

## **.NET**

В 2000 году пошли слухи о том, что корпорация Microsoft начала работу над проектом .NET. Теперь, когда продукт выпущен, и уже идет разработка .NET 2005, поднялся настоящий шум со стороны любителей Java. Как оказалось, эти технологии имеют много общего. Разумеется, Microsoft переняла кое-что от технологии Java. Но ведь и Java в свое время ничего нового не открыла - в частности, то, чем Sun так гордится, именуемое JavaVirtualMachine, было не их идеей, а посему - не будем заикливаться на том, кто у кого что перенял. Нам ведь результат важен, не правда ли?

Во-первых .NET предлагает FCL. FCL (FrameworkClassLibrary) - это большая, просто огромная, коллекция готовых к использованию классов. Java API по сравнению с FCL - жалкое зрелище. Я не буду уделять этому много времени, скажу только, что в состав FCL входят более 10000 классов, и это количество просто не сравнимо с количеством классов входящих в Java API.

Основа же .NET лежит в CLR. CLR (CommonLanguageRuntime) - общезыковая исполняющая среда. Эта среда, где выполняются .NET приложения. Конечно же, это аналогия виртуальной машины Java, и мы поговорим об их сходствах.

CLR, которая на русский переводится как "общезыковая исполняющая среда" - это сложный программный аппарат, предназначенный для стирания границ между разными языками программирования. CLR легко исполняет программы, части которых написаны на разных языках. В состав языков .NET входят: C++, C#, J#, VisualBasic.

**Язык IL.** Этот язык с первого взгляда похож на ассемблер,...однако у них не много общего. В частности, в IL нет регистров, и все операции выполняются через стек.

*Тип (type) - класс .NET.* Классы в .NET называются типами. для запуска .NET программ требуется .NET Framework (содержащий CLR и FCL).

Так же не забудем о **JITter**.

. Сборщик мусора тоже не является новшеством по отношению к Java, но для новичков в Java и в .NET я уделю пару слов этой великой программной мысли.

Сборщик мусора работает следующим образом. Он отслеживает все ссылки на экземпляры типов, и когда понимает, что ваше приложение больше не может работать с данным экземпляром, сборщик мусора освобождает память, которую занимает экземпляр, до этого проверяя все ссылки, на которые ссылался сам экземпляр

## **Java**

Когда-то очень давно, в 1990 году, была разработана технология Java (совмещающая язык Java и платформу Java). Сейчас она принадлежит компании Sun. Этот язык многие

считали прорывом в мире программирования. Главная идея технологии Java - это кросс-платформенность.

Основа же Javасостоит в (JavaVirtualMachine)

Виртуальная машина Java - программа, которая эмулирует вычислительную машину, запускает код Java приложения внутри себя, полностью его контролируя.

**Байт-код.** Байт-код един для всех платформ и архитектур. Таким образом, разработчик, написав и отладив свой код, может быть уверен в полной совместимости кода с разными платформами (если только он не использует функции, не предоставленные Java на других платформах). Как я уже говорил, за перевод байт-кода в процессорные инструкции отвечает JIT compiler (JustInTimecompiler).

**JITcompiler.** JIT компилятор компилирует байт-код в машинные коды. Сначала виртуальная машина определяет операционную систему, размер оперативной памяти, процессор и другие устройства. На основе полученных данных JIT-компилятор предварительно (или динамически) компилирует байт-код в инструкции, "подогнанные" под программную и аппаратную среду. Короче говоря, Java JIT компилятор оптимизирует код, как и JITter от .NET. Однако, как считает Microsoft, оптимизация JITter'a более совершенна.

**GarbageCollector.** Автоматическая сборка мусора встроена в виртуальную машину. Она работает в фоновом режиме, осуществляя поиск потерянных объектов и освобождая занятую ими память. Она аналогична сборщику мусора в .NET, только там программист не случайно теряет объекты, а сознательно переносит ответственность на плечи сборщика мусора.

[Историческое введение  
CLR и JavaVirtualMachine  
Что лучше - Java или .NET](#)

Автор: [SpongeBob](#)  
05.07.2004 11:37:00

Историческое введение

## Java

Когда-то очень давно, в 1990 году, была разработана технология Java (совмещающая язык Java и платформу Java). Сейчас она принадлежит компании Sun. Этот язык многие считали прорывом в мире программирования. Главная идея технологии Java - это кросс-платформенность. Так, программы на Java сейчас работают и в Windows, и в Unix, и на мобильных телефонах, то есть везде, где установлена виртуальная машина Java (JavaVirtualMachine). В состав технологии Java входит Java API (ApplicationProgrammingInterface) - набор классов, который предлагается разработчику и отличается на разных платформах.

