При создании сборки в неё помещаются *метаданные*, которые являются описанием всех типов в сборке и их элементов. Программист может работать с метаданными в программе, используя специальный механизм, называемый *отражением* (*reflection*). Главные элементы, которые необходимы для использования возможностей отражения – класс `System.Type` и типы из пространств имен `System.Reflection` и `System.Reflection.Emit`.

Класс `System.Type` служит для получения информации о типе. Чтобы получить объект этого класса, существует несколько возможностей.

1. Вызвать у объекта метод `GetType()`. Данный метод определен на уровне `System.Object`, а значит, присутствует у любого объекта:

```
Foo A = new Foo();     // Foo – это некий класс
Type t = A.GetType();
```

2. Использовать статический метод `Type.GetType()`, которому передается имя типа в виде строки:

```
Type t = Type.GetType("Foo");
```

3. Использовать операцию C# `typeof`, параметром которой является тип:

```
Type t = typeof(Foo);
```

Элементы класса `Type` позволяют узнать имя типа, имя базового типа, является ли тип универсальным, в какой сборке он размещается и другую информацию. Кроме этого, имеются специальные методы, возвращающие данные о полях, свойствах, событиях, методах и их параметрах.

Как показывает пример, информация об элементах типа хранится в объектах специальных классов (`FieldInfo`, `PropertyInfo`, `MethodInfo`). Эти классы находятся в пространстве имен `System.Reflection`. Кроме этого, код примера выведет информацию только об открытых элементах типа. Чтобы управлять набором получаемых данных, можно использовать перечисление `BindingFlags`. Методы класса `Type` имеют соответствующие перегруженные версии.

```
var bf = BindingFlags.Public | BindingFlags.NonPublic |
         BindingFlags.Static | BindingFlags.Instance;
FieldInfo[] fi = t.GetFields(bf);
```

Пространство имен `System.Reflection` содержит типы для получения информации и манипулирования сборкой и модулем сборки. При помощи класса `Assembly` можно получить информацию о сборке, при помощи класса `Module` - о модуле. Кроме этого, метод `Assembly.Load()` позволяет динамически загрузить определенную сборку в память во время работы приложения. Подобный подход называется *поздним связыванием*.

```
Assembly A = Assembly.Load("FooLib");
foreach (Module M in A.GetModules())
{
    foreach (Type T in M.GetTypes())
    { . . . }
}
```

При помощи позднего связывания можно создать объекты типов из сборки, а также работать с элементами созданных объектов (например, вызывать методы). Для создания объекта при позднем связывании можно применить метод `CreateInstance()` класса `System.Activator`. Класс `MethodInfo` имеет метод `Invoke()`, который позволяет вызвать метод созданного объекта. Первый параметр метода `Invoke()` – это тот объект, у которого вызывается метод, второй параметр – массив объектов-параметров метода.

Отражение и позднее связывание позволяют не только работать с существующим кодом, но также предоставляют средства для генерации инструкций промежуточного языка MSIL. Такая возможность востребована, в основном, разработчиками компиляторов. Средства генерации кода сосредоточены в пространстве имен `System.Reflection.Emit`.