Оглавление

1. История и недостатки COM

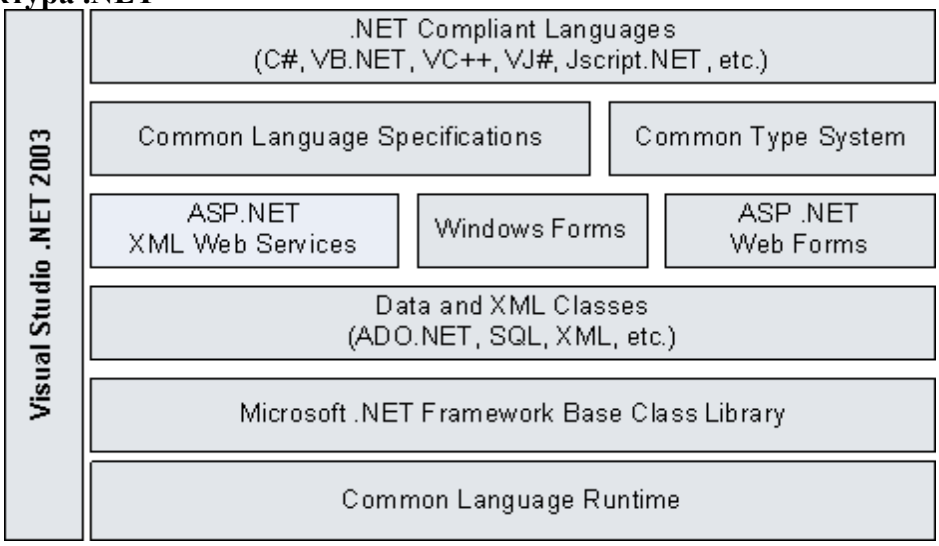
Технология COM (Component Object Model — модель компонентных объектов) позволяла строить библиотеки, которые можно было использовать в различных ЯП. COM — это технология стандарт от компании MS, предназначенный для создания ПО на основе взаимодействующих компонентов, каждый из которых может использоваться во многих программах одновременно. Стандарт воплощает в себе идеи полиморфизма и инкапсуляции ООП. История:

- COM был разработан в 1993 году MS как основа для развития технологии OLE (технология связывания и внедрения объектов в другие документы и объекты)
- На основе COM были реализованы технологии: OLE Automation, ActiveX, DCOM, COM+, DirectX и CPCOM
- В современных Windows COM используется очень широко

Недостатки:

- усложненная инфраструктура
- хрупкая модель развертывания
- возможность работы только под Win
- обработка ошибок — использование HRESULT
- необходимость использования два ЯП (.idl для описания интерфейса и, обычно, C++ для написания реализации)
- регистрация в системном реестре (GUID ключом)

2. Архитектура .NET



- CLR — исполнительный компонент для всех программ
- Base Class Library — библиотека с функциональностью, которая доступна для языков, исполняющих .NET Framework. Она состоит из классов, интерфейсов и повторно используемых типов, которые интегрируются с CLR
- ADO.NET — используется для создания слоя доступа к данным, запроса и манипулирования данными из нижележащего источника, например SQL
- ASP.NET — используется для построения сложных интернет приложений
- CLS — набор основных функциональных возможностей языка.
- CTS — спецификация определяющая как какой-либо тип должен быть определен для его правильного выполнения средой .NET.

3. Преимущества .NET

- Возможность взаимодействия с существующем кодом — позволяет комбинировать существующие двоичные компоненты COM (взаимодействовать с ними) с более новыми программными компонентами .NET и наоборот
- Поддержка многочисленных ЯП — приложения .NET можно создавать с использованием любого числа ЯП
- Общий исполняющий механизм, разделяемый всеми поддерживающими .NET языками — набор определенных типов, которые способен понимать каждый поддерживающий .NET ЯП
- Языковая интеграция — поддерживается межъязыковое наследование, обработка исключения и отладка кода (прим. Базовый класс написан на C#, а расширен на VB)
- Обширная библиотека базовых классов — эта библиотека позволяет избегать сложностей, связанных с выполнением низкоуровневых обращений к API
- Упрощенная модель развертывания — не регистрируется в системном реестре., более того позволяет существовать нескольким версиям одной и той же сборки
- Открытые исходные коды — недавние события
- Кроссплатформеность — недавние события

4. Строительные блоки платформы .NET (CLR, CTS и CLS)

Общеязыковая исполняющая среда (Common Language Runtame — CLR)

- CLR компилируется и выполняет программы на Microsoft Intermediate Language (MSIL). Задачей CLR является автоматическое обнаружение, загрузка и управление объектами .NET
- Кроме того, среда CLR заботится о ряде низкоуровневых деталей, таких как управление памятью, размещение приложения, координирование потоков и выполнение проверок, связанных с безопасностью

Общая система типов (Common Type System — CTS)

