AIDE-MÉMOIRE

C#

Christophe **Pichaud**

AIDE-MÉMOIRE C#

DUNOD

Mise en page : Belle Page

© Dunod, 2021
11 rue Paul Bert, 92240 Malakoff
www.dunod.com
ISBN 978-2-10-081322-3

Table des matières

Ce livre est dédié à mes parents Jean-Marc et Mireille, qui ont toujours cru en moi et m'ont toujours encouragé et conseillé.

Les noms de mes filles sont partout dans les exemples de code : Edith (Didou), Lisa (le mini-mini) et Audrey (Maggie).

Petit Lulu, avec ce deuxième bouquin, on ira aux States !

Préface

Lorsque j'ai commencé le développement il y a plus de trente ans, un de nos enjeux était de faire tourner notre application dans un minimum de mémoire. 640 Ko, oui vous lisez bien, Ko, même pas Mo, alors Go n'en parlons pas. Nous étions principalement à la recherche des fuites mémoires, par exemple à savoir si notre destructeur avait bien été appelé. Et même si nous disposions d'outils performants de détection, certaines conditions exceptionnelles faisaient que nous avions toujours des problèmes de fuites mémoires.

Quand un nouveau langage C# en bêta est arrivé chez Microsoft en 1999, (nom de code : Cool), mon premier réflexe a été de coder une classe, avec un constructeur, et ce qui s'apparentait pour moi à l'époque, à un destructeur. Rien, nada, aucun passage dans le destructeur. Erreur de débutant, même pour un développeur chevronné, Eric, RTFM quoi ! (si vous ne connaissez pas je vous laisse découvrir le sens de cette expression).

Si j'avais lu attentivement la documentation, j'aurais appris que tous les objets .NET sont des références (pas tous en réalité), et non des pointeurs, qu'ils sont assujettis à un ramasse-miette, qui ira les collecter en temps voulu. Merci .NET et C#, vous m'enlevez une épine dans le pied, je vais pouvoir me focaliser réellement sur la valeur de mon logiciel et non plus sur de la plomberie.

Néanmoins, est-ce que cela veut dire que nous ne devons plus y prêter attention aujourd'hui ?

Même si c'est une avancée, il est toujours important d'éviter le gaspillage. Connaître ce qu'on consomme comme ressources et comment les libérer, reste une très bonne pratique, d'autant plus que les enjeux pour les développeurs se sont déplacés aujourd'hui dans le cloud.

Un mauvais code peut faire consommer plus de puissance CPU, plus de mémoire, et croyez-moi sur parole, cela a pour conséquence un impact non seulement sur la carte bleue, mais également sur l'empreinte énergétique.

Développer pour le cloud implique également de prendre conscience que certains paradigmes changent. Le plus significatif est celui de la disponibilité des ressources qui ne seront jamais garanties à 100 %, mais par exemple à 99,95 % (pour tout un tas de raisons que je ne détaillerai pas ici), différence infime, mais qu'il faudra prendre en compte. En tant que développeur, il faudra passer aux modèles, retry, circuit breaker, gatekeeper, etc. et C# et les librairies .NET vous y aideront.

Un autre aspect important de la plateforme .NET était la promesse de sa portabilité. En 2016, Microsoft rachète Xamarin, la société fondée par Miguel de Icaza. Xamarin avait déjà implémenté une solution Mono, basée sur le *Common Language Infrastructure* (spécification ouverte de la plateforme .NET) pour que Microsoft .NET Core naisse et fonctionne sur des systèmes autres que Windows.

Mais venons-en à la présentation de cet *Aide-mémoire C#*. Comme à son habitude, Christophe Pichaud nous délivre un ouvrage complet et détaillé qui vous aidera à comprendre la philosophie de la plateforme et du langage C#.

Certaines notions abordées ne seront pas utiles pour tout le monde, mais elles pourront sans doute aider lors de l'analyse et le débogage.

Savoir par exemple qu'il est possible de faire de l'interopérabilité entre du code C/C++ et C# est un plus pour éviter de recoder une librairie C qui fonctionne à merveille depuis des années.

