.Net Framework

.NET Framework — это платформа разработки, для создания приложений для Windows, Windows Phone, Windows Server и Microsoft Azure.

История

Машинный язык

«Hello, world!» для процессора архитектуры x86

```
BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD 10 E2 F9 CD
20 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21
```

История

Язык ассемблера

```
.MODEL
        SMALL
.DATA
        msg DB 'Hello World',13,10,'$'
.CODE START:
        mov ax, @DATA
        mov ds, ax
        mov ax, 0900h
        lea dx, msg
        int 21h
        mov ax, 4C00h
        int 21h
END START
```

История

Языки высокого уровня

- Fortran

```
print *, "Hello, World!"
end
```

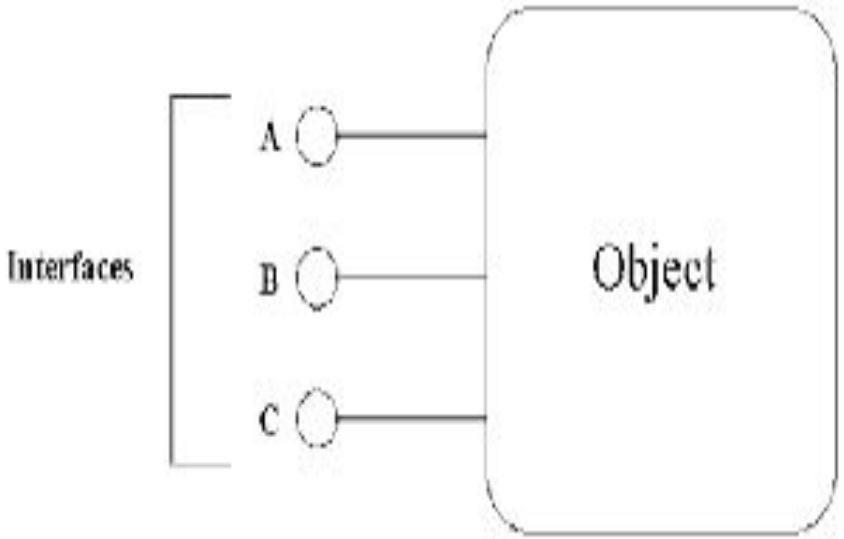
- LISP
- ALGOL
- Pascal
- C

История

- Процедурно ориентированное программирование
- Объектно ориентированное программирование

