Histoire et développement du C#

Histoire:

C# est un langage assez récent. Il a été disponible en versions bêta depuis l'année 2000 avant d'être officiellement disponible en février 2002 en même temps que la plate-forme .NET de Microsoft à laquelle il est lié

Présentation

C# est un langage de programmation orientée objet, fortement typé, dérivé de C et de C++, ressemblant au langage Java. Il est utilisé pour développer des applications web, ainsi que des applications de bureau, des services web, des commandes, des widgets ou des bibliothèques de classes. En C#, une application est un lot de classes où une des classes comporte une méthode Main, comme cela se fait en Java.
C# est destiné à développer sur la plateforme .NET, une pile technologique créée par Microsoft pour succéder à COM.
Les exécutables en C# sont subdivisés en assemblies, en namespaces, en classes et en membres de classe. Un assembly est la forme compilée, qui peut être un programme (un exécutable) ou une bibliothèque de classes (une dll). Un assembly contient le code exécutable en MSIL, ainsi que les symboles. Le code MSIL est traduit en langage machine au moment de l'exécution par la fonction just-in-time de la plateforme .NET.

.NET

C# est destiné à développer sur la plateforme .NET. Le cœur de cette pile technologique est le framework .NET, composé de:

- Les environnements ASP.NET et Winforms qui servent à exécuter des applications web, resp. de bureau conçus pour la plateforme .NET.
- Une bibliothèque de classes qui permet de manipuler des fichiers, manipuler des tableaux ou des structures en arbres, accéder à Internet, créer des interfaces graphiques, accéder à des bases de données, accéder au registre Windows et beaucoup d'autres choses. La plupart des fonctionnalités sont offertes par des classes de l'espace de noms System.
- Le Common Language Runtime (abr. CLR) est le runtime utilisé par les langages de la plateforme .NET (C#, Visual Basic .NET, J#, etc.), les services fournis par la CLR sont le lancement et l'exécution de programmes, le ramasse-miettes et la gestion d'exceptions. Un programme pour la plateforme .NET est tout d'abord compilé en une forme intermédiaire, le MSIL, puis ce code MSIL est transformé en code machine qui sera exécuté par la CLR. Ce code machine est appelé *managed code* parce que son exécution est sous le contrôle de la CLR.

Un autre produit de la plateforme .NET est l'environnement de développement Visual Studio .NET, outil généralement utilisé pour programmer en C#.

Caractéristiques

C# est un langage dérivé du C++, il apporte un typage sûr, ainsi que les possibilités d'encapsulation, d'héritage et de polymorphisme des langages orientés objet. En C# tous les types sont des objets. Le langage comporte un ramasse-miettes et un système de gestion d'exceptions.

Le typage sûr signifie notamment que les opérations suivantes sont refusées : utilisation de variable non initialisée, tentative d'accéder au-delà des limites d'un tableau, conversions de type dont les résultats ne sont pas prévisibles, dépassement des limites lors d'opérations arithmétiques.

Beaucoup de possibilités de Java se retrouvent dans C# et il y a une forte ressemblance entre un code écrit en C# et le code équivalent en Java.

En C# les variables peuvent être d'un type *référence* ou d'un type *valeur*. Les types valeur sont les types primitifs, les énumérations, les *struct* et les types nullable. Les types référence sont les classes, les interfaces, les tableaux et les *delegate*.

- **Types primitifs**

Les types primitifs sont sbyte, short, int, long, byte, ushort, uint, ulong, char, float, double, decimal et bool.

- **class**

Les constructions les plus fondamentales du langage C# sont les classes. Celles-ci peuvent contenir des constantes, des champs, des propriétés, des indexeurs, des méthodes, des événements, des opérateurs, des constructeurs, des destructeurs ou des sous-classes. Les classes élémentaires sont string et object.

- **struct**

Les struct sont similaires aux classes, mais ce sont des types valeurs et ils ne peuvent pas être hérités.

- **delegate**

Un delegate est une référence à une méthode qui comporte certains paramètres. Les delegate permettent d'assigner des méthodes à des variables et les passer en paramètre.

- **enum**

Un type énuméré est un type valeur qui comporte un lot de constantes. Chaque type énuméré a un type sous-jacent : un type primitif déterminé en fonction des valeurs des constantes.

- **type nullable**

Les nullable sont des types primitifs qui peuvent en plus avoir la valeur *null*. Chaque type primitif T a un type nullable associé T?. Par exemple une variable de type int? peut contenir un int ou null.

