Les Énumérations en Nit

Yannick Denis Stephan Michaud

October 14, 2013

La syntaxe de l'énumeration

Analyse de la grammaire Nit

Dans la grammaire de Nit, il existe présentement des tokens et des productions pour des enums. Par exemple, dans le fichier nit.sablecc3xx à la ligne 101, il existe une déclaration de token pour les mots-clés d'un enum et à la ligne 236 la déclaration de sa production. La syntaxe que nous allons proposer est semblable à celle d'une déclaration de classe, alors nous ne croyons pas avoir de grandes modifications à la grammaire de Nit. Par contre, il faudra distinguer le 'universal' du 'enum' et faire suivre les changements sur les productions qui utilisaient ses tokens.

Syntaxe proposée

Nous avons fait une analyse des énumerations dans d'autres langages afin de proposer bonne syntaxe tout en suivant le "coding-style" de Nit. Après réflexion, nous proposons la syntaxe suivante:

```
enum <Nom>
    <Clé> [= valeur]
    .
    .
end
```

Pour accéder aux valeurs, il suffit d'écrire la ligne suivante:

```
Nom:clé
```

Voici un exemple:

```
enum A
    b
    d
end
enum B
    b = "je suis un string"
    d = 'C'
end

print B:b // Afficherait "je suis un string"
```

Dans cet exemple, les enums ont des clés de même nom, mais des valeurs différents.

Fonctionalité souhaitable

En pensant à des problèmes dans le dévelopement de logiciels, nous avons pensé à une fonctionalité des énumérations qui serait souhaitable d'implémenter. Prenons le problème de distribution d'un logiciel dans l'écosystème informatique d'aujourd'hui avec plusieurs architectures et systèmes d'exploitation. Il peut rapidement devenir difficile de savoir quelle librairie doit être utilisée pour telle combinaison d'architecture et de OS.

Une solution à laquelle nous avons pensé c'était l'ajout d'une fonction de jonction entre 2 enums. Ceci permetterait d'avoir deux énumerations et de générer un 3e selon le contenu des 2 parents. Voici une syntaxe qui pourrait être considéré pour cette fonctionalité:

```
enum A
    a
end
enum B
    b
end
enum C
    super A * B
end
```

Donc, dans le cas de notre exemple, il serait possible de créer 2 énumerations et de faire la jonction entre les 2.

```
enum Arch
    x86
    amd64
```

```
    ia64
    arm
end
enum Os
    gnu
    bsd
    solaris
    nt
end
enum Libs
    super Arch * Os

    common = "fibonacci,int_stack"
end
```

Ensuite, pour importer la bonne librairie on pourrait faire:

```
if host == Libs:amd64.gnu then
    import amd64_gnu_lib
end

import Libs:common
```

Une difficulté à laquelle nous avons pensé c'est comment baliser le comportement de cette jonction pour éviter que son utilisation devienne un fardeau pour l'utilisateur. Une possible solution auquel nous avons pensé c'est de forcer l'utilisateur qui désire profiter de cette fonctionalité à implémenter certaines fonctions contenus dans une interface. Il faudra aussi considérer une modification de la production "superclass" pour accepter le '*' dans la déclaration.

Étude de l'état de l'art

Nous avons étudier les enums dans plusieurs langages de programmation en identifiant les avantages et désavantages de chacun afin de déterminer la forme la plus souhaitable pour l'implémentation du enum en Nit.

C++ | Objective-C | C

Voici un exemple du enum dans ces langages :

```
enum SEASON {
    SPRING = 0,
    SUMMER = 1,
```

```
    FALL = 2,
    WINTER = 3
};
```

Avantages

- Forme simple et facile à lire
- Possibilité d'assigner des valeurs différentes de celles par défaut
- Possibilité d'assigner un type

Désavantages

- La présence d'accolades qui risque de causer des problèmes en Nit
- Le ';' à la fin qui pourrait être éviter

Ruby

Voici un exemple du enum en Ruby :

```
class Color
    BLUE=1
    RED=2
    GREEN=3
    YELLOW=4
    ORANGE=5
    PURPLE=6
end
```

Avantages

- Forme compact et court
- Possibilité d'assigner des valeurs différentes de celles par défaut
- Style semblable à Nit

Désavantages

- Pas un vrai enum, car il est fait à partir d'une classe

Python

Voici un exemple du enum en Python (vielle version) :

```
from enum import Enum
Animal = Enum('Animal', 'ant bee cat dog')
```

Désavantages

- Peu lisible (trop compact)
- Nécessité d'importer enum
- Impossibilité d'ajouter des valeurs différentes de celles par défaut

Voici un autre exemple du enum en Python (nouvelle version) :

```
from enum import Enum
class Color(Enum):
    blue = 1
    red = 2
    green = 3
    yellow = 4
    orange = 5
    purple = 6
```

Avantages

- Très lisible
- Possibilité d'assigner des valeurs différentes de celles par défaut

Désavantages

- Implémentation en forme de classe
- Nécessité d'importer enum

Java

Voici un exemple du enum en Java :

```
public enum Day {
    SUNDAY = 0, MONDAY, TUESDAY, WEDNESDAY,
    THURSDAY, FRIDAY, SATURDAY
}
```

Avantages

- Très lisible
- Possibilité d'assigner des valeurs différentes de celles par défaut

Désavantages

- La présence d'accolades qui risque de causer des problèmes en Nit

Delphi

Voici un exemple du enum en Delphi :

```
type
    TDay = (Mon=1, Tue, Wed, Thu, Fri, Sat, Sun);   // Enumeration values
var
    day : TDay;           // Enumeration variable
```

Avantages

- Possibilité d'assigner des valeurs différentes de celles par défaut

Désavantages

- Séparation de la déclaration du type et de la variable
- Structure moins lisible que d'autres

Remerciements

Nous aimerons remercier Alexis Laferrière pour son temps, ses conseil sur le fonctionnement interne du langage Nit.