Pour un développeur Linux, savoir que nous avons désormais la possibilité d'avoir des containers Windows peut sembler étrange, mais ceux-ci sont les garants d'une compatibilité avec du code existant.

Désormais, le même code peut tourner sur Windows et sur Linux, nous pouvons enfin laisser derrière nous les anciennes querelles de clocher,

et nous concentrer sur ce que nous aimons le plus : le développement et réunir des développeurs de mondes différents.

La large diffusion de .NET sous toutes ses formes, ou simplement celle de son compilateur « portable », est donc la garantie d'une normalisation certaine de son implantation. Elle ne peut laisser indifférent l'industriel, l'ingénieur ou le développeur soucieux de la propagation et de la mise en valeur rapide de ses produits ou de ses idées, à grande échelle.

Eric Vernié

Cloud Solution Architect chez Microsoft France

Avant-propos

À propos de l'auteur

Christophe Pichaud est un développeur logiciel français basé à Paris. Dans sa carrière, il a construit de grosses infrastructures bancaires, participé à l'ouverture de la première banque en ligne (Banque Populaire) et à la construction des services bancaires pour 2 500 agences Société Générale (MAIA, URTA). Il réalise aussi des migrations C++ et implémente des applications hybrides avec la stack Microsoft .NET. Ses clients passés sont Accenture, Avanade, Microsoft, Sogeti, Capgemini, Palais de l'Élysée (présidence de la République), SNCF, Total, Danone, CACIB et BNP Paribas.

Il est certifié MCSD et MCSD.NET. De plus, il participe aux événements Microsoft en tant que speaker (TechDays, DevDays) et MVP sur les stands Ask The Expert. Il est MVP[1] depuis 2018 dans la catégorie Developer Technologies. C'est un contributeur régulier au magazine *Programmez* depuis 2011. Il est aussi le Community Manager des « .NET Azure Rangers », qui contient 26 membres dont 8 MVP et dont les activités sont l'animation de sessions techniques, l'écriture d'articles techniques et la promotion des technologies Microsoft ou Cloud. Christophe travaille chez **Infeeny**[2], une filiale du groupe Econocom spécialisée dans les Technologies Microsoft.

1 https://mvp.microsoft.com/en-us/PublicProfile/5003105?fullName=Christophe%20Pichaud.

2 https://infeeny.com/.

Pourquoi lire ce livre ?

Depuis le temps que je lis des livres sur C# et le Microsoft .NET Framework, je suis attristé de voir qu'il en existe si peu en français – on peut les compter sur les doigts d'une main – tandis qu'en anglais on en trouve des dizaines. En octobre 2019 Dunod me contacte pour savoir si je connais quelqu'un pouvant rédiger *Aide-mémoire C#*. Je passe le message sur le canal Microsoft MVP et trouve un collègue MVP qui, malheureusement se désiste quelques semaines plus tard. C'est ainsi que m'est venue l'idée de le remplacer…

Ce livre se veut un condensé de technologies autour des sujets suivants :

- la compréhension du CLR ;
- la compréhension du framework et des classes de la BCL (*Base Class Library*) ;
- la maîtrise du langage C# ;
- la maîtrise de l'environnement Visual Studio.

À qui s'adresse ce livre ?

Ce livre s'adresse aux développeurs ayant des connaissances de base en programmation et s'intéressant au monde des technologies .NET de Microsoft et plus particulièrement au langage C#. Cet ouvrage est comme un ensemble de recettes, vous pouvez lire les chapitres dans n'importe quel ordre car il est organisé par thèmes.

Structure du livre

Le livre est organisé en trois parties :

- partie I : la plateforme .NET ;
- partie II : le langage C# ;
- partie III : le framework .NET et .NET Core.

Exemples de code et prérequis

Les exemples de code sont en téléchargement libre depuis l'URL suivante : https://github.com/ChristophePichaud/Aide-Memoire-CS-NET.

À propos des compilateurs

À l'heure où ce livre est écrit, la version de Visual Studio est la v.16.6.5.

