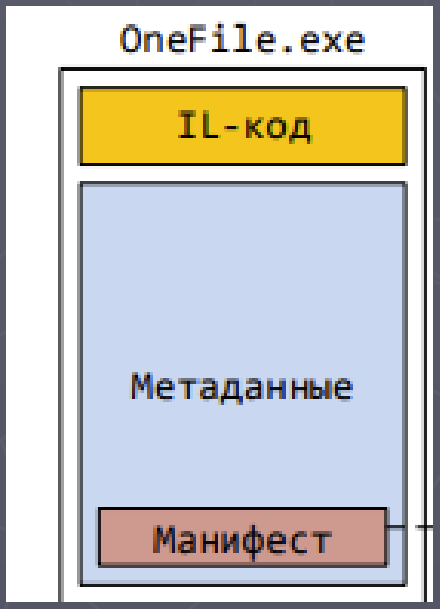
Рефлексия

Рефлексия

- Процесс выявления типов во время выполнения приложения

System.Reflection

При создании сборки в неё помещаются метаданные, которые являются описанием всех типов в сборке и их элементов



основные классы

Класс	Назначение
Assembly	сборка манипулирование этой сборкой
AssemblyName	информация о сборке
EventInfo	информация о событии
FieldInfo	информация о поле
MethodInfo	информация о методе
PropertyInfo **ConstructorInfo**	информация о свойстве, конструкторе
Module	доступ к определенному модулю внутри сборки
ParameterInfo:	класс, хранящий информацию о параметре метода

System.Type

- класс, позволяет получить информацию о членах типа.
представляет изучаемый тип, инкапсулируя всю информацию о нем.
его свойства:

Name возвращает имя типа
Assembly возвращает название сборки, где определен тип
Namespace возвращает название пространства имен, где определен тип
IsArray возвращает true, если тип является массивом
IsClass возвращает true, если тип представляет класс
IsEnum возвращает true, если тип является перечислением
IsInterface возвращает true, если тип представляет интерфейс

- **FindMembers()** - воз. массив объектов MemberInfo данного типа
- **GetConstructors()** - конструкторы данного типа в виде набора объектов ConstructorInfo
- **GetEvents()** - события данного типа в виде массива объектов EventInfo
- **GetFields()** - поля данного типа в виде массива объектов FieldInfo
- **GetInterfaces()** - реализуемые данным типом интерфейсы в виде массива объектов Type
- **GetMembers()** -члены типа в виде массива объектов MemberInfo
- **GetMethods()** - методы типа в виде массива объектов MethodInfo
- **GetProperties()** - свойства в виде массива объектов PropertyInfo

получить данный тип

- с помощью оператора typeof,
- с помощью метода GetType() класса Object
- применяя статический метод Type.GetType().

```
Type myType = typeof(Person);

Console.WriteLine(myType);  // Person

public class Person
{
    public string Name { get;}
    public Person(string name) => Name = name;
}
```

надо создавать объект класса

```
Person tom = new Person("Tom");
Type myType = tom.GetType();
```

```
Type? myType = Type.GetType("Person", false, true);
```

будет ли генерироваться исключение, если класс не удастся найти

надо ли учитывать регистр символов в первом параметре. true означает, что регистр игнорируется.

тип располагается в другом пространстве имен, его также надо указать:

```
Type? myType = Type.GetType("PeopleTypes.Person", false, true);

Console.WriteLine(myType);  // PeopleTypes.Person

namespace PeopleTypes
{
    public class Person
    {
        public string Name { get;}
        public Person(string name) => Name = name;
    }
}
```

находится в другой сборке dll, то после полного имени класса через запятую указывается имя сборки:

```
Type myType = Type.GetType("PeopleTypes.Person, MyLibrary", false, true);
```

```
Type myType = typeof(PeopleTypes.Person);

Console.WriteLine($"Name: {myType.Name}");                  // получаем краткое имя типа
Console.WriteLine($"Full Name: {myType.FullName}");         // получаем полное имя типа
Console.WriteLine($"Namespace: {myType.Namespace}");        // получаем пространство имен т
Console.WriteLine($"Is struct: {myType.IsValueType}");      // является ли тип структурой
Console.WriteLine($"Is class: {myType.IsClass}");           // является ли тип классом

namespace PeopleTypes
{
    class Person
    {
        public string Name { get; }
        public Person(string name) => Name = name;
    }
}
```

```
Name: Person
Full Name: PeopleTypes.Person
Namespace: PeopleTypes
Is struct: False
Is class: True
```