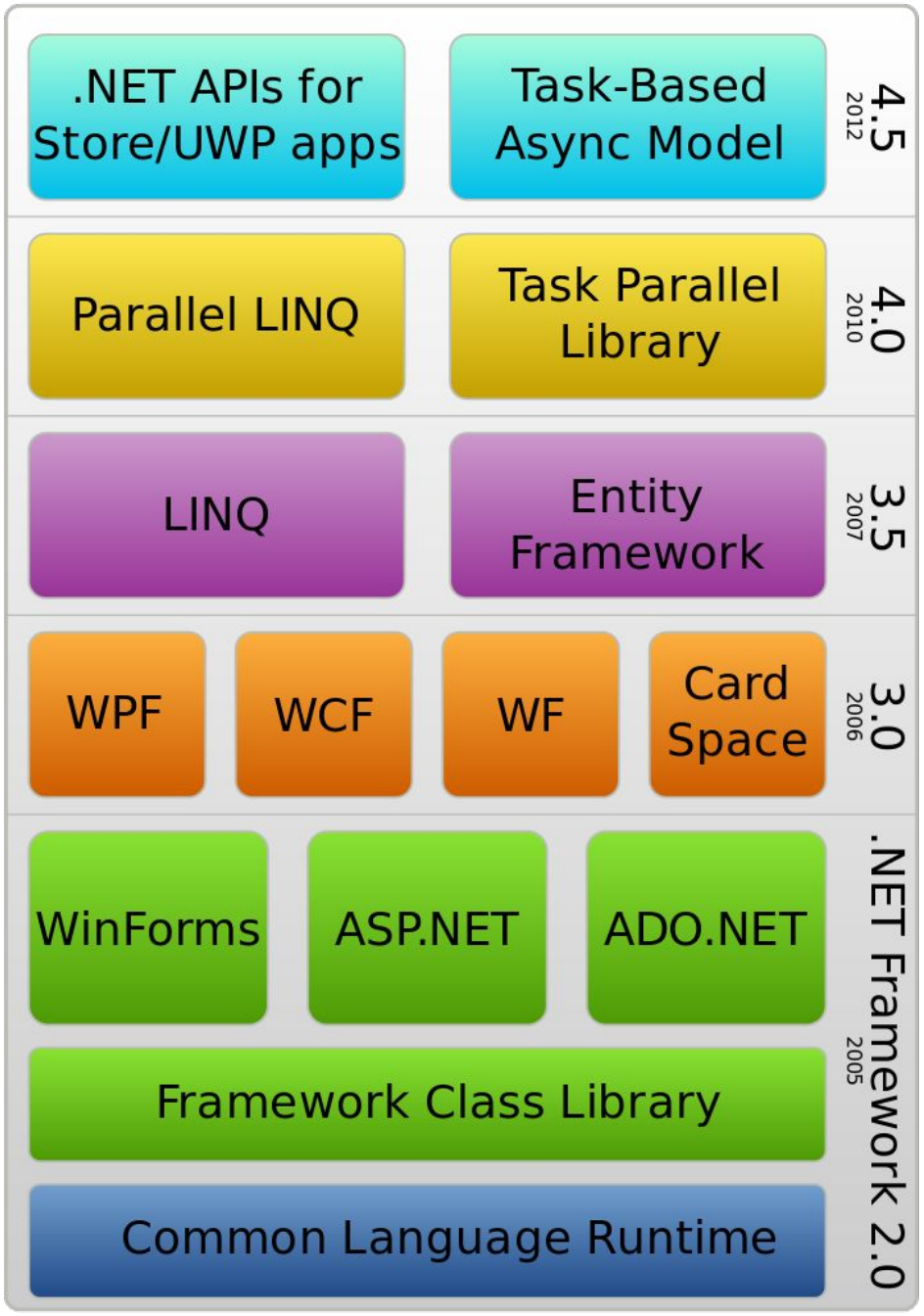
История

COM (*Component Object Model)*



.Net Framework

- Поддержка многочисленных языков программирования.
- Обширная библиотека базовых классов.
- Общий исполняющий механизм, разделяемый всеми поддерживающими .NET языками.
- Языковая интеграция.


.NET APIs for Store/UWP apps
Task-Based Async Model
4.5
2012
Parallel LINQ
Task Parallel Library
4.0
2010
LINQ
Entity Framework
3.5
2007
WPF
WCF
WF
Card Space
3.0
2006
WinForms
ASP.NET
ADO.NET
Framework Class Library
Common Language Runtime
.NET Framework 2.0
2005

Основные компоненты

- CLR (Common Language Runtime)
- BCL (Base Class Library) или FCL (Framework Class Library)
- CTS (Common Type System)
- CLS (Common Language Specification)

CLR

- Название среды — общеязыковая среда выполнения (Common Language Runtime, CLR) — говорит само за себя: это среда выполнения, которая подходит для разных языков программирования. Основные возможности CLR (управление памятью, загрузка сборок, безопасность, обработка исключений, синхронизация) доступны в любых языках программирования, использующих эту среду

- Исходный код программы может быть написан на любом языке, поддерживающем среду выполнения CLR. Затем соответствующий компилятор проверяет синтаксис и анализирует исходный код программы. Вне зависимости от типа используемого компилятора результатом компиляции будет являться управляемый модуль (managed module) — стандартный переносимый исполняемый (portable executable, PE) файл 32-разрядной (PE32) или 64-разрядной Windows (PE32+), [illegible] лнения CLR