- В спецификации CTS полностью описаны все возможные типы данных и все программные конструкции, поддерживаемые исполняющей средой
- В CTS показано, как эти сущности могут взаимодействовать друг с другом, и указано, как они представлены в формате метаданных .NET

Общая спецификация (Common Language Specification — CLS)

- Набор правил, подробно описывающих минимальное и полное множество характеристик, которые должен поддерживать отдельный компилятор .NET, чтобы генерировать программный код, обслуживаемый средой CLR и в то же время доступный в унифицированной манере всем языкам, ориентированным на платформу .NET

5. Отношения между CLR, CTS, CLS с библиотеками базовых классов

Библиотеки базовых классов определяю типы, которые можно использовать для построения программных приложений любого вида (ASP.NET — построение веб-сайтов, WCF — создание сетевых служб, WPF — настольные приложения). Предоставляет типы для взаимодействия с XML — документами, локальным каталогом и файловой системой текущего компьютера, для коммуникации с реляционными базами данных.

6. Управляемый и не управляемый код

Управляемый код — код программы, исполняемый под управление виртуальной машины .NET CLR. Не управляемы код — обычный машинный код.

Слово управляемый относится к методу обмена информацией между программой и исполняющей средой. То есть управляющая среда в любой точке может приостановить исполнение и получить информацию, специфичную для текущего состояния.

Управляемый код и неуправляемый код могут взаимодействовать друг с другом.

7. Интеграция различных языков .NET

Еще одно полезное преимущество интеграции различных языков .NET в одном унифицированном программном решении вытекает из того простого факта, что каждый язык программирования имеет свои сильные (а также слабые) стороны. Например, некоторые языки программирования обладают превосходной встроенной поддержкой сложных математических вычислений. В других лучше реализованы финансовые или логические вычисления, взаимодействие с мэйнфреймами и т.п. А когда преимущества конкретного языка программирования объединяются с преимуществами платформы .NET, выигрывают все.

Конечно, в реальности велика вероятность того, что будет возможность тратить большую часть времени на построение программного обеспечения с помощью предпочитаемого языка .NET. Однако, после освоения синтаксиса одного из языков .NET, изучение синтаксиса какого-то другого языка существенно упрощается. Вдобавок это довольно выгодно, особенно тем, кто занимается консультированием по разработке ПО. Например, тому, у кого предпочитаемым языком является C#, в случае попадания в клиентскую среду, где все построено на Visual Basic, это все равно позволит эксплуатировать функциональные возможности .NET Framework и разбираться в общей структуре кодовой базы с минимальным объемом усилий и беспокойства.

В .NET поддерживается межъязыковое наследование, межъязыковая

обработка исключений и межъязыковая отладка кода. Например, базовый класс может быть определен в C#, а затем расширен в Visual Basic.

8. Роль языка CIL, понятия модуля, сборки и метаданных

Сразу можно не понять, зачем нужно компилировать исходный код программы в промежуточный язык CIL. Одной из причин этого является то, что в платформе .NET используется много языков программирования (интеграция языков), и все компиляторы .NET выдают приблизительно одинаковы наборы CIL-инструкций, которые могут немного быть подправлены, но основное содержание будет одинаково для всех языков. Из этого следует, что все языки программирования могут взаимодействовать друг с другом в пределах четко обозначенной области.

Модуль CLR – это байтовый поток, хранящийся обычно в виде файла в локальной файловой системе или на Web-сервере. Модуль CLR содержит:

- код (обычно в формате CIL, но может быть код и в формате инструкций конкретного процессора)
- метаданные
- ресурсы

В рамках технологии .NET сборка — двоичный файл, содержащий управляемый код. Когда компилятор платформы .NET создает EXE или DLL модуль, содержимое этого модуля называется сборкой. Сборка содержит в себе: номер версии, метаданные и инструкции IL.

Сборка – это логический набор из одного или произвольного количества модулей.

Модули являются физическими конструкциями, а сборка является логической конструкцией.

Метаданные в .NET — термин обозначающий определенные структуры данных, добавляемые в код CIL для описания высокоуровневой структуры кода. Метаданные описывают все классы и члены классов, определенные в сборке, а также классы и члены классов, которые текущая сборка вызывает из другой сборки. Метаданные для метода содержат полное описание метода, включая его класс (а также сборку, содержащую этот класс), его возвращаемый тип и все параметры этого метода.