Note de l'auteur

"C# is a cornerstone of Microsoft's new .NET platform. Inheriting many features from both Java and C++, C# is destined to become the high-level programming language of choice for building high-performance Windows and Web applications and components – from XML based Web services to middle-tier business objects and system-level applications[1]*."*

Stanley B. Lippman[2], 2001

C# est un langage moderne, élégant, productif, orienté objet qui permet de faire des applications mobiles, du web, du desktop, du cloud, du gaming, de l'IoT et de l'IA.

J'ai assisté en 2001 à San Francisco à la conférence .NET organisée par DevelopMentor durant laquelle les équipes de Microsoft présentaient C# et .NET Framework. C'était COM 3.0. À l'époque, on développait en Visual Basic 6 et en Visual C++ et Microsoft nous présentait le futur du développement selon Microsoft. On sentait qu'il y avait un truc spécial. Le monde

1 C # est la pierre angulaire de la nouvelle plateforme .NET de Microsoft. Héritant de nombreuses fonctionnalités de Java et C ++, C # est destiné à devenir le langage de programmation de haut niveau de choix pour la création d'applications et de composants Windows et Web hautes performances – des services Web XML aux objets métier de niveau intermédiaire et au niveau des applications système.

2 Stanley B. Lippman a fait l'implémentation du premier compilateur C++ chez Bell Laboratories avec Bjarne 'La même année, il travaille chez Microsoft comme architecte dans l'équipe Visual C++.

allait changer. C'était génial. Je suis reparti au bout d'une semaine avec mon Visual Studio .NET BETA et le .NET Framework SDK BETA2 DVD en poche ainsi qu'avec le *MSDN Magazine*.

À l'époque, je travaillais à la Société Générale où j'ai présenté le truc. On faisait du C++, des composants COM avec ATL/COM, des UI avec les MFC et de la communication client/serveur avec des sockets, des Named Pipes, du MQ Series et du Tuxedo. J'explique à mes collègues que .NET c'est l'avenir et qu'il va falloir s'y mettre. Je reste à la SG faire du C++ pendant quelques années et en 2005 je rentre chez Avanade où on ne fait que du .NET. C'était hallucinant : on faisait des SI entiers en .NET avec des Web Services, des sites Web ASP.NET et on utilisait Enterprise Library car Avanade avait une librairie ACA.NET pour le développement dédié aux entreprises. Chez Avanade, joint-venture Microsoft/Accenture, on avait un accès direct aux ressources Microsoft et on avait plein de documents confidentiels (NDA[1]) et des slides de conférences. C'était l'âge d'or de .NET. À l'époque, Visual Studio 2005 était d'une rapidité phénoménale. On faisait des interfaces graphiques de bureau avec WinForms, des applications Web avec ASP.NET WebForms et des Web Services ASMX. On se connectait à SQL Server ou Oracle avec ADO.NET et on faisait du Reporting avec Crystal Report qui était livré avec Visual Studio. Avec Avanade, on a fait des applications de folie pour tout le CAC40. Avec Enterprise Library et ACA.NET, on avait une longueur d'avance et on développait beaucoup en réutilisant les briques.

En 2006, Vista est sorti et ce fut le traumatisme Longhorn[2]. Vista devait contenir beaucoup de .NET à l'intérieur de Windows. C'était le projet WinFX. Or WinFX, c'était Avalon (WPF), Indigo (Windows Communication Foundation) et WWF (Windows Workflow Foundation), un système de fichiers entièrement objet nommé WinFS mais… la guerre entre la Windows Division qui fait Windows et la Developer Division qui fait .NET a eu raison du projet. WinDiv reprochait à .NET un manque de fiabilité, de rapidité et de robustesse. Ils ont envoyé une liste des requêtes pour .NET et DevDiv n'a jamais pu y répondre et donc WinDiv les a ignorés et a stoppé

1 *Non Disclosure Agreement* : accord de confidentialité.