Рис. 1.1. Компиляция исходного кода в управляемые модули

Таблица 1.1. Части управляемого модуля

Часть	Описание
Заголовок PE32 или PE32+	Стандартный заголовок PE-файла Windows, аналогичный заголовку Common Object File Format (COFF). Файл с заголовком в формате PE32 может выполняться в 32- и 64-разрядной версиях Windows, а с заголовком PE32+ — только в 64-разрядной. Заголовок обозначает тип файла: GUI, CUI или DLL, он также имеет временную метку, показывающую, когда файл был собран. Для модулей, содержащих только IL-код, основной объем информации в заголовке PE32(+) игнорируется. В модулях, содержащих машинный код, этот заголовок содержит сведения о машинном коде
Заголовок CLR	Содержит информацию (интерпретируемую CLR и утилитами), которая превращает этот модуль в управляемый. Заголовок включает нужную версию CLR, некоторые флаги, метку метаданных `MethodDef` точки входа в управляемый модуль (метод `Main`), а также месторасположение/размер метаданных модуля, ресурсов, строгого имени, некоторых флагов и пр.
Метаданные	Каждый управляемый модуль содержит таблицы метаданных. Есть два основных вида таблиц — это таблицы, описывающие типы данных и их члены, определенные в исходном коде, и таблицы, описывающие типы данных и их члены, на которые имеются ссылки в исходном коде
Код Intermediate Language (IL)	Код, создаваемый компилятором при компиляции исходного кода. Впоследствии CLR компилирует IL в машинные команды

Все CLR-совместимые компиляторы генерируют IL-код.IL-код иногда называют управляемым (managed code), потому что CLR управляет его выполнением.

Каждый компилятор, предназначенный для CLR, помимо генерирования IL-кода, должен также создавать полные метаданные (metadata) для каждого управляемого модуля

Метаданные — это набор таблиц данных, описывающих то, что определено в модуле, например типы и их члены. В метаданных также есть таблицы, указывающие, на что ссылается управляемый модуль, например на импортируемые типы и их члены.

На самом деле среда CLR работает не с модулями, а со сборками.

Преимущества метаданных

Метаданные устраняют необходимость в заголовочных и библиотечных файлах при компиляции, так как все сведения об упоминаемых типах/членах содержатся в файле с реализующим их IL-кодом. Компиляторы могут читать метаданные прямо из управляемых модулей.

Среда Microsoft Visual Studio использует метаданные для облегчения написания кода. Ее функция IntelliSense анализирует метаданные и сообщает, какие методы, свойства, события и поля предпочтительны в данном случае и какие именно параметры требуются конкретным методам

В процессе верификации кода CLR использует метаданные, чтобы убедиться, что код совершает только «безопасные по отношению к типам» операции.

Метаданные позволяют сборщику мусора отслеживать жизненный цикл объектов. При помощи метаданных сборщик мусора может определить тип объектов и узнать, какие именно поля в них ссылаются на другие объекты

Сборка (assembly) — это абстрактное понятие, понять смысл которого на первых порах бывает нелегко.

Во-первых, сборка обеспечивает логическую группировку одного или нескольких управляемых модулей или файлов ресурсов

Во-вторых, это наименьшая единица многократного использования, безопасности и управления версиями