## .NET

В 2000 году пошли слухи о том, что корпорация Microsoft начала работу над проектом .NET. Теперь, когда продукт выпущен, и уже идет разработка .NET 2005, поднялся настоящий шум со стороны любителей Java. Как оказалось, эти технологии имеют много общего. Разумеется, Microsoft переняла кое-что от технологии Java. Но ведь и Java в свое время ничего нового не открыла - в частности, то, чем Sun так гордится, именуемое JavaVirtualMachine, было не их идеей, а посему - не будем заикливаться на том, кто у кого что перенял. Нам ведь результат важен, не правда ли?

Во-первых .NET предлагает FCL. FCL (FrameworkClassLibrary) - это большая, просто огромная, коллекция готовых к использованию классов. Java API по сравнению с FCL - жалкое зрелище. Я не буду уделять этому много времени, скажу только, что в состав FCL входят более 10000 классов, и это количество просто не сравнимо с количеством классов входящих в Java API.

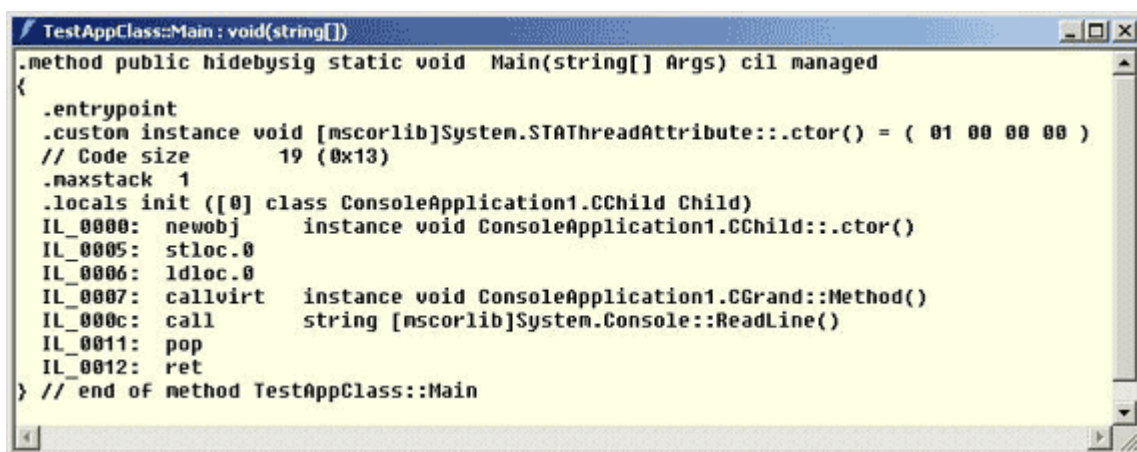
Основа же .NET лежит в CLR. CLR (CommonLanguageRuntime) - общезыковая исполняющая среда. Эта среда, где выполняются .NET приложения. Конечно же, это аналогия виртуальной машины Java, и мы поговорим об их сходствах.

## CLR и Java Virtual Machine

### CLR

CLR, которая на русский переводится как "общезыковая исполняющая среда" - это сложный программный аппарат, предназначенный для стирания границ между разными языками программирования. CLR легко исполняет программы, части которых написаны на разных языках. В состав языков .NET входят: C++, C#, J#, VisualBasic. Отмечу, что язык C# был разработан Microsoft специально для .NET. Как достигается совместимость языков? Оказалось Microsoft, пообщавшись с институтами, изучающими языки программирования, изобрели еще один язык - IL (IntermediateLanguage) - промежуточный язык.

**Язык IL.** Этот язык с первого взгляда похож на ассемблер,



```
TestAppClass::Main: void(string[])
.method public hidebysig static void Main(string[] Args) cil managed
{
    .entrypoint
    .custom instance void [mscorlib]System.STAThreadAttribute::.ctor() = ( 01 00 00 00 )
    // Code size          19 (0x13)
    .maxstack 1
    .locals init ([0] class ConsoleApplication1.CChild Child)
    IL_0000: newobj          instance void ConsoleApplication1.CChild::.ctor()
    IL_0005: stloc.0
    IL_0006: ldloc.0
    IL_0007: callvirt          instance void ConsoleApplication1.CGrand::Method()
    IL_000c: call              string [mscorlib]System.Console::ReadLine()
    IL_0011: pop
    IL_0012: ret
} // end of method TestAppClass::Main
```

...однако у них не много общего. В частности, в IL нет регистров, и все операции выполняются через стек. К примеру, при сложении 2 чисел, они поочередно закладываются в стек, выполняется метод сложения, и результат кладется тоже в стек. IL можно назвать "высокоуровневым машинным языком". Он легче ассемблера, и на нем вполне можно программировать. Изучая IL, я очень удивился, увидев в IL-коде команду

newobj, которая создаёт новый экземпляр типа.