9. Управляемый код и сборка

- C# порождает код, который может выполняться только в рамках исполняющей среды .NET
- Для обозначения кода, ориентированного на исполняющую среду .NET, применяется термин управляемый код
- Двоичный модуль, который содержит управляемый код, называется сборкой
- В противоположность этому, код который не может обслуживаться непосредственно исполняющей средой .NET, называется неуправляемым кодом

Сборки:

- Хотя двоичные модули .NET имею тоже самое файловое расширение как и неуправляемые двоичные компоненты Win внутренне они устроены по разному
- Двоичные модули .NET содержат не специфическое, а независимые от платформы инструкции на промежуточном языке и метаданные типов
- Когда файл .dll/.exe был создан с помощью .NET- компилятора, полученный большой

двоичный объект называется сборкой

- Сборка представляет собой поддерживающий версии само-описательный двоичный файл, обслуживаемый средой CLR
- Приложение .NET конструируется за счет соединения вместе любого количества сборок

10. Преимущества использования сборок

- Увеличивает возможности для повторного использования кода
- Определяет границы типов. Сборка, в которой находится тип, в дальнейшем определяет его идентичность
- Сборки являются единицами, поддерживающими версии (Major — основная, Minor — второстепенная версия приложения или библиотеки, Build — номер построения приложения или библиотеки, Revision — номер ревизии для текущего построения)
- Сборки являются само-описательными. Сборка содержит метаданные, которые описывают структуру каждого из имеющихся в ней типов
- Сборки поддерживают конфигурации (XML - файлы). С их помощью можно указать CLR — среде: где конкретно искать сборки (причем не только локально) и какую версию сборки использовать

11. Формат сборки .NET

Сборка состоит из следующих элементов:

- Манифест сборки — содержит метаданные сборки
- Метаданные типов — используя эти метаданные, сборка определяет местоположение типов в файле приложения, а также места размещения их в памяти
- CIL код (код приложения) в который компилируется код C# или другой ЯП
- заголовок файла Windows
- заголовок файла CLR
- Доп. ресурсы

12. Закрытые и разделяемые сборки

Закрытые — размещаются в том же каталоге, что и клиентское приложение, в котором они используются.

Разделяемые сборки — это библиотеки, предназначенные для применения во многих приложениях на одной и той же машине, и они развертываются в специальном каталоге, который называется глобальным кешем сборок.

13. Строгие имена

При задании строгого имени для сборки для нее создается уникальный идентификатор, что позволяет избежать конфликтов сборок.

Сборка со строгим именем создается с помощью закрытого ключа, который соответствует открытому ключу, распространяемому со сборкой, и самой сборке. Сборка включает манифест сборки, который содержит имена и хэши всех файлов, из которых состоит сборка. Сборки с одинаковым строгим именем должны быть идентичны.

Задавать строгие имена для сборок можно с помощью Visual Studio или программы командной строки. Дополнительные сведения см. в разделе Практическое руководство. Подписание сборки строгим именем или Sn.exe (средство строгих имен).

При создании сборки со строгим именем она содержит простое текстовое имя сборки, номер версии, дополнительные сведения о языке и языковых параметрах, цифровую подпись и открытый ключ, который соответствует закрытому ключу, используемому для подписи.

При использовании ссылки на сборку со строгим именем можно пользоваться определенными преимуществами, например отслеживанием версий и защитой имен. Сборки со строгими именами могут устанавливаться в глобальный кэш сборок, который необходим для включения некоторых сценариев. Сборки со строгими именами полезны в следующих сценариях:

- Необходимо обеспечить возможность ссылок на ваши сборки сборкам со строгими именами или предоставить доступ friend к вашим сборкам из других сборок со строгими именами.
- Приложению требуется доступ к различным версиям одной и той же сборки. Это означает, что требуются разные версии сборки для параллельной загрузки в одном домене приложений без возникновения конфликтов. (например, если в сборках с одним простым именем существуют разные расширения API, назначение строгих имен обеспечивает однозначную идентификацию всех версий сборки);
- Использование вашей сборки не должно отрицательно сказываться на производительности приложений, поэтому сборка должна быть нейтральной к доменам. Для этого требуется строгое именование, так как нейтральная к доменам сборка должна устанавливаться в глобальный кэш сборок.
- Если требуется применять централизованное обслуживание приложений с помощью политики издателя (это означает, что сборка должна устанавливаться в глобальном кэше сборок);

14. Метаданные типов

Компоновочный блок в .NET, кроме CIL-инструкций, еще содержит и исчерпывающие и точные метаданные, которые описывают все типы, используемые в нем (классы, списки, структуры и др.), которые определены в бинарном объекте, и все члены каждого из типов (свойства, события, методы и др.).

Метаданные создаются автоматически компилятором. Из-за того, что метаданные в .NET (дот нет) так подробны и точны, компоновочные блоки способны сами себя описать, причем так полно, что для бинарных .NET объектов нет нужды регистрироваться в реестре системы.

Метаданные используются в среде выполнения .NET и в средах программирования, например, в Visual Studio в режиме проектирования, имеется возможность IntelliSense, которая основана на чтении метаданных компоновочного блока. Так же метаданные

используются в утилитах просмотра объектов, в инструментах отладки программного кода и в компиляторе языка программирования. Так же стоит заметить, что использование метаданных является основой множества .NET-технологий, в которые входит отображение типов, динамическое связывание и удаленный доступ, Web-сервисы XML и сериализацию объектов.