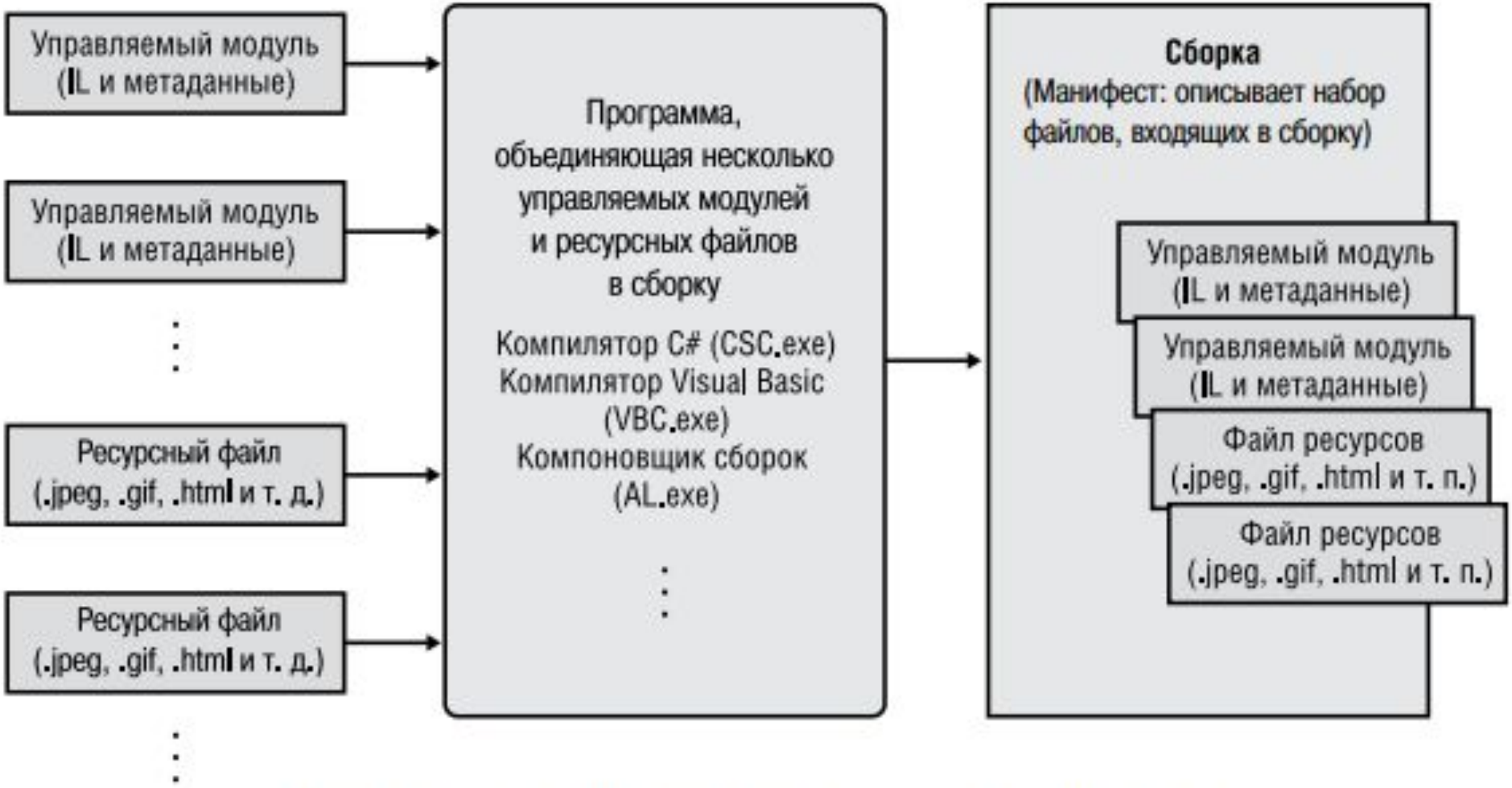


Рис. 1.2. Объединение управляемых модулей в сборку

Манифест представляет собой обычный набор таблиц метаданных. Эти таблицы описывают файлы, которые входят в сборку, общедоступные экспортируемые типы, реализованные в файлах сборки, а также относящиеся к сборке файлы ресурсов или данных.

Модули сборки также содержат сведения о других сборках, на которые они ссылаются (в том числе номера их версий). Эти данные делают сборку самоописываемой (self-describing).

