le site officiel, et le dépôt de modules reste plus ou moins vide.
— Le support de JavaScript partiel. Les 2/3 du SDK n'existent tout simplement pas en JavaScript...
— Le logo « Éléphant » est déjà pris par PHP !
— Sans doute d'autres trucs qu'il faudrait plus d'usage pour s'en rendre compte.

5. Et maintenant ?

Eh bien voilà, je suis arrivé au bout de cette présentation. N'hésitez pas à dire ci-dessous, dans les commentaires, ce que vous en avez pensé.

Le plus simple est sans doute d'essayer par vous-même pour vous faire une idée, si vous avez un minimum de connaissances en programmation et en anglais :

— Les fonctionnalités
— Pour commencer , des exemples à télécharger et essayer
— L'introduction à Ceylon
— Le tour de Ceylon
— La FAQ
— Le forum

Et puis, vous pouvez toujours tenter de faire le Javaquarium en Ceylon . Tiens, c'est une idée ça : je vais le faire !

Si vous aimez le futur et la programmation d'un langage, jetez un œil sur l'avenir du projet !

Ah j'oubliais : pourquoi « Ceylon » ? Parce que c'est l'ancien nom du Sri Lanka (Ceylan en français), que c'est proche de Java, et qu'on y produit pas mal de thé – boisson plus consommée dans le monde que le café...

6. Crédits

— Les images (y compris l'icône) sont issues du site de Ceylon et donc sont sous licence CC-BY-SA 3.0 Unported.
— Les codes d'exemples sont eux aussi du site de Ceylon et sont donc sous licence Apache Software Licence 2.0.

1. C'était avant la sortie de Java 8, qui est sans doute la plus grosse avancée de Java de tous les temps depuis sa sortie.
2. *Panem et circences* , probablement.
3. Je n'ai pas la moindre idée de la traduction standard de ce terme, si tant est qu'elle existe.

Contenu masqué

Contenu masqué n°1

Utilisation de Java dans Ceylon :

```
import java.util { HashMap }

value javaHashMap = HashMap<String,Integer>();
javaHashMap.put("zero", 0);
javaHashMap.put("one", 1);
javaHashMap.put("two", 2);
print(javaHashMap.values());
```

Utilisation de Javascript dans Ceylon :

```
dynamic {
    dynamic req = XMLHttpRequest();
    req.onreadystatechange = void () {
        if (req.readyState==4) {
            document.getElementById("greeting")
                    .innerHTML = req.status==200
                            then req.responseText
                            else "error";
        }
    };
    req.open("GET", "/sayHello", true);
    req.send();
}
```

Retourner au texte.

Contenu masqué n°2

Imaginons que je veuille faire un traitement sur un arbre. Un arbre a des branches et des feuilles, que l'on regroupe sous le terme générique « nœud » :

```
abstract class Node() of Leaf | Branch {}
```

On fait un traitement quelconque sur ces nœuds. Le traitement différencie les feuilles des nœuds :

Contenu masqué

```
Node node = ... ;
switch (node)
case (is Leaf) {
    //node is of type Leaf here
    print(node.leafValue);
}
case (is Branch) {
    //node is of type Branch here
    doSomething(node.leftNode);
    doSomething(node.rightNode);
}
```

Si demain je rajoute un troisième sous-type à mes nœuds, par exemple des fleurs ou des fruits, le compilateur va me signaler qu'il manque un bout du traitement à chaque fois que ce genre de `switch` est utilisé. Retourner au texte.

Liste des abréviations

JVM Java Virtual Machine – La machine virtuelle Java.. 1, 2, 8