2 https://arstechnica.com/information-technology/2011/06/windows-8-for-software-developers-the-longhorn-dream-reborn/.

le projet. Windows a été entièrement fait en CC++ sans .NET à l'intérieur. Sachez avant de clore cette parenthèse que Windows est toujours exclusivement fait en C++. J'ai continué à faire des gros projets .NET pour les grands comptes. J'ai participé à un énorme projet mondial Exchange avec Microsoft Corp pour Orange. Dans ce projet .NET était utilisé pour le provisioning des serveurs Windows et Exchange. Dans les années 2010, les entreprises ont massivement migré leur code Visual Basic et C++ en .NET C# ou .NET VB.NET. A cette époque les annonces d'emplois pleuvent et les salaires de développeur .NET sont attractifs. En 2011, je fais la connaissance de François Merand, qui a passé dix ans chez Microsoft à l'architecture, et il fonde les « .NET Rangers by Sogeti ». Je signe chez lui pour faire du consulting .NET sous la responsabilité de Namic Ait-Meddour avec une dizaine d'individus dont Keelan Clech à Lyon, des ex-Winwise comme Jason De Oliveira, Loïc Baumann, Guillaume Rouchon et Vincent Labatut à Paris. Nous sommes des experts Cloud, .NET et ALM. C'est le début de Azure et ALM, c'est la gestion du cycle de vie des applications (on appelle ça le DevOps maintenant) avec TFS. On était des experts communicants ; on faisait des articles, du blog, des conférences, des vidéos, des missions, des events Microsoft comme les TechDays. Et en 2017, après un an à développer une application GUI multi-threads pour des traders en WinForms, je suis débauché par Microsoft pour mes compétences C++. Je fais du C++ et du .NET.

Moi qui suis un fan de C++, je dois reconnaître que quand je fais du C# / .NET, tout est trop simple. Une classe, un fichier, un document XML ou JSON, une page Web, un serveur Web API, une connexion SQL Server, tout est simple. Avec .NET, celui qui veut faire une application peut le faire avec une productivité inégalée. Une interface graphique en WinForms c'est aussi rapide qu'en VB6. On pose les contrôles Windows, on gère les événements et c'est fini. Pour sérialiser une classe, on ajoute un attribut au-dessus de la classe et c'est terminé, c'est automatisé. Quand je vois les efforts qu'il me faut faire en C++ pour avoir une application de sérialisation multiplateforme avec la STL et le support XML avec Boost, c'est l'enfer. De plus, en C# il existe des librairies tierces qui permettent de fournir des contrôles UI pour que les applications soient magnifiques. Par exemple :

- Telerik ;
- SyncFusion (mon produit préféré) ;

- DevExpress ;
- Infragistics.

Bref, faire une application .NET avec C#, c'est simple, ludique, rapide et ça marche plutôt rapidement. Le moteur JIT est fourni par le backend de Visual C++. On a donc une rapidité de folie. Soyons honnête, le code C# .NET s'exécute forcément moins rapidement que C++ qui génère de l'assembleur ×64 optimisé mais le résultat est bluffant si l'on considère la productivité accrue de l'environnement. Il est vrai que les applications de communications boîtes noires seront toujours faites en C++ ou en Go, mais les Web API que l'on développe dans les systèmes Azure Kubernetes sont faites en C# .NET Core sous Linux.

.NET Core est multiplateforme. Il fonctionne sous Windows, Linux et Mac. Il est open-source et Microsoft accepte les contributions de la communauté.

La technologie .NET est passionnante à explorer car elle est le fruit d'une rivalité entre Microsoft et Sun. Ce dernier avait sorti Java et Microsoft a essayé de faire des extensions à Java *via* Visual J++. Microsoft a écopé d'un procès et s'est rétracté. C# et .NET étaient nés. Microsoft ne veut pas le reconnaître, mais .NET, c'est le Java de Microsoft, c'est la même architecture : un *framework* avec une *Base Class Library*, un *byte code* (le MSIL), une machine virtuelle d'exécution et un moteur JIT d'émission de code. L'avantage de .NET et C#, c'est que Microsoft possède toute la stack alors que Java est maintenant la propriété d'Oracle et que le modèle open-source et les conflits entre Oracle et Google ne présument pas de bonnes choses pour Java en libre. Les entreprises qui font du Java sont obligées de souscrire un contrat de support à Oracle pour faire des applications d'entreprises, et ce n'est pas donné ! Cependant, il existe des implémentations libres du JDK qui permettent de s'affranchir d'Oracle comme OpenJDK ou Azul.