Во-первых, сборка обеспечивает логическую группировку одного или нескольких управляемых модулей или файлов ресурсов

JIT (Just in Time) компилятор

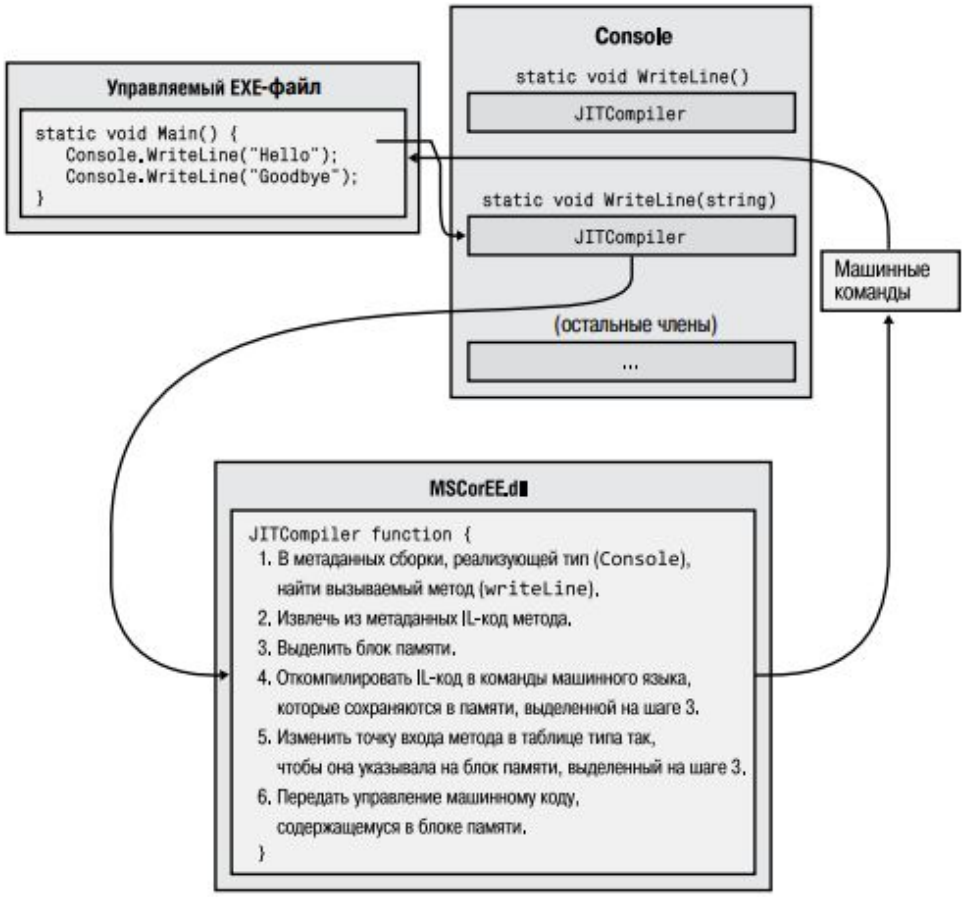


Рис. 1.4. Первый вызов метода

Когда метод `Main` первый раз обращается к методу `WriteLine`, вызывается функция `JITCompiler`. Она отвечает за компиляцию IL-кода вызываемого метода в собственные команды процессора. Поскольку IL-код компилируется непосредственно перед выполнением («just in time»), этот компонент CLR часто называют *JIT-компилятором*.

ПРИМЕЧАНИЕ

Если приложение исполняется в x86 версии Windows или в режиме WoW64, JIT-компилятор генерирует команды для архитектуры x86. Для приложений, выполняемых как 64-разрядные в версии x64 Windows, JIT-компилятор генерирует команды x64. Наконец, если приложение выполняется в ARM-версии Windows, JIT-компилятор генерирует инструкции ARM.