15. Ключевые слова

CTS устанавливает четко определенный набор фундаментальных типов данных. Ключевые слова всех языков .NET в конечном итоге соответствуют одному и тому же типу CTS, определенному в сборке mscorlib.dll

Тип данных CTS	Ключевое слово VB	Ключевое слово C#	Ключевое слово C++/CLI
System.Byte	Byte	byte	unsigned char
System.SByte	SByte	sbyte	signed char
System.Int16	Short	short	short
System.Int32	Integer	int	int или long
System.Int64	Long	long	__int64
System.UInt16	UShort	ushort	unsigned short
System.UInt32	UInteger	uint	unsigned int или unsigned long
System.UInt64	ULong	ulong	unsigned __int64
System.Single	Single	float	float
System.Double	Double	double	double
System.Object	Object	object	object^
System.Char	Char	char	wchar _ t
System.String	String	string	String^
System.Decimal	Decimal	decimal	Decimal
System.Boolean	Boolean	bool	bool

16. Рефлексия

В мире .NET рефлексией (reflection) называется процесс обнаружения типов во время выполнения. С применением служб рефлексии те же самые метаданные, которые отображает утилита ildasm.exe, можно получать программно в виде удобной объектной модели. Например, рефлексия позволяет извлечь список всех типов, которые содержатся внутри определенной сборки *.dll или *.exe (или даже внутри файла *.netmodule если речь идет о многофайловой сборке), в том числе методов, полей, свойств и событий, определенных в каждом из них. Можно также динамически обнаруживать набор интерфейсов, которые поддерживаются данным типом, параметров, которые принимает данный метод, и других деталей подобного рода (таких как имена базовых классов, информация о пространствах имен, данные манифеста и т.д.).

17. Класс System.Type

Представляет объявления типов для классов, интерфейсов, массивов, значений, перечислений параметров, определений универсальных типов и открытых или закрытых сконструированных универсальных типов.

Класс Type является корневым классом для функциональных возможностей пространства имен System.Reflection и основным способом доступа к метаданным. С помощью членов класса Type можно получить сведения об объявленных в типе элементах: конструкторах,

методах, полях, свойствах и событиях класса, а также о модуле и сборке, в которых развернут данный класс.

Объект Type возвращает оператор языка C# typeof (оператор GetType в Visual Basic, оператор typeid в Visual C++).

18. Позднее и раннее связывание

Раннее связывание - на этапе компиляции.

Позднее - при выполнении.

Поздним связыванием (late binding) называется технология, которая позволяет создавать экземпляр определенного типа и вызывать его члены во время выполнения без кодирования факта его существования жестким образом на этапе компиляции. При создании приложения, в котором предусмотрено позднее связывание с типом из какой-то внешней сборки, добавлять ссылку на эту сборку нет никакой причины, и потому в манифесте вызывающего кода она непосредственно не указывается.

первый взгляд увидеть выгоду от позднего связывания не просто. Действительно, если есть возможность выполнить раннее связывание с объектом (например, добавить ссылку на сборку и разместить тип с помощью ключевого слова new), следует обязательно так и поступать. Одна из наиболее веских причин состоит в том, что раннее связывание позволяет выявлять ошибки во время компиляции, а не во время выполнения. Тем не менее, позднее связывание тоже играет важную роль в любом создаваемом расширяемом приложении.

Класс System. Activator (определенный в сборке mscorlib.dll) играет ключевую роль в процессе позднего связывания в .NET. В текущем примере интересует пока что только его метод Activator.CreateInstance(), который позволят создавать экземпляр подлежащего позднему связыванию типа. Этот метод имеет несколько перегруженных версий и потому обеспечивает довольно высокую гибкость. В самой простой версии CreateInstance() принимает действительный объект Type, описывающий сущность, которая должна размещаться в памяти на лету.

19. Понятие CTS

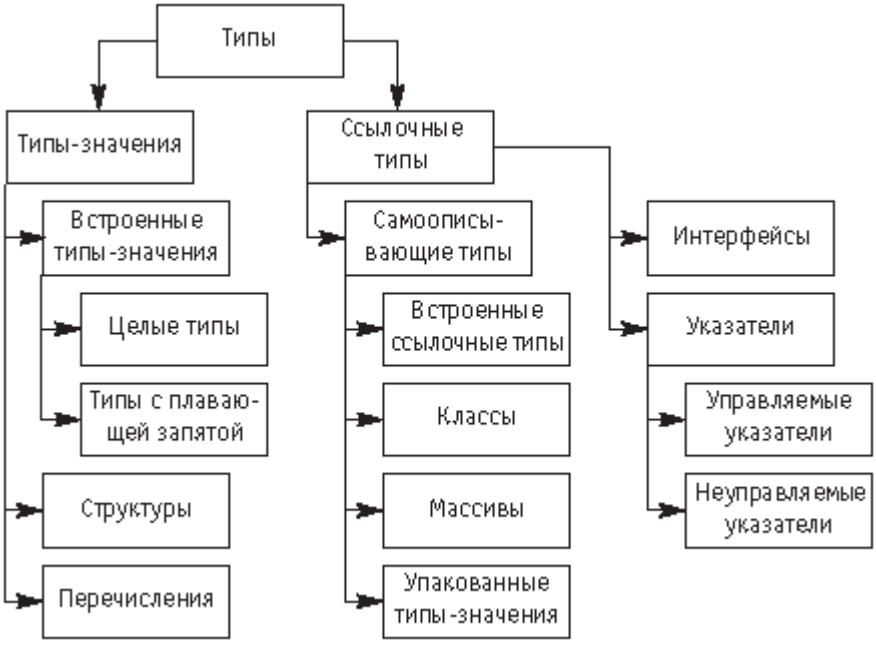
CTS — формальная спецификация, определяющая, как какой-либо тип (класс, интерфейс, структура, встроенный тип данных) должен быть определен для его правильного выполнения средой .NET. Определяет, как определения типов и специальные значения типов представлены в компьютерной памяти.

