Veille technologique sur le langage orienté objet C#

SOMMAIRE

Table des matières

Son Histoire :

Le C# est un langage de programmation orientée objet, commercialisé par Microsoft depuis 2002 et destiné à développer sur la plateforme Microsoft .NET. Ce langage est un dérivé du C et du C++ et s'est fortement inspiré du langage Java dans sa structure, 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.

Son évolution :

Version 1.0 :

La version 1.0 est la toute première sortie en 2002, elle était conçue pour le .NET Framework 1.0 et 1.1

Version 2.0 :

Mit à disposition du public en octobre 2005 par Microsoft, après une longue période de beta-tests, la version 2.0 de la bibliothèque .NET, accompagnée d'une nouvelle version de la quasi-totalité des outils associés est proposé. Le C# ne fait pas exception à la règle et sort donc en version 2.0 pour accompagner ces changements.

Les principales modifications de cette version sont :

- Les classes partielles, permettant de répartir l'implémentation d'une classe sur plusieurs fichiers.
- Les types génériques, qui ne sont pas une simple copie des templates C++. Par exemple, on trouvera dans les génériques C# la restriction de types.
- Un nouvel itérateur qui permet l'utilisation de coroutines via le mot-clé yield, équivalent du yield que l'on trouve en Python.
- Les méthodes anonymes avec des règles de fermeture configurables.
- Les types « nullables », c'est-à-dire la possibilité de spécifier qu'un type de valeur peut être nul. Ceux-ci sont déclarés avec le caractère point d'interrogation « ? » suivant le nom du type, comme ceci : int? i = null;

- Le nouvel opérateur double point d'interrogation « ?? » utilise deux opérandes et retourne le premier non nul. Il a été introduit pour spécifier une valeur par défaut pour les types « nullables ».

La fonctionnalité des types nullables fut corrigée quelques semaines seulement avant la sortie publique de la version 2.0, car il a été mis en lumière que si la valeur de la variable était bien nulle, cette variable n'était pas nulle au sens traditionnel du terme, c'est-à-dire qu'il ne s'agit pas d'une référence vide. Ainsi, la conversion d'un type primitif de valeur nulle en objet donnait une référence non nulle vers une valeur nulle. Il fallut donc, pour corriger ce problème, corriger le noyau du CLR et effectuer de nombreuses vérifications et corrections sur tous les produits de la gamme .NET 2.0 (Visual Studio 2005, SQL Server 2005, C# et VB.NET).

Version 3.0 :

La version 3.0 est sortie pour la version 3.5 du .NET Framework. Sorite en 2007, la principale nouveauté est l'incorporation du LINQ pour « Language integrated queries ». L'apport de cette nouveauté permet de faire énormément de choses en matière de traitement sur les objets de type collection. Les principales modifications en dehors du LINQ sont :

- Nouvelle possibilité d'initialisation d'un objet : À la place de Client c = new Client(); c.Nom = "Dupont";, on peut utiliser Client c = new Client{ Nom = "Dupont" };

- Expressions lambda : ListeTrucs.Where(delegate(Truc x) { return x.Size > 10; }); devient ListeTrucs.Where(x => x.Size > 10);

- Introduction des types anonymes : var x = new { Nom = "Dupont" } peut être utilisé à la place de class __anonymous { private string _nom; public string Nom { get { return _nom; } set { _nom = value; } } } __anonymous x = new __anonymous(); x.Nom = "Dupont";
- Méthodes étendues : permet d'ajouter des méthodes à une classe en y ajoutant un premier paramètre this.

Le code compilé en C# 3.0 est entièrement compatible avec celui du 2.0, étant donné que les améliorations apportées ne sont que purement syntaxiques ou ne consistent qu'en des raccourcis compensés au moment de la compilation. Les nouveautés introduites dans les bibliothèques de la version 3.5 (LINQ, etc.) ne sont cependant pas utilisables avec les versions précédentes de C#.

Cette version exige Windows XP ou une version supérieure (Vista ou Windows 7). Elle n'est pas disponible pour Windows 2000.

Version 4.0 :

Les principales nouveautés de la version 4.0 qui est sortie en même temps que le .NET 4.0 en 2010 sont :

- Le typage dynamique des variables à l'aide du mot clé dynamic ;
- Les arguments nommés et facultatifs ;
- La prise en charge de la covariance et de la contravariance pour les interfaces et les délégués génériques.
- La nouvelle bibliothèque parallèle : Task Parallel Library ;
- Une version optimisée de la plate-forme entité d'accès aux bases de données (Entity Framework) via l'utilisation de LINQ ;
- La version parallèle de LINQ appelée PLINQ.