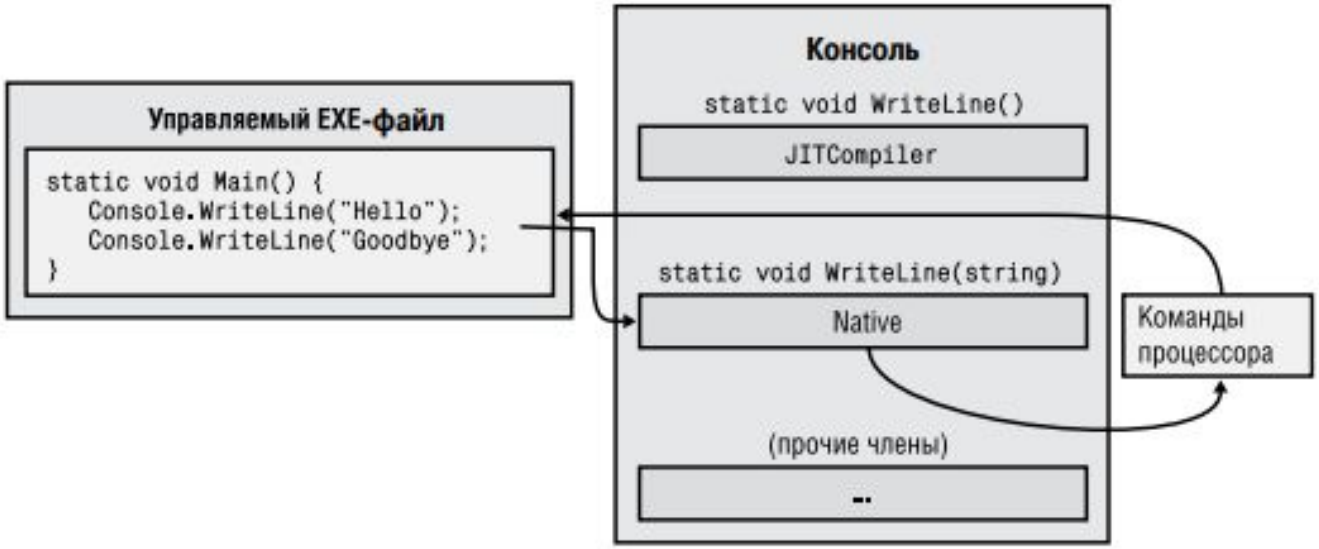