20. Система типов и обеспечение безопасности

Система общих типов CTS определяет способ объявления, использования и управления типами в среде CLR, а также является важной составной частью поддержки межъязыковой интеграции в среде выполнения. Система общих типов выполняет следующие функции.

- Формирует инфраструктуру, которая позволяет обеспечивать межъязыковую интеграцию, безопасность типов и высокопроизводительное выполнение кода.
- Предоставляет объектно-ориентированную модель, поддерживающую полную реализацию многих языков программирования.
- Определяет правила, которых необходимо придерживаться в языке. Эти правила помогают обеспечить взаимодействие объектов, написанных на разных языках.
- Предоставляет библиотеку, которая содержит типы-примитивы (например, Boolean, Byte, Char, Int32 и UInt64), используемые в разработке приложений.

21. Ядро общей системы типов



22. Понятие общей языковой спецификации (CLS)

Каждый отдельно взятый язык, совместимый с .NET, может не поддерживать абсолютно все функциональные средства, определенные спецификацией CTS. Для этого создали *общеязыковую спецификацию* (CLS) в которой описано подмножество общих типов и программных конструкций, которые должны поддерживать все языки программирования для .NET. CLS — набор правил, описывающих во всех подробностях минимальное и полное количество возможностей, которое должен поддерживать некоторый .NET компилятор.

23. Причина для изучения грамматики языка CIL

- Точно знаешь какие лексемы CIL отображают ключевые слова из различных ЯП .NET
- Умеешь дизассемблировать существующие сборки .NET, редактировать лежащий в их основе CIL и заново компилировать обновленную кодовую базу в модифицированный двоичный файл .NET
- Умеешь строить динамические сборки с использованием пространства имен System.Refkection,Emit
- Умеешь использовать такие возможности CTS, которые не поддерживаются управляемыми языками более высокого уровня, но существуют на уровне CIL. На самом деле CIL является единственным языком .NET, который позволяет получать доступ к абсолютно всем возможностям CTS

24. Директивы, атрибуты и коды операция CIL

Лексемы — последовательность допустимых символов языка программирования, имеющая смысл для транслятора. Транслятор рассматривает программу как последовательность лексем.

Директивы — позволяю информировать компилятор CIL о том, как должны определяться пространства имен, типы и члены, входящие в состав сборки. Синтаксически директивы представляются с использованием префикса в виде точки (прим. .namespace, .class, .pulickeytone, .override, .method, .assembly и т.д.)

Атрибуты — директивы CIL могут сопровождаться разнообразными атрибутами CIL,

которые уточняют способ обработки той или иной директивы. Например директива .class может быть снабжена атрибутом public, extends и implements.

Коды операций CIL имеют непонятный и нечитабельный вид. Например, для загрузки в память переменной string в CIL применяется код операции, который имеет не дружественное имя наподобие LoadString, а записывается как ldstr. Некоторые КО повторяют свои аналоги в C#(box, unbox, throw, sizeof). Коды операций CIL всегда используются только в контексте реализации членов и никогда не сопровождаются префиксом в виде точки.

В действительности лексемы, подобные ldstr. Являются мнемоническими эквивалентами CIL действительных двоичных кодов операций.