Version 5.0 :

La version 5.0 est sortie en 2012 avec la version 4.5 du .NET. Cette version permet de programmer plus simplement des programmes asynchrones grâce à l'ajout des mots clés async et await.
Également, le comportement des closures dans la boucle foreach a été modifié. Il n'est désormais plus nécessaire d'introduire une variable locale dans une boucle foreach pour éviter les problèmes de closure.
À noter également les informations relatives à l'appelant permettant de connaître le nom de la méthode qui a appelé une propriété.

Version 6.0 :

Cette version est sortie en 2015 avec le .NET 4.6, elle apporte plusieurs modifications.
Les propriétés implémentées automatiquement (ou propriétés automatiques) sont apparues en C# 3, pour simplifier la déclaration de propriétés qui se contentent d'encapsuler l'accès à des champs. Bien qu'elles permettent de rendre le code plus concis, elles présentent un inconvénient : il n'est pas possible de les initialiser au niveau de la déclaration, il faut forcément le faire dans le constructeur. De plus, il n'est pas possible de faire des propriétés automatiques en lecture seule, puisqu'elles n'ont pas de mutateur (setter) et on ne pourrait donc pas leur affecter de valeur.

C# 6 remédie à ce problème en permettant d'initialiser les propriétés automatiques au niveau de la déclaration

Cette version implémente aussi la compatibilité avec linux avec une version du langage qui lui est réservé.

Version 7.0 :

Compatible pour les versions égales ou supérieures au .NET 4.5, cette version sortie en 2016 se veut de simplifier le code avec une meilleurs optimisation des performances avec l'ajout de plusieurs petites nouveautés, voici les principales :

- Variable de sortie : il est désormais possible de déclarer une variable de sortie (avec le out) directement dans les arguments de la méthode appelé, évitant ainsi de devoir les déclarer au préalable.

- Correspondance de modèle : introduit la notion de patterns, qui, abstraitement parlant, sont des éléments syntaxiques qui peuvent tester qu'une valeur a une certaine "forme" et extraire des informations de la valeur quand c'est le cas.

- Tuples : Il est courant de vouloir renvoyer plusieurs valeurs d'une méthode. Les options disponibles dans les anciennes versions de C# ne sont pas optimales. Pour faire mieux, C# 7.0 ajoute des « types » de tuple et des « littéraux » de tuple. L'appelant de la méthode recevra un tuple et pourra accéder aux éléments individuellement.

- Fonctions locales : Parfois, une fonction d'assistance n'a de sens qu'à l'intérieur d'une seule méthode qui l'utilise. On peut maintenant déclarer de telles fonctions dans d'autres corps de fonction en tant que fonction locale

Version 8.0 :

Sortie en 2019 cette version apporte surtout le modificateur de membre ReadOnly qui permet de spécifier que ce membre que peut pas être définie ou redéfinie ailleurs que dans le contrôleur de la classe. Elle ajoute également :

- Méthodes d'interface par défaut : On peut désormais ajouter des membres aux interfaces et fournir une implémentation pour ces membres. Cette fonctionnalité de langage permet aux auteurs d'API d'ajouter des méthodes à une interface dans les versions ultérieures sans pour autant nuire à la compatibilité des binaires ou des sources avec les implémentations existantes de cette interface. Les implémentations existantes héritent de l'implémentation par défaut.

- Expressions switch : Souvent, une instruction switch produit une valeur dans chacun de ses blocs case. Les expressions switch permettent d'utiliser une syntaxe d'expression plus concise, comprenant moins de mots clés case et break répétitifs et moins d'accolades.

- Modèles de tuples : Certains algorithmes dépendent de plusieurs entrées. Les modèles de tuples permettent de basculer des unes aux autres en fonction de plusieurs valeurs exprimées sous forme de tuple.

- Déclarations using : Une déclaration using est une déclaration de variable précédée par le mot clé . Elle indique au compilateur que la variable déclarée doit être supprimée à la fin de la portée englobante.