Syntaxe

Voici un “Hello World!” en C#:

```
1  using System;
2
3  public class HelloWorld
4  {
5      public static void Main(string[] args)
6      {
7          Console.WriteLine(‘Hello World!’);
8      }
9  }
```

Différences avec le C

Le langage compte un certain nombre de changements par rapport au C/C++ ; on notera particulièrement les points suivants :

- la manipulation directe de pointeurs ne peut se faire qu'au sein d'un code marqué *unsafe*, et seuls les programmes avec les permissions appropriées peuvent exécuter des blocs de code *unsafe*. La plupart des manipulations de pointeurs se font via des références sécurisées, dont l'adresse ne peut être directement modifiée, et la plupart des opérations de pointeurs et d'allocations sont contrôlées contre les dépassements de mémoire. Les pointeurs ne peuvent pointer que sur des *types de valeurs*, les types *objets*, manipulés par le ramasse-miettes, ne pouvant qu'être référencés ;
- les objets ne peuvent pas être explicitement détruits. Le ramasse-miettes s'occupe de libérer la mémoire lorsqu'il n'existe plus aucune référence pointant sur un objet. Toutefois, pour les objets gérant des types non managés, il est possible d'implémenter l'interface `IDisposable` pour spécifier des traitements à effectuer au moment de la libération de la ressource ;
- l'héritage multiple de classes est interdit, mais une classe peut implémenter un nombre illimité d'interfaces, et une interface peut hériter de plusieurs interfaces ;
- le C# est beaucoup plus typé que le C++ ; les seules conversions implicites sont celles entre les différentes gammes d'entiers et celles d'un type dérivé à un type parent. Aucune conversion implicite n'a lieu entre booléens et entiers, entre membres d'énumération et entiers, ni de pointeurs sur un type *void* (quoique pour ce dernier point l'utilisation de références sur le type Object permette d'obtenir le même effet). Les conversions définies par l'utilisateur peuvent être définies comme implicites ou explicites ;
- la syntaxe pour la déclaration des tableaux n'est pas la même : `int[] a = new int[5]` remplace `int a[5]`. Car il s'agit d'une allocation dynamique, `int[] a` étant la déclaration d'une référence (nulle si non initialisée). L'allocation manuelle d'un tableau sur la pile reste cependant possible avec le mot clé `stackalloc` ;
- les membres d'une énumération sont rassemblés dans leur propre espace de noms ;
- le C# ne gère pas les *templates*, mais cette fonctionnalité a été remplacée par les types génériques apparus avec C# 2.0 ;
- les propriétés ont été introduites, et proposent une syntaxe spécifique pour l'accès aux données membres (ainsi que la facilitation de l'accès simultané par plusieurs *threads*) ;
- la réflexion totale des types est disponible ;
- les délégués, qui sont des listes de pointeurs sur fonctions, sont utilisés notamment pour la programmation événementielle.

Différences avec le Java

Bien que le C# soit similaire à Java, il existe des différences notables, par exemple :

- Java n'autorise pas la surcharge des opérateurs ;
- Java a des exceptions vérifiées, alors que les exceptions du C# ne sont pas vérifiées, comme en C++ ;
- Java permet la génération automatique de la documentation HTML à partir des fichiers sources à l'aide des descriptions Javadoc-syntax, tandis que le C# utilise des descriptions fondées sur le XML ;
- Java n'a pas de langage préprocesseur ;
- C# prend en charge les indexeurs, les méthodes déléguées, et les événements (là où Java se contente du patron de conception Observateur) ;
- C# ne prend pas en charge les implémentations anonymes d'interfaces et de classes abstraites ;
- C# ne prend en charge que les classes internes statiques ;
- C# prend en charge les structures en plus des classes (les structures sont des types *valeur* : on stocke le contenu et non l'adresse) ;

- C# utilise une syntaxe intégrée au langage (DllImport) et portable pour appeler une bibliothèque native, tandis que Java utilise *Java Native Interface* ;
- C# intègre la généricité, et la machine .NET a été modifiée pour permettre cela (Java l'intègre également, mais son implémentation a été réalisée dans le compilateur javac sans changer le bytecode Java). Plus de détails sur l'aspect théorique de cette réalisation peuvent être trouvés dans la référence, diapositives 70 à 89.

Pourquoi apprendre le C#:

il y a de multiples raisons d'apprendre le C# :

- C'est un langage très utilisé en entreprise avec beaucoup plus de demande de développeurs par les entreprises que d'offre. Cela induit un choix du travail par le développeur et des rémunérations très intéressantes aussi ! Au niveau des salaires un débutant est à 30–33k€ , avec 5 ans d'expérience 40k–50k€, 10 ans et plus 50k-60k€ à peu près en région parisienne.
- La raison pour laquelle il est très demandé en entreprise est parce qu'il est versatile :
    - On peut faire des sites web rapidement avec C# ASP NET MVC (j'ai même un cours pour apprendre à créer des sites web en 5 heures sur Udemy, cf lien ci-dessous)
    - On peut faire des applications de bureau avec WPF
    - On peut faire des applications mobiles multiplateforme (Apple + Android) avec Xamarin.
    - Des technologies comme EntityFramework ou LINQ permettent de manipuler facilement des données de base de données avec du C#
    - Développer des jeux vidéo avec Unity
- C'est un langage qui a bien pris en compte les problématiques actuelles du développement d'où sa popularité. Développer du code maintenable et évolutif est relativement aisé.
- Du fait de sa popularité, il y a beaucoup de questions répondues sur Stackoverflow et Google, il est donc facile de résoudre des erreurs en C#. A chaque fois que j'ai un problème, je regarde sur Google et il y a au moins 100 autres développeurs qui ont eu le même souci et qui ont écrit la réponse à mon problème !