25. Основанная на стеке природа CIL

В языках .NET высокого уровня низкоуровневые детали CIL обычно максимально возможно скрываются из виду. Одним из хорошо скрываемым аспектом является тот факт, что CIL является ЯП, основанным на использовании стека. Формально сущность, используемая для хранения набора вычисляемых значений, называется виртуальным стеком выполнения. В CIL невозможно получать доступ к элементам данных, включая локально определенные переменные, входные аргументы методов и данные полей типа. Вместо этого элемент данных должен быть явно загружен в стек и затем извлекаться оттуда для последующего использования. CIL предоставляет несколько кодов операций, которые служат для помещения значения в стек — загрузка. (ldarg, ldc, ldfld, ldloc, ldobj, ldstr) В CIL определен набор доп. Кодов операций, которые перемещают самое верхнее значение из стека в память — сохранение (pop — выгружает, но не сохраняет, starg, stloc, stobj, stsfld).
Пример: .

```
locals init ([0] string myMessage)
ldstr «Hello.»
stloc.0
ldloc.0
call void [mscorlib]System.Console::WriteLine(string)
```

26. Возвратное проектирование

Возвратное проектирование (Reverse Engineering) — ситуации при которых необходимо модифицировать сборку, исходный код которой больше не доступен, необходимо изменить неэффективный код CIL или надо сконструировать библиотеку взаимодействие с COM и учесть ряд атрибутов COM IDL.

- Можно просматривать код CIL, сгенерированный компилятором C#.
- Можно сбрасывать код CIL, содержащийся внутри загруженной в ildasm сборки, во внешний файл.
- Полученный подобным образом CIL-код можно легко редактировать и затем компилировать с помощью ilasm.

27. Директивы и атрибуты CIL

См. 24 пункт.

Пример: .namespace MyNamespace { .class public MyBaseClass() }

28. Соответствие между базовыми классами .NET, C# и CIL

Базовый класс .NET	Ключевое слово в C#	Представление CIL	Константная нотация CIL
System.SByte	sbyte	int8	I1
System.Byte	byte	unsigned int8	U1
System.Int16	short	int16	I2
System.UInt16	ushort	unsigned int16	U2
System.Int32	int	int32	I4
System.UInt32	uint	unsigned int32	U4
System.Int64	long	int64	I8
System.UInt64	ulong	unsigned int64	U8
System.Char	char	char	CHAR
System.Single	float	float32	R4
System.Double	double	float64	R8
System.Boolean	bool	bool	BOOLEAN
System.String	string	string	-
System.Object	object	object	-
System.Void	void	void	VOID

29. Определение членов типов в CIL

Типы в .NET могут иметь различные члены. Так, перечисления могут иметь набор пар имен и значений, а структуры и класы — конструкторы, поля, методы, статические члены и т.д.

Определение полей данных в CIL — перечисления, структуры и классы поддерживают поля данных. В каждом случае для их определения должна использоваться директива .field

Определение конструкторов для типов CIL — в CTS поддерживается создание конструкторов, действующих как на уровне всего экземпляра, так и на уровне конкретного класса (стат. Конструктор). В CIL первые представляются с помощью лексемы .ctor, а вторые — посредством лексемы .cctor. Обе лексемы должны сопровождаться атрибутами rtspecialname и specialname.

```
.method public hidebysig specialname rtspecialname instance void .ctor(string s, int32
i) cil managed {
 // Какой-то код реализации
}
```

Определение свойств в CIL:

```
.method public hidebysig specialname instance string get_TheString() cil managed
{
     // Код реализации
}
.method public hidebysig specialname instance void set_TheString(string 'value') cil
managed
{
    // Какой-то код
}
.property instance string TheString()
{
  .get instance string MyUI.MyUserInfo::get_TheString()
  .set instance void MyUI.MyUserInfo::set_TheString(string)
}
```

Определение параметров членов - в целом аргументы в CIL задаются (более или менее) тем же образом, что и в C#. Например, каждый аргумент в CIL определяется за счет указания типа данных, к которому он относится, и затем самого его имени. Более того, как и в C#, в CIL можно определять входные, выходные и передаваемые по ссылке параметры.

Определение сборок:

