

# Разработка приложений на платформе .NET

## Лекция 10

Сборки  
Reflection

# Сегодня

---

- ◎ Сборки
- ◎ Reflection

# Сегодня

---

- ◎ Сборки
- ◎ Reflection

# Сборка (Assembly)

---

- Базовая единица .NET – программы
  - Исполняемый файл (\*.exe)
  - Динамическая библиотека (\*.dll)
- Функции сборок
  - Содержать и группировать код
  - Единица безопасности
    - Определяют границы типов (инкапсуляция)
    - Имеют цифровую подпись (защита от подделок)
  - Единица развертывания
    - Возможна замена отдельных сборок при обновлении приложения
  - Единица версий
    - Сборки отличающиеся только версией могут сосуществовать на одном компьютере
- Сборки полностью себя описывают

# Содержимое сборки

---

- Windows заголовков
- CLR заголовков
- Манифест
  - Файлы сборки (один или несколько)
  - Требуемые сборки
  - Метаданные сборки
  - Список внешних ресурсов
- Метаданные типов
- Исполняемый IL код
- Ресурсы

# Демонстрации

---

Анализ сборки

Установка ссылки на сборку

Использование сборок на разных языках сборки

# Имена сборок

---

- ◎ Простое имя
  - Совпадает с именем файла без расширения
- ◎ Сильное имя (**strong name**)
  - Имя
  - Версия
  - Культура
  - Открытый ключ (+ цифровая подпись)
- ◎ Сильные имена уникальны

# Где находятся сборки

---

## ● Локально

- в каталоге приложения
- в подкаталоге с именем сборки
- в специально настроенном подкаталоге
- могут храниться любые сборки (с сильными и простыми именами)
- могут использоваться только этим приложением

## ● В глобальном кэше сборок (GAC, Global Assembly Cache)

- Только сборки со строгими именами
- Только библиотечные сборки (.dll)
- Могут использоваться разными программами
- Для установки требуются администраторские права

# Поиск нужной сборки

---

## ◎ Сильное имя?

- Нет:
  - Ищется `.dll` в текущей папке
  - Ищется `.exe` в текущей папке
  - Поиск `.dll` в подпапке с именем сборки
  - Поиск `.exe` в подпапке с именем сборки
- Да:
  - Ищется в **ГАС**
  - Если в **ГАС** нет, то ищется локально

# Достоинства сильных имен

---

- ◎ Нет конфликта версий (DLL Hell)
  - Сборки с одинаковыми именами могут существовать совместно
- ◎ Совместное использование
  - Меньше расход дискового пространства
- ◎ Безопасность
  - Цифровая подпись
    - Защита от подделки кода или сборки
    - Проверка изготовителя сборки

# Задание сильного имени сборке

---

- Подпись сборки
  - Генерация пары открытого и закрытого ключа
    - Утилита генерации пары ключей `sn.exe`
    - `sn -k имя_файла.snk`
  - Выбор файла с ключами в **Visual Studio** в разделе свойств проекта, подписывание
  - Оба шага можно выполнить в **Visual Studio** в разделе свойств проекта, подпись
- Установка версии сборки
  - Атрибут `assembly: AssemblyVersion` в файле свойств сборки `AssemblyInfo.cs`
  - Для автоматического увеличения номера версии сборки при компиляции можно задать `[assembly: AssemblyVersion("1.0.*")]`
- Задание необходимых полей для описания сборки.
  - Издатель и т.д.
  - В файле `AssemblyInfo.cs` или в свойствах проекта -> **Application- > Assembly Information**

# Демонстрации

---

Задание сильного имени сборке

# Установка в GAC

---

- Утилита `gacutil.exe`
- Установка сборки в GAC
  - `gacutil /i Имя_сборки`
- Удаление сборки из GAC
  - `gacutil /u Имя_сборки`
- Необходимы права администратора
  - и привилегии администратора в UAC
- GAC расположен в (скрытой) папке
  - `c:\Windows\Microsoft.NET\assembly`
  - До .NET 4 - `c:\Windows\assembly\`

# Демонстрации

---

Установка в ГАС

(возможна, только если есть права администратора)

# Сегодня

---

- ◎ Сборки
- ◎ **Reflection**

# Информация о типах

---

- Хранится в сборке
  - В виде метаданных
- Может быть извлечена
  - `ildasm.exe`
  - программно
- **Reflection** - получение информации о типах во время выполнения
- **Пространство имен System.Reflection**
  - Получение метаданных в виде объектной модели
    - Получение информации о типах, методах, свойствах, полях типа, параметров методов и т.д.
  - Динамическое управление загрузкой модулей
  - Динамическое создание экземпляров типов
  - Доступ к членам по именам
  - Вызов методов на “лету”, не зная о них в момент компиляции

# Объектная модель

---

- **Assembly** - представляет сборку
- **Type** - представляет тип
- **MemberInfo** – представляет любой член типа
- **ConstructorInfo** – представляет конструктор
- **MethodInfo** – представляет метод
  - **ParameterInfo** – представляет параметр
- **FieldInfo** – представляет поле
- **PropertyInfo** – представляет свойство
- **EventInfo** – представляет событие

# Загрузка сборок

---

## ● Автоматическая

- Загружает CLR
- В манифесте указана ссылка на сборку (добавляется при указании ссылки в проекте)
- Данные о сборке задаются на этапе разработки
- При отсутствии сборки – **Exception** при старте программы

## ● Динамическая

- Самостоятельная загрузка сборки
- Программа не содержит явных ссылок на загружаемую сборку
- Данные о сборке формируются в процессе работы программы
- Возможна мягкая обработка отсутствия сборки
- Программа и CLR при старте ничего не знает об динамически-загружаемой сборке