Поиск реализованных интерфейсов. GetInterfaces()

```
Type myType = typeof(Person);

Console.WriteLine("Реализованные интерфейсы:");
foreach (Type i in myType.GetInterfaces())
{
    Console.WriteLine(i.Name);
}

public class Person : IEater, IMovable
{
    public string Name { get;}
    public Person(string name) => Name = name;
    public void Eat() => Console.WriteLine($"{Name} eats");

    public void Move()=> Console.WriteLine($"{Name} moves");
}
interface IEater
{
    void Eat();
}
interface IMovable
{
    void Move();
}
```

возвращает массив объектов Type

Применение рефлексии

```
using System.Reflection;
```

```
Type t = typeof(Int32);
 Console.WriteLine("Full name = " + t.FullName);
 Console.WriteLine("Base type is = " + t.BaseType);
 Console.WriteLine("Is sealed = " + t.IsSealed);
 Console.WriteLine("Is class = " + t.IsClass);
 foreach (Type iType in t.GetInterfaces()) {
     Console.WriteLine(iType.Name);
 }
 foreach (FieldInfo fi in t.GetFields()) {
     Console.WriteLine("Field = " + fi.Name);
 }
```

```
Full name = System.Int32
Base type is = System.ValueType
Is sealed = True
Is class = False
IComparable
IFormattable
IConvertible
IComparable`1
IEquatable`1
Field = MaxValue
Field = MinValue
```

Получение всех компонентов типа. GetMembers()

возвращает все доступные компоненты типа в виде объекта MemberInfo.

Этот объект позволяет извлечь некоторую информацию о компоненте типа.

DeclaringType: возвращает полное название типа.

MemberType: возвращает значение из перечисления MemberTypes:

MemberTypes.Constructor, MemberTypes.Method, MemberTypes.Field, MemberTypes.Event, MemberTypes.Property, MemberTypes.NestedType

Name: возвращает название компонента

```
using System.Reflection;      // подключаем функционал рефлексии

Type myType = typeof(Person);

foreach (MemberInfo member in myType.GetMembers())
{
    Console.WriteLine($"{member.DeclaringType} {member.MemberType} {member.Name}");
}

public class Person
{
    string name;
    public int Age { get; set; }
    public Person(string name, int age)
    {
        this.name = name;
        this.Age = age;
    }
    public void Print() => Console.WriteLine($"Name: {name} Age: {Age}");
}
```

получим все общедоступные члены класса Person

получаем только все публичные компоненты класса, и нам не выводится информация о приватной переменной name

```
Person Method get_Age
Person Method set_Age
Person Method Print
System.Object Method GetType
System.Object Method ToString
System.Object Method Equals
System.Object Method GetHashCod
Person Constructor .ctor
Person Property Age
```

мы получаем весь функционал, в том числе унаследованный от базовых классов (в данном случае функционал Object).

```
foreach (PropertyInfo pi in t.GetProperties()) {
        Console.WriteLine("Property = " + pi.Name);
    }
    foreach (MethodInfo mi in t.GetMethods()) {
        Console.WriteLine("Method Name = " + mi.Name);
        Console.WriteLine("Method Return Type = " + mi.ReturnType);
        foreach (ParameterInfo pr in mi.GetParameters()) {
            Console.WriteLine("Parameter Name = " + pr.Name);
            Console.WriteLine("Type = " + pr.ParameterType);
        }
    }
```

покажет данные только об открытых элементах типа

```
Method Name = TryParse
Method Return Type = System.Boolean
Parameter Name = s
Type = System.String
Parameter Name = result
Type = System.Int32&
Method Name = TryParse
Method Return Type = System.Boolean
Parameter Name = s
Type = System.String
Parameter Name = style
Type = System.Globalization.NumberStyles
Parameter Name = provider
Type = System.IFormatProvider
Parameter Name = result
Type = System.Int32&
Method Name = GetTypeCode
Method Return Type = System.TypeCode
Method Name = GetType
Method Return Type = System.Type
```

```
Method Name = CompareTo
Method Return Type = System.Int32
Parameter Name = value
Type = System.Object
Method Name = CompareTo
Method Return Type = System.Int32
Parameter Name = value
Type = System.Int32
Method Name = Equals
Method Return Type = System.Boolean
Parameter Name = obj
Type = System.Object
Method Name = Equals
Method Return Type = System.Boolean
Parameter Name = obj
Type = System.Int32
Method Name = GetHashCode
Method Return Type = System.Int32
Method Name = ToString
Method Return Type = System.String
Method Name = ToString
Method Return Type = System.String
Parameter Name = format
Type = System.String
Method Name = ToString
Method Return Type = System.String
```