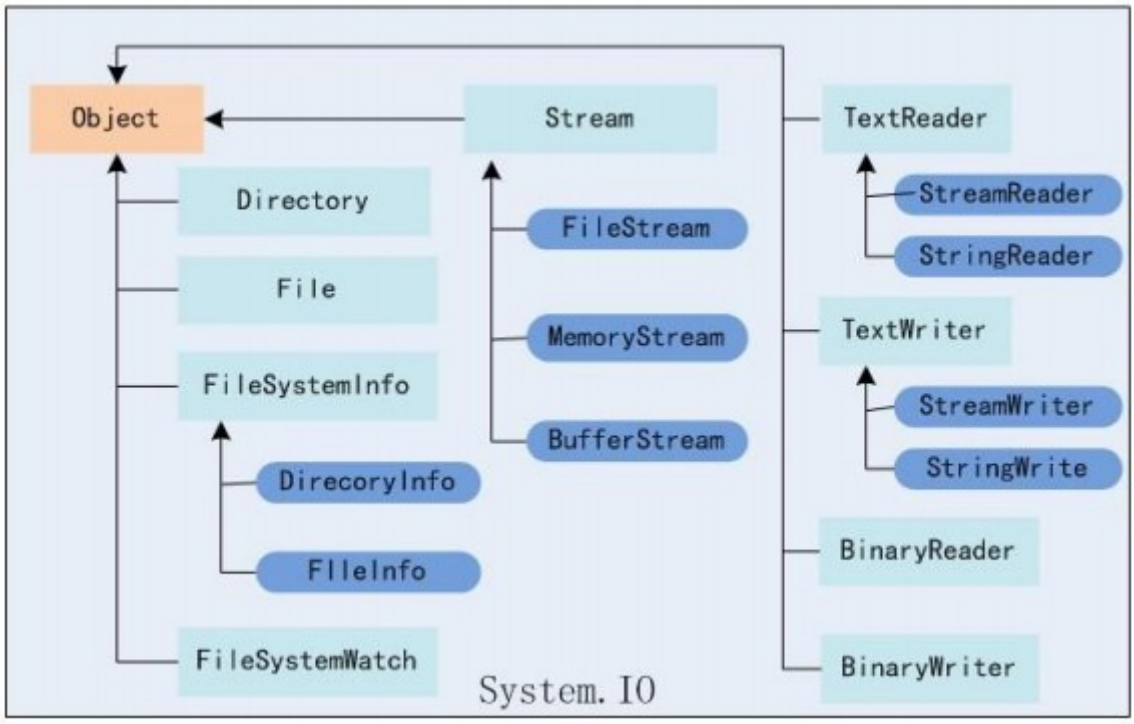
```
.assembly CILTypes { .ver 1:0:0:0 }
.assembly extern mscorlib { .publickeytoken = {B7 7A …} .ver 4:0:0:0 }
```

30. Коды операции CIL

См. 24 пункт.

31. Пространства имен System.IO

Пространство имен System.IO содержит типы, позволяющие осуществлять чтение и запись в файлы и потоки данных, а также типы для базовой поддержки файлов и папок.



32. Программное отслеживание файлов

FileSystemWatcher - этот класс позволяет отслеживать модификации внешних файлов в определенном каталоге. Ожидает уведомления файловой системы об изменениях и инициирует события при изменениях каталога или файла в каталоге. С помощью FileSystemWatcher можно организовать мониторинг файлов на предмет любых действий, указанных в перечислении System.I0.NotifyFilters. Работа с классом:

- установить Path, а также Filter
- какие события мы ожидаем
- создать обработчики событий changed, created, deleted (работают в сочетании с делигатом FileSystemEventHandler), renamed (работает в сочитании с RenamedEventArgs)

33. Сериализация объектов

Сериализация — процесс перевода какой-либо структуры данных в последовательность битов. Обратной к операции сериализации является операция десериализация — восстановление начального состояния структуры данных из битовой последовательности. Используется для передачи объектов. Последовательность сохраняемых данных содержит всю информацию, необходимую для реконструкции состояния объекта с целью последующего использования. Сериализация может быть: двоичной, XML — сер., SOPA — сер., JSON — сер. Сериализацию можно настраивать — т.е. какие объекты будут сериализованы, и как будет производится сериализация. Преимущества: очень просто сохранять большие объемы данных с минимальными усилиями. Меньше кода требуется, чем чтение/запись.

34. Структурированная обработка исключений

Исключения — аномалии, возникающие во время выполнения, которые трудно, а порой и невозможно учесть во время программирования приложения. В терминологии .NET под исключением подразумеваются программные ошибки, некорректный пользовательский ввод и ошибки времени выполнения. Исключением называется ситуация, когда член типа не в состоянии решить возложенную на него задачу. Структурированная обработка исключений в .NET — это прием, предназначенный для работы с исключительными ситуациями, возникающими во время выполнения. Преимущества:

- унифицированный подход к обработке ошибок, который является общим для всех языков .NET
- Синтаксис для всех ЯП .NET одинаковый
- В отличии от запутанных числовых значений, просто обозначающих текущую проблему, они представляю собой объекты, в которых содержится читабельное описание проблемы, а также стек вызовов на момент первоначального возникновения исключения
- Пользователю можно предоставить URL — с описанием проблемы

Внутренние исключение — ситуация когда исключение может быть сгенерировано во время обработки другого исключения. Если во время обработки исключения возникает еще одно исключение, то согласно рекомендации необходимо сохранить новый объект исключения как внутреннее исключение в новом объекте того же типа, что и у исходного исключения.

35. Время жизни объектов

Время жизни объекта — время с момента создания объекта (конструкция) до его уничтожения (деструкция). Объект уничтожается когда на него ни одна ссылка не ссылается.

В .NET существует 3 поколения:

- Поколение 0. Новый созданный объект, который еще никогда не помечался как кандидат на удаление;
- Поколение 1. Объект, который уже однажды пережил сборку мусора. Сюда обычно попадают объекты, которые были помечены для удаления, но все-таки не удалены из-за того, что места в памяти (куче) было предостаточно;
- Поколение 2. Объекты, которые пережили более одной очистки памяти сборщиком мусора.

Очевидно, что чем выше поколение объекта, то тем дольше он находится в памяти. Более

того, вероятность дальнейшего его существования в куче больше, чем у объекта более низкого поколения.