Un DSI qui veut faire une application ou un SI avec une technologie a le choix entre .NET Core et Java. Microsoft ou Oracle/Google. Les deux stratégies sont payantes.

L'avenir c'est .NET Core.

Christophe Pichaud (christophep@cpixxi.com),

Paris, France. Mardi 14 Juillet 2020

Remerciements

Ce livre a été relu et corrigé par de nombreux amis et relations de travail.

Je tiens à remercier Bruno Barthère, Eva Delord, Sébastien Barré, Jérémy Jeanson, Cédric Michel, Cédric Leblond, Frédéric Breton, Cheick Diaby, Pascal Roze, Sylvain Pontoreau, Richard Clark et Eric Vernié.

Merci à mon éditeur Jean-Luc Blanc des éditions Dunod qui m'a fait confiance une deuxième fois après l'*Aide-Mémoire C++*. J'espère que nous allons continuer sur d'autres projets. Merci aussi à Maxine Pouzet de chez Dunod pour sa patience et ses relectures.

Introduction à C#

Comme tout bon ouvrage de développement logiciel, nous allons commencer par coder le fameux « hello world ! ».

```
using System;


namespace ConsoleApp1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World !»);
        }
    }
}
```

La première ligne indique que nous allons utiliser le namespace System qui est le namespace principal du framework .NET.

Ensuite, on trouve une fonction statique main dans la déclaration d'un namespace ConsoleApp1 qui est le point d'entrée de l'application. Console. WriteLine fait un affichage sur la sortie standard.

Pour compiler ce programme, on utilise Visual Studio et un projet console .NET Framework. On fait CTRL + F5, ça compile et ça tourne !

1

La plateforme .NET

Microsoft dévoile .NET Framework en 2002 avec deux composants majeurs :

- le CLR (*Common Language Runtime*) ;
- le .NET *Framework Class Library.*

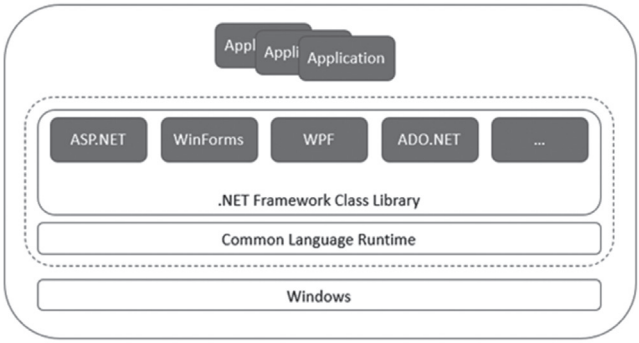

Figure 1.1 La plateforme Microsoft .NET

Le .NET Framework est la nouvelle stratégie de Microsoft pour écrire des applications Windows.

Quel que soit le langage utilisé (C#, VB.NET, F#, C++/CLI), les compilateurs génèrent des metadonnées et du code intermédiaire MSIL (*Microsoft Intermediate Language*) qui est ensuite JIT-compilé (*Just-in-time compilation*) et exécuté par la machine virtuelle CLR. Quel que soit le langage utilisé, les applications .NET ont toutes le même niveau de performance.

Les metadonnées sont des informations contextuelles (informations de sécurité, noms des éléments, informations sur les dépendances, versions des différents modules).

Les compilateurs produisent des modules managés (exe, dll) appelées *assemblies*. Une assembly contient du code MSIL, des metadonnées et des ressources (jpg, gif, html, etc.). Les assemblies sont des modules dits « managés » car à l'exécution, le CLR traque les différents éléments en mémoire et les libère au fur à mesure : c'est le travail du garbage collector.

Une assembly est autodescriptive et son déploiement nécessite le plus souvent une simple opération XCOPY et il n'y a pas besoin de la base de registres ou de l'Active Directory pour obtenir des informations supplémentaires.