*Tun (type)* - класс *.NET*. Классы в *.NET* называются типами. Именно в *IL* компилируются программы написанные на *.NET*. Т.к. процессор не может выполнять *IL*-инструкции, для запуска *.NET* программ требуется *.NET Framework* (содержащий *CLR* и *FCL*), который можно бесплатно скачать с сайта Microsoft. Для тех, кто заинтересовался языком *IL*, предлагаю прочитать статью MSDN по [этому адресу](#).

Для преобразования *IL*-команд в машинные инструкции, разработчики из Microsoft произвели на свет JIT компилятор (JustInTimecompiler, JITter) - компилятор реального времени.

**JITter.** Сейчас я позволю себе чуть-чуть углубиться в работу JITter'a и привести маленький исходник на *C#* (т.к. этот язык был создан специально для *.NET*), чтобы вы лучше и быстрее поняли, как он работает.

Код *C#*:

```
using System;
namespace ConsoleApplication1
{
    class TestAppClass
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello, .NET world");
            Console.WriteLine(".NET is the power!");
        }
    }
}
```

Сам по себе код прост. Программа дважды выводит текст на консоль. Но давайте подумаем, что же произойдет, когда метод *Console.WriteLine* вызовется в первый раз.

*Method (method)* - то, что мы привыкли называть процедурой или функцией. При первом вызове метода, *CLR* обнаружит, что код метода еще не переведен в ассемблер, и передаст управление JITter'у. JITter ассемблирует метод, сохраняя инструкции в оперативную память, и возвратит *CLR* указатель (адрес ячейки) на первую команду метода. Отгадайте, удалит ли ассемблерный код *CLR* после завершения выполнения метода? Конечно, нет. Представьте, какой была бы производительность *.NET* программ, если бы каждый раз коды метода переводились из *IL*-кода в ассемблер. Согласитесь, что достаточно низкой.

При втором вызове *CLR* обнаружит, что код метода *Console.WriteLine* уже ассемблирован (переведет в процессорные команды), получит адрес первой ячейки, содержащий код, и начнет с нее выполнение. Главная прелесть JITter'a в том, что, компилируя *IL*-код, JITter создает команды, оптимизированные под процессор и операционную систему. То есть при компиляции возможности компьютера будут использованы по максимуму.

**GarbageCollector.** Сборщик мусора тоже не является новшеством по отношению к Java, но для новичков в Java и в *.NET* я уделю пару слов этой великой программной мысли. Сборщик мусора работает следующим образом. Он отслеживает все ссылки на экземпляры типов, и когда понимает, что ваше приложение больше не может работать с данным экземпляром, сборщик мусора освобождает память, которую занимает экземпляр,

до этого проверяя все ссылки, на которые ссылался сам экземпляр. Вот маленький пример:

```
using System;
namespace ConsoleApplication1
{
    class StupidClass {}
    class MainApp
    {
        public static void Main(string[] args)
        {
            int a = 1;
            if (a == 1)
            {
                StupidClassExample = new StupidClass();
            }
            /* Здесь GarbageCollector поймет, что я уже не смогу использовать Example, поэтому
            освободит память, выделенную для этого экземпляра */
        }
    }
}
```

И еще: вы НИКОГДА не знаете, когда, наконец, GarbageCollector начнет очищать память. Поэтому если хотите, чтобы память освободилась немедленно, вам нужно дописать специальный код, который будет заставлять сборщик мусора работать. Вроде бы основные концепции CLR я изложил, теперь переходим к виртуальной машине Java...

## JavaVirtualMachine

Виртуальная машина Java - программа, которая эмулирует вычислительную машину, запускает код Java приложения внутри себя, полностью его контролируя.