36. Автоматическое управление памятью

Сборка мусора (GC) — одна из форм автоматического управления памятью. Сборка мусора полностью освобождает разработчика от необходимости следить за использованием и своевременным освобождением памяти.

37. Алгоритм сборки мусора

- GC проверяет наличие в куче больше не используемых приложением объектов, чтобы освободить занятую ими память
- GC предполагает что все объекты в куче — мусор
- GC переходит к этапу маркировки — проходит по стеку потока и проверяет корни. Если оказывается, что корень ссылается на объект, в поле этого объекта включается бит — признак маркировки объекта
- После маркировки GC проверяет следующий корень и продолжает маркировать объекты. Встретив уже маркированный объект, сборщик мусора останавливается
- После проверки всех корней куча содержит набор маркированных и немаркированных объектов. Маркированные объекты достижимы из кода приложения. Недостижимые (немаркированные) — мусор и занимаемая ими память становиться доступной.
- GC переходит сжатию к (compact phase) — проходит кучу линейно в поисках непрерывных блоков немаркированных объектов, то есть мусора. Небольшие блоки он не трогает, а в больших непрерывных блоках он перемешает вниз все «немусорные» объекты, сжимая при этом кучу.
- Перемещение объектов в памяти делает недействительными все переменные и регистры процессора, содержащие указатели на объекты. Поэтому сборщик мусора должен вновь проверить и обновить все корни приложения, чтобы все значения корней указывали на новые адреса объектов в памяти. Кроме того, если объект содержит поле, указывающее на другой перемещенный объект, сборщик должен исправить и эти поля.

38. Пространство имен System.Thread

Пространство имен System.Threading содержит классы и интерфейсы, которые дают возможность программировать в многопоточном режиме. В дополнение к классам, чтобы синхронизировать действия потока и доступ к данным (Mutex, Monitor, Interlocked, AutoResetEvent и т д) данное пространство имен включает класс ThreadPool, который позволяет использовать кластера система- предоставленных потоков и класс Timer, который выполняет обратные вызовы на потоках из пула потоков.

39. Синхронизация потоков в .NET

Платформа .NET Framework предоставляет диапазон примитивов синхронизации для управления взаимодействием потоков и во избежание состояния гонки. Они могут быть условно разделены на три категории: блокировка, сигнал и блокированные операции.

Эти категории точно не определены: некоторые механизмы синхронизации обладают характеристиками нескольких категорий; события, освобождающие одновременно только один поток, по своему действию подобны блокировкам; снятие любой блокировки можно считать сигналом, а блокированные операции могут быть использованы для создания блокировок. Однако эти категории остаются полезными.

Важно помнить, что синхронизация потоков выполняется совместно. Если хотя бы один поток обходит механизм синхронизации и получает доступ напрямую к защищенному ресурсу, такой механизм синхронизации не может считаться эффективным.

- Блокировка - предоставляют единовременное управление ресурсом одному потоку или определенному количеству потоков. Поток, запрашивающий монопольную блокировку, если блокировка уже используется, блокируется до того момента, как блокировка станет доступной.
- Самым простым способом ожидания сигнала от другого потока является вызов метода Join, который блокирует поток до завершения другого потока. У метода Join есть две переопределенных версии, которые позволяют блокированному потоку освободиться от ожидания по истечении заданного времени. Дескрипторы ожидания предоставляют более полный набор возможностей ожидания и сигнализации.
- Блокируемые операции — это простые атомарные операции, выполняемые в области памяти статическими методами класса Interlocked. Эти атомарные операции включают добавление, увеличение и уменьшение, обмен, условный обмен в зависимости от сравнения, а также операции чтения для 64-разрядных значений на 32-разрядных платформах.

40. Пулы потоков

Пул потоков — это коллекция потоков, которые могут использоваться для выполнения нескольких задач в фоновом режиме. Это позволяет разгрузить главный поток для асинхронного выполнения других задач.

Пулы потоков часто используются в серверных приложениях. Каждый входящий запрос назначается потоку из пула, таким образом, запрос может обрабатываться асинхронно без задействования главного потока и задержки обработки последующих запросов.

Когда поток в пуле завершает выполнение задачи, он возвращается в очередь ожидания, в которой может быть повторно использован. Повторное использование позволяет приложениям избежать дополнительных затрат на создание новых потоков для каждой задачи.

Обычно пулы имеют максимальное количество потоков. Если все потоки заняты, дополнительные задачи помещаются в очередь, где хранятся до тех пор, пока не появятся свободные потоки.

Можно реализовать собственный пул потоков, но гораздо проще использовать пул, предоставляемый .NET Framework через класс ThreadPool.

Используя группировку потоков в пул, можно вызвать метод ThreadPool.QueueUserWorkItem с делегатом для процедуры, которую требуется выполнить, а Visual Basic или C# создаст поток и выполнит процедуру.