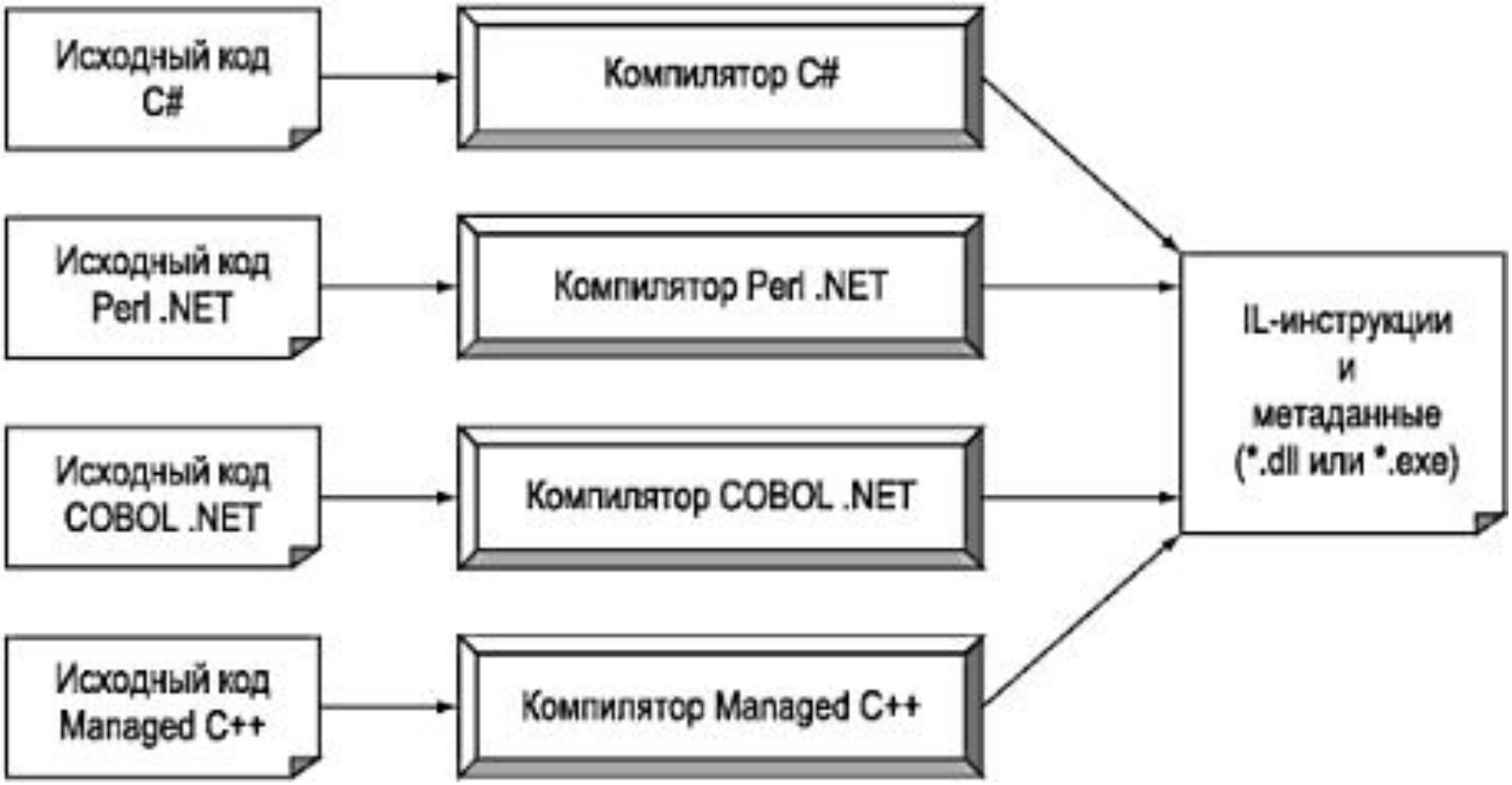