Version 9.0 :

Sortie en 2020, cette version en conçu pour fonctionner sur le .NET 5. Parmi les principales nouveautés nous avons :

- Types d'enregistrements : on peut utiliser le mot clé record pour définir un type de référence qui fournit des fonctionnalités intégrées pour l'encapsulation des données.
- Immuabilité : Un type d'enregistrement n'est pas nécessairement immuable. Vous pouvez déclarer des propriétés avec des set accesseurs et des champs qui ne le sont pas readonly

. Toutefois, bien que les enregistrements puissent être mutables, ils facilitent la création de modèles de données immuables. Les propriétés que vous créez à l'aide de la syntaxe positionnel sont immuables.

- Égalité des valeurs : L'égalité des valeurs signifie que deux variables d'un type d'enregistrement sont égales si les types correspondent et que toutes les valeurs de propriété et de champ correspondent. Pour les autres types de référence, l'égalité correspond à l'identité. Autrement dit, deux variables d'un type référence sont égales si elles font référence au même objet.
- Héritage : Un enregistrement peut hériter d'un autre enregistrement. Toutefois, un enregistrement ne peut pas hériter d'une classe, et une classe ne peut pas hériter d'un enregistrement.
- Setter init uniquement : on peut créer init des accesseurs au lieu d'un set accesseurs pour les propriétés et les indexeurs. Les appelants peuvent utiliser la syntaxe de l'initialiseur de propriété pour définir ces valeurs dans les expressions de création, mais ces propriétés sont en lecture seule une fois que la construction est terminée. Les Setters init uniquement fournissent une fenêtre pour modifier l'État.
- Performances et interopérabilité : Trois nouvelles fonctionnalités améliorent la prise en charge de l'interopérabilité native et des bibliothèques de bas niveau qui requièrent des performances élevées : les entiers de taille native, les pointeurs de fonction et l'omission de le localsinit indicateur.

Version 10 :

Cette version est sortie en novembre 2021 pour le .NET 6. Elle comprend les principales modifications suivantes :

- Structures d'enregistrement : Vous pouvez déclarer des enregistrements de type valeur à l'aide des déclarations ou readonly record struct. Vous pouvez maintenant clarifier le fait qu'un record est un type référence avec le record class déclaration.
- Gestionnaire de chaînes interpolées : on peut créer un type qui génère la chaîne résultante à partir d'une expression de chaîne interpolée. Les bibliothèques .NET utilisent cette fonctionnalité dans de nombreuses API. Vous pouvez en créer un en suivant ce didacticiel
- Directives using globales : Vous pouvez ajouter le global modificateur à une global pour indiquer au compilateur que la directive s'applique à tous les fichiers sources de la compilation. Il s'agit généralement de tous les fichiers sources d'un projet.
- Améliorations des expressions lambda : Ces nouvelles fonctionnalités rendent les expressions lambda plus similaires aux méthodes et aux fonctions locales. Elles facilitent l'utilisation d'expressions lambda sans déclarer une variable d'un type délégué, et elles fonctionnent de manière plus transparente avec la nouvelle ASP.NET Core les api minimales.
- Autoriser l'attribut AsyncMethodBuilder sur les méthodes : Dans C# 10 et versions ultérieures, vous pouvez spécifier un autre générateur de méthode Async pour une méthode unique, en plus de spécifier le type de générateur de méthodes pour toutes les méthodes qui retournent un type de tâche spécifique. Un générateur de méthode Async personnalisé active des scénarios de réglage des performances avancés dans lesquels une méthode donnée peut bénéficier d'un générateur personnalisé.

Version 11 :

Cette version est toujours pour la version 6 du .NET. La version stable officiel n'est pas encore rendue disponible au grand publique mais la documentation de certaines nouveautés est déjà fournie par microsoft. Les nouveautés connus pour le moment sont :

- Membres abstraits statiques dans les interfaces : Vous pouvez ajouter des membres abstraits statiques dans des interfaces pour définir des interfaces qui incluent des opérateurs surchargeables, d'autres membres statiques et des propriétés statiques. Le scénario principal de cette fonctionnalité consiste à utiliser des opérateurs mathématiques dans des types génériques. l'équipe du runtime .net a inclus des interfaces pour les opérations mathématiques dans le package System. runtime. expérimental NuGet.
- Attributs génériques : Vous pouvez déclarer une classe générique dont la classe de base est . Cela fournit une syntaxe plus pratique pour les attributs qui requièrent un System.Type paramètre. Auparavant, vous deviez créer un attribut qui utilise Type comme paramètre de constructeur