Java-приложения компилируются в особый код - байт-код (byte-code), исполняемый на виртуальной машине Java. Как Вы уже, наверное, поняли, IL от Microsoft - аналогия байт-кода. Перед первым запуском приложения виртуальная машина проверяет, являются ли все инструкции корректными и безопасными. Затем виртуальная машина производит сборку модулей и устанавливает связи между именами, при этом поиск недостающих модулей производится не только на данном компьютере, но в Интернете. В первый раз связывание происходит относительно долго, повторно - гораздо быстрее. Затем, собственно, и начинается работа приложений. Виртуальная машина вызывает JIT компилятор, который динамически компилирует байт-код в машинные инструкции, одновременно оптимизируя их, или же сначала компилирует код, а потом запускает приложение. Это зависит от конкретной виртуальной машины.

**Байт-код.** Байт-код един для всех платформ и архитектур. Таким образом, разработчик, написав и отладив свой код, может быть уверен в полной совместимости кода с разными платформами (если только он не использует функции, не предоставленные Java на других платформах). Как я уже говорил, за перевод байт-кода в процессорные инструкции отвечает JIT compiler (JustInTimecompiler).

**JITcompiler.** JIT компилятор компилирует байт-код в машинные коды. Сначала виртуальная машина определяет операционную систему, размер оперативной памяти, процессор и другие устройства. На основе полученных данных JIT-компилятор

предварительно (или динамически) компилирует байт-код в инструкции, "подогнанные" под программную и аппаратную среду. Короче говоря, Java JIT компилятор оптимизирует код, как и JITter от .NET. Однако, как считает Microsoft, оптимизация JITter'a более совершенна.

**GarbageCollector.** Автоматическая сборка мусора встроена в виртуальную машину. Она работает в фоновом режиме, осуществляя поиск потерянных объектов и освобождая занятую ими память. Она аналогична сборщику мусора в .NET, только там программист не случайно теряет объекты, а сознательно переносит ответственность на плечи сборщика мусора.

## Что лучше - Java или .NET

Разумеется, лучше .NET :). По прогнозам аналитиков, через 5 лет .NET разработчиков будет на 60 % больше, чем Java разработчиков.

CLR и JavaVirtualMachine - очень похожи. Однако в то время, когда Sun приспособливает язык Java к разным платформам, Microsoft приспособливает разные языки к одной платформе - к платформе .NET. Всем известно, что корпорация Microsoft очень давно пытается перетянуть пользователей сторонних операционных систем в мир Windows, и .NET - очередной шаг в эту сторону. И если .NET программ будет много (а их будет много), и они не будут поддерживаться другими операционными системами, переход к Windows осуществится сам по себе...

Нужно отдать должное, корпорация Microsoft разработала и впрямь удивительную технологию, которая охватывает и решает множество проблем...

FCL - кладовка более 10 000 типов, готовых к употреблению. К ним относятся типы для работы с файлами, базами данных, Интернетом и т.д. Сравнить с FCL бедный Java API - просто бессмысленно.

Сборки - еще одна прелесть платформы Microsoft .NET. Сборки вполне можно рассматривать как пакеты, которые могут содержать разные части приложения - как сам exe-файл, так и отдельные модули. CLR позволяет запускать приложения, даже если некоторые модули отсутствуют. Это очень упрощает процесс разворачивания программ. Так, программист может продавать целую программу, а также дополнительные функции отдельно. Сборки решают главную проблему DLL, которую программисты прозвали "адом DLL". Суть проблемы заключается в следующем. Вы установили программу с DLL. Потом вы установили другую программу, которая поставляется с DLL, имя которой совпадает с первой (содержимое - разное), и при установке старая DLL перезаписывается новой. Теперь при запуске первой программы, она, вероятно, не получит нужной библиотеки и откажется работать.

.NET же обеспечивает приложениям получение именно той DLL, которая требуется. Идентификация достигается парой открытого и закрытого ключей. Сборки также полностью исключают работу с реестром. Теперь установка программного обеспечения сводится к копированию файлов. Это решает проблемы "засорения" операционной системы и уменьшает вероятность сбоя программ.

ADO .NET - Простая в использовании технология работы с данными через Интернет.

ASP .NET - это следующее поколение ASP (ActiveServerPages), в котором предоставлена новая, основанная на .NET Framework модель программирования. При помощи данной технологии, Вы можете легко создавать Web-формы и повторно используемые компоненты, а также Web-службы.

.NET не привязывает вас к использованию одного языка. У вас в распоряжении имеются C++ (ManagedExtensions), VisualBasic, C#, J#. Язык J# позволяет создавать .NET приложения на языке Java.

.NET позволяет разрабатывать на высоком уровне сложные программные продукты для Smartphone (Pocket PC).