Рис. 1.5. Повторный вызов метода

Языки .Net

- C#
- C++/CLI
- Visual Basic
- F#
- Iron Python
- Iron Ruby
- …

Компиляция кода



Сборка(assembly)

Сборка представляет собой самоописываемый двоичный файл, обслуживаемый CLR.

Содержит:

- Манифест
- IL код
- Ресурсы

Метаданные

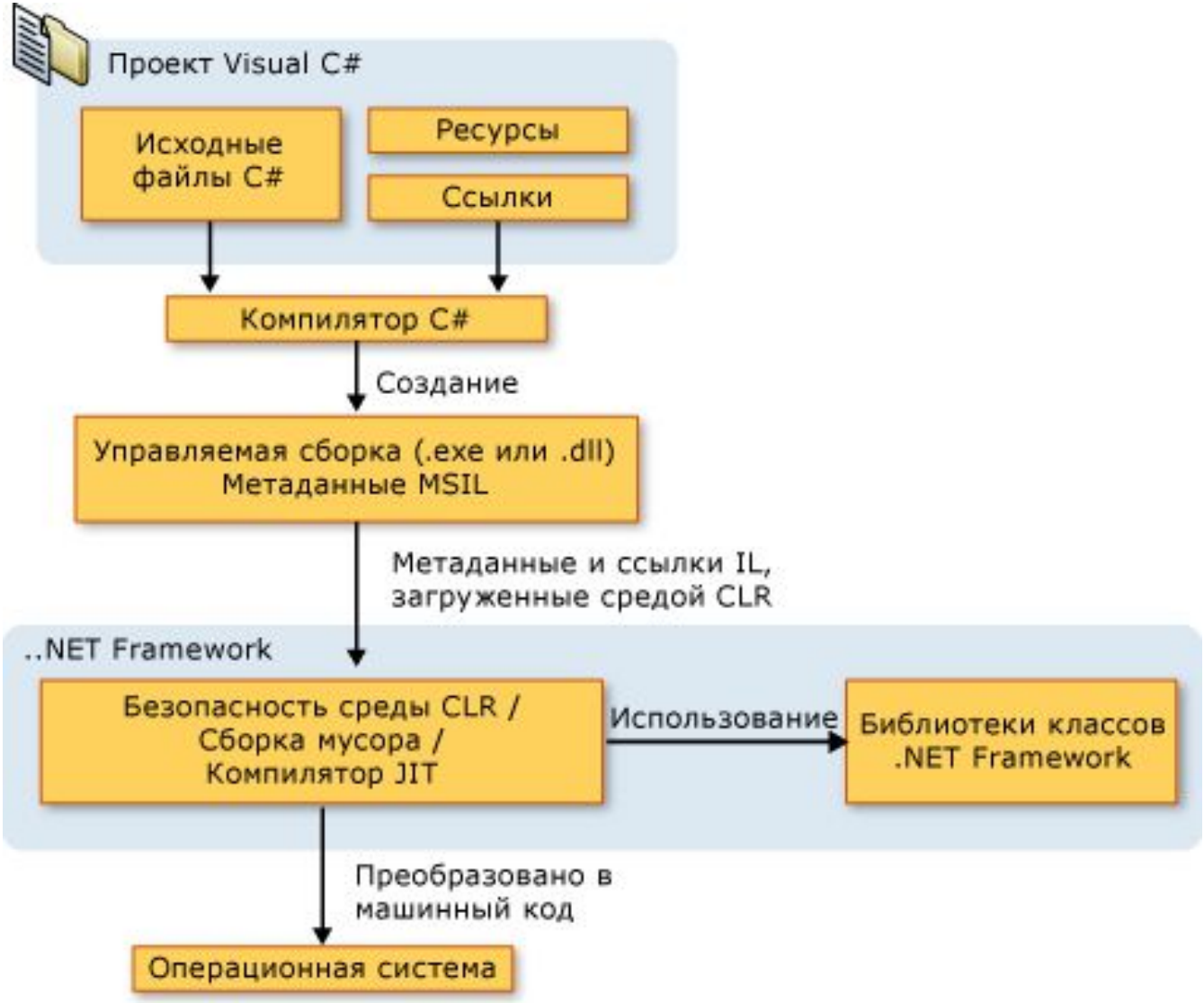
Данные о данных

- Метаданные устраняют необходимость в заголовочных файлах
- Верификации кода CLR использует метаданные
- Метаданные позволяют сборщику мусора отслеживать жизненный цикл объектов.

Манифест

Манифест это метаданные описывающие сборку.

Выполнение


Проект Visual C#
Исходные файлы C#
Ресурсы
Ссылки
Компилятор C#
Создание
Управляемая сборка (.exe или .dll)
Метаданные MSIL
Метаданные и ссылки IL, загруженные средой CLR
..NET Framework
Безопасность среды CLR / Сборка мусора / Компилятор JIT
Использование
Библиотеки классов .NET Framework
Преобразовано в машинный код
Операционная система

FCL(Framework Class Library)

Набор сборок в формате DLL, содержащих несколько тысяч определений типов, каждый из которых предоставляет некоторую функциональность.

CTS (Common Type System)

Формальная спецификация, описывающая способ определения и поведение типов.

CLS (Common Language Speciication)

Спецификация, в которой перечислен минимальный набор возможностей, которые должны поддерживаться компилятором для генерирования типов, совместимых с другими компонентами, написанными на других CLS-совместимых языках.

CLR & CTS