System.Reflection.BindingFlags

- Флаги BindingFlags, связанные с получением информации о типе

Флаг	Описание
Default	Отсутствие специальных флагов
IgnoreCase	Игнорировать регистр имён получаемых элементов
DeclaredOnly	Получить элементы, объявленные непосредственно в типе (игнорировать унаследованные элементы)
Instance	Получить экземплярные элементы
Static	Получить статические элементы
Public	Получить открытые элементы
NonPublic	Получить закрытые элементы
FlattenHierarchy	Получить public и protected элементы у типа и у всех его предков

Объединяя данные значения с помощью побитовой операции ИЛИ можно комбинировать вывод.

```
Type tt = typeof(Int32);
var bf =  BindingFlags.Public |
          BindingFlags.NonPublic |
          BindingFlags.Static |
          BindingFlags.Instance;

FieldInfo[] fi = tt.GetFields(bf);
```

```
using System.Reflection;      // подключаем функционал рефлексии

Type myType = typeof(Person);

foreach (MemberInfo member in myType.GetMembers(BindingFlags.DeclaredOnly
            | BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public))
{
    Console.WriteLine($"{member.DeclaringType} {member.MemberType} {member.Name}");
}

public class Person
{
    string name;
    public int Age { get; set; }
    public Person(string name, int age)
    {
        this.name = name;
        this.Age = age;
    }
    public void Print() => Console.WriteLine($"Name: {name} Age: {Age}");
}
```

```
Person Method get_Age
Person Method set_Age
Person Method Print
Person Constructor .ctor
Person Property Age
Person Field name
Person Field <Age>k__BackingField
```

Получение одного компонента по имени. GetMember(

метод , в который передается имя компонента. И опционально можно передать флаги BindingFlags.

```
Type myType = typeof(Person);

// получаем метод print
MemberInfo[] print = myType.GetMember("Print", BindingFlags.Instance | BindingFlags.Public
foreach (MemberInfo member in print)
{
    Console.WriteLine($"{member.MemberType} {member.Name}");
}
```

при получении одного члена типа опять же возвращается массив MemberInfo[], поскольку в классе может быть несколько элементов с одним именем, например, несколько перегруженных версий метода Print.

Получение информации о методах. GetMethods()

Этот метод возвращает все методы типа в виде массива объектов MethodInfo. Его свойства предоставляют информацию о методе

- IsAbstract: возвращает true, если метод абстрактный
- IsFamily: возвращает true, если метод имеет модификатор доступа protected
- IsFamilyAndAssembly: возвращает true, если метод имеет модификатор доступа private protected
- IsFamilyOrAssembly: возвращает true, если метод имеет модификатор доступа protected internal
- IsAssembly: возвращает true, если метод имеет модификатор доступа internal
- IsPrivate: возвращает true, если метод имеет модификатор доступа private
- IsPublic: возвращает true, если метод имеет модификатор доступа public
- IsConstructor: возвращает true, если метод предоставляет конструктор
- IsStatic: возвращает true, если метод статический
- IsVirtual: возвращает true, если метод виртуальный
- ReturnType: возвращает тип возвращаемого значения

методы MethodInfo;

GetMethodBody(): возвращает тело метода в виде объекта MethodBody

GetParameters(): возвращает массив параметров, где каждый параметр представлен объектом типа ParameterInfo

Invoke(): вызывает метод

```
Type myType = typeof(Printer);

Console.WriteLine("Методы:");
foreach (MethodInfo method in myType.GetMethods())
{
    string modificator = "";

    // если метод статический
    if (method.IsStatic) modificator += "static ";
    // если метод виртуальный
    if (method.IsVirtual) modificator += "virtual ";

    Console.WriteLine($"{modificator}{method.ReturnType.Name} {method.Name} ()")
}
class Printer
{
    public string DefaultMessage { get; set; } = "Hello";
    public void PrintMessage(string message, int times = 1)
    {
        while (times-- > 0) Console.WriteLine(message);
    }
    public string CreateMessage() => DefaultMessage;
}
```

в категорию методов также попадают и свойства, которые по сути представляют два метода: get и set.

```
Методы:
String get_DefaultMessage ()
Void set_DefaultMessage ()
Void PrintMessage ()
String CreateMessage ()
Type GetType ()
virtual String ToString ()
virtual Boolean Equals ()
virtual Int32 GetHashCode ()
```

получим только методы самого класса без унаследованных, как публичные, так и все остальные:

```
ype myType = typeof(Printer);

onsole.WriteLine("Методы:");
oreach (MethodInfo method in myType.GetMethods(BindingFlags.DeclaredOnly
          | BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public))

   Console.WriteLine($"{method.ReturnType.Name} {method.Name} ()");

lass Printer

   public string DefaultMessage { get; set; } = "Hello";
   protected internal void PrintMessage(string message, int times = 1)
   {
       while (times-- > 0) Console.WriteLine(message);
   }
   private string CreateMessage() => DefaultMessage;
```

получаем все методы экземпляра, как публичные, так и непубличные, но исключаем статические

```
Методы:
String get_DefaultMessage ()
Void set_DefaultMessage ()
Void PrintMessage ()
String CreateMessage ()
```

Исследование параметров. GetParameters()

можно получить все параметры метода в виде массива объектов ParameterInfo.

Attributes: возвращает атрибуты параметра

DefaultValue: возвращает значение параметра по умолчанию

HasDefaultValue: возвращает true, если параметр имеет значение по умолчанию

IsIn: возвращает true, если параметр имеет модификатор in

IsOptional: возвращает true, если параметр является необязательным

IsOut: возвращает true, если параметр является выходным, то есть имеет модификатор out

Name: возвращает имя параметра

ParameterType: возвращает тип параметра

```
foreach (MethodInfo method in typeof(Printer).GetMethods())
{
    Console.Write($"{method.ReturnType.Name} {method.Name} (");
    //получаем все параметры
    ParameterInfo[] parameters = method.GetParameters();
    for (int i = 0; i < parameters.Length; i++)
    {
        var param = parameters[i];
        // получаем модификаторы параметра
         string modificator = "";
        if (param.IsIn) modificator = "in";
        else if (param.IsOut) modificator = "out";

        Console.Write($"{param.ParameterType.Name} {modificator} {param.Name}");
        // если параметр имеет значение по умолчанию
        if (param.HasDefaultValue) Console.Write($"={param.DefaultValue}");
        // если не последний параметр, добавляем запятую
        if (i < parameters.Length - 1) Console.Write(", ");
    }
    Console.WriteLine(")");
}
```

```
class Printer
{
    public void PrintMessage(string message, int times = 1)
    {
        while (times-- > 0) Console.WriteLine(message);
    }
     public void CreateMessage(out string message) => message = "Hello Metanit.com";
}
```

```
Void PrintMessage (String  message, Int32  times=1)
Void CreateMessage (String& out message)
Type GetType ()
String ToString ()
Boolean Equals (Object  obj)
Int32 GetHashCode ()
```

если параметр имеет модификатор ref, in, out, то в конце названия типа добавляется амперсанд - String&.

Вызов методов. Invoke()

```
public object? Invoke (object? obj, object?[]? parameters);
```

- Первый параметр представляет объект, для которого вызывается метод.
- Второй объект представляет массив значений, которые передаются параметрам метода.
- метод может возвращать результат в виде значения object?.

```
using System.Reflection;

var myPrinter = new Printer("Hello");

// получаем метод Print
var print = typeof(Printer).GetMethod("Print");
// вызываем метод Print
print?.Invoke(myPrinter, parameters: null); // Hello

class Printer
{
    public string Text { get;}
    public Printer(string text) => Text = text;
    public void Print() => Console.WriteLine(Text);
}
```

метод не принимает параметров, параметру parameters передается значение null.

Получение конструкторов. ConstructorInfo

возвращает массив объектов класса ConstructorInfo.

Свойство IsFamily: возвращает true, если конструктор имеет модификатор доступа protected

Свойство IsFamilyAndAssembly: возвращает true, если конструктор имеет модификатор доступа private protected

Свойство IsFamilyOrAssembly: возвращает true, если конструктор имеет модификатор доступа protected internal

Свойство IsAssembly: возвращает true, если конструктор имеет модификатор доступа internal

Свойство IsPrivate: возвращает true, если конструктор имеет модификатор доступа private

Свойство IsPublic: возвращает true, если конструктор имеет модификатор доступа public

Получение конструкторов. ConstructorInfo

Метод GetMethodBody(): возвращает тело конструктора в виде объекта MethodBody

Метод GetParameters(): возвращает массив параметров, где каждый параметр представлен объектом типа ParameterInfo

Метод Invoke(): вызывает конструктор

```
Console.WriteLine("Конструкторы:");
foreach (ConstructorInfo ctor in myType.GetConstructors(
    BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public))
{
    string modificator = "";

    // получаем модификатор доступа
    if (ctor.IsPublic)
        modificator += "public";
    else if (ctor.IsPrivate)
        modificator += "private";
    else if (ctor.IsAssembly)
        modificator += "internal";
    else if (ctor.IsFamily)
        modificator += "protected";
    else if (ctor.IsFamilyAndAssembly)
        modificator += "private protected";
    else if (ctor.IsFamilyOrAssembly)
        modificator += "protected internal";

    Console.Write($"{modificator} {myType.Name}(");
    // получаем параметры конструктора
    ParameterInfo[] parameters = ctor.GetParameters();
    for (int i = 0; i < parameters.Length; i++)
    {
        var param = parameters[i];
        Console.Write($"{param.ParameterType.Name} {param.Name}");
        if (i < parameters.Length - 1) Console.Write(", ");
    }
```

```
Type myType = typeof(Person);
```

```
class Person
{
    public string Name { get; }
    public int Age { get; }
    public Person(string name, int age)
    {
        Name = name; Age = age;
    }
    public Person(string name) : this(name, 1) { }
    private Person() : this("Tom") { }
}
```

```
Конструкторы:
public Person(String name, Int32 age)
public Person(String name)
private Person()
```

Получение информации о полях. GetFields()

возвращает массив объектов класса FieldInfo.

Метод GetValue(): возвращает значение поля
Метод SetValue(): устанавливает значение поля

```
Console.WriteLine("Поля:");
foreach (FieldInfo field in myType.GetFields(
    BindingFlags.Instance | BindingFlags.NonPublic | BindingFlags.Public | BindingFl
{
    string modificator = "";

    // получаем модификатор доступа
    if (field.IsPublic)
        modificator += "public ";
    else if (field.IsPrivate)
        modificator += "private ";
    else if (field.IsAssembly)
        modificator += "internal ";
    else if (field.IsFamily)
        modificator += "protected ";
    else if (field.IsFamilyAndAssembly)
        modificator += "private protected ";
    else if (field.IsFamilyOrAssembly)
        modificator += "protected internal ";

    // если поле статическое
    if (field.IsStatic) modificator += "static ";

    Console.WriteLine($"{modificator}{field.FieldType.Name} {field.Name}");
}
```

Чтобы получить и статические, и не статические, и публичные, и непубличные поля

```
class Person
{
    static int minAge = 0;
    string name;
    int age;
    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
    public void Print() => Console.WriteLin
}
```

Поля:
private String name
private Int32 age
private static Int32 minAge

Получение и изменение значения поля. GetField()

Для получения одного поля по имени, в который передается имя поля.

рефлексия позволяет получать значения и изменять их даже у приватных полей.

```
class Person
{
    static int minAge = 1;
    string name;
    int age;
    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
    public void Print() => Console.WriteLine($"{name} - {age}");
}
```

```
Type myType = typeof(Person);
Person tom = new Person("Tom", 37);

// получаем приватное поле name
var name = myType.GetField("name", BindingFlags.Instance | BindingFlags.NonPublic);

// получаем значение поля name
var value =name?.GetValue(tom);
Console.WriteLine(value);   // Tom

// изменяем значение поля name
name?.SetValue(tom, "Bob");
tom.Print();    // Bob - 37
```

Свойства GetProperties()

Для извлечения всех свойств типа. возвращает массив объектов PropertyInfo

Для получения одного свойства по имени применяется метод GetProperty(), в который передается название свойства и который возвращает объект PropertyInfo?.

Свойство Attributes: возвращает коллекцию атрибутов свойства

Свойство CanRead: возвращает true, если свойство доступно для чтения

Свойство CanWrite: возвращает true, если свойство доступно для записи

Свойство GetMethod: возвращает get-акссесор в виде объекта MethodInfo?

Свойство SetMethod: возвращает set-акссесор в виде объекта MethodInfo?

Свойство PropertyType: возвращает тип свойства

Метод GetValue(): возвращает значение свойства

Метод SetValue(): устанавливает значение свойства

Используя PropertyInfo можно манипулировать значением свойства. получим и изменим значение свойства

```
Type myType = typeof(Person);
Person tom = new Person("Tom", 37);
// получаем свойство Age
var ageProp = myType.GetProperty("Age");
// получаем значение свойства Age у объекта tom
var age = ageProp?.GetValue(tom);
Console.WriteLine(age); // 37
// устанавливаем новое значение для свойства Age объекта tom
ageProp?.SetValue(tom, 22);
tom.Print();     // Tom - 22
```

Assembly. Для управления сборками

можно загружать сборку, исследовать ее.

Чтобы динамически загрузить сборку в приложение, надо использовать статические методы Assembly.LoadFrom() или Assembly.Load().

Метод LoadFrom() принимает в качестве параметра путь к сборке.

Основные элементы класса Assembly

Имя элемента	Описание
`CreateInstance()`	Находит по имени тип в сборке и создаёт его экземпляр
`FullName`	Строковое свойство с полным именем сборки
`GetAssembly()`	Ищет в памяти и возвращает объект `Assembly`, который содержит указанный тип (статический метод)
`GetCustomAttributes()`	Получает атрибуты сборки
`GetExecutingAssembly()`	Возвращает сборку, которая содержит выполняемый в текущий момент код (статический метод)
`GetExportedTypes()`	Возвращает `public`-типы, определённые в сборке
`GetFiles()`	Возвращает файлы, из которых состоит сборка
`GetLoadedModules()`	Возвращает все загруженные в память модули сборки
`GetModule()`	Получает указанный модуль сборки
`GetModules()`	Возвращает все модули, являющиеся частью сборки
`GetName()`	Возвращает объект `AssemblyName` для сборки
`GetReferencedAssemblies()`	Возвращает объекты `AssemblyName` для всех сборок, на которые ссылается данная сборка
`GetTypes()`	Возвращает типы, определённые в сборке
`Load()`	Статический метод, который загружает сборку по имени
`LoadFrom()`	Статический метод; загружает сборку из указанного файла
`LoadModule()`	Загружает внутренний модуль сборки в память

Основные элементы класса Module

Имя элемента	Описание
`Assembly`	Свойство с указанием на сборку (объект `Assembly`) модуля
`FindTypes()`	Получает массив классов, удовлетворяющих заданному фильтру
`FullyQualifiedName`	Строка, содержащая полное имя и путь к модулю
`GetType()`	Пытается выполнить поиск указанного типа в модуле
`GetTypes()`	Возвращает все типы, определённые в модуле
`Name`	Строка с коротким именем модуля

в проекте MyApp, который компилируется в сборку MyApp.dll, имеется файл Program.cs со следующим кодом:

```
Person tom = new Person("Tom");
Console.WriteLine($"Hello, {tom.Name}");

class Person
{
    public string Name { get; }
    public Person(string name) => Name = name;
}
```

В другом проект исследуем сборку MyApp.dll на наличие в ней различных типов

```
using System.Reflection;

Assembly asm = Assembly.LoadFrom("MyApp.dll");

Console.WriteLine(asm.FullName);
// получаем все типы из сборки MyApp.dll
Type[] types = asm.GetTypes();
foreach (Type t in types)
{
    Console.WriteLine(t.Name);
}
```

полное название сборки

```
MyApp, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null
EmbeddedAttribute
NullableAttribute
NullableContextAttribute
Program
Person
```

содержит пять типов

Позднее связывание

- Механизм отражения позволяет реализовать на платформе .NET позднее связывание (late binding).
- Обозначает процесс динамической загрузки сборок и типов при работе приложения, создание экземпляров типов и работу с их элементами.
- позволяет создавать расширяемые приложения, когда дополнительный функционал программы неизвестен, и его могут подключить сторонние разработчики

```
Assembly asm = Assembly.LoadFrom("Data.exe");
```

- **System.Activator**
- **Activator.CreateInstance()** можно создавать экземпляры заданного типа

можно создавать экземпляры заданного типа

```
Assembly asm = Assembly.LoadFrom("Data.exe");

Type typ = asm.GetType("Data.Program");

// создаем экземпляр класса Program
object obj = Activator.CreateInstance(typ);

// получаем метод GetArray
MethodInfo method = typ.GetMethod("GetArray");

// вызываем метод, передаем значения для параметров и получаем
object result = method.Invoke(obj, new object[] { 6, 100, 3 });
    Console.WriteLine((result));
```

получаем тип

создаем его экземпляр

получаем сам метод

вызываем его

объект, для которого вызывается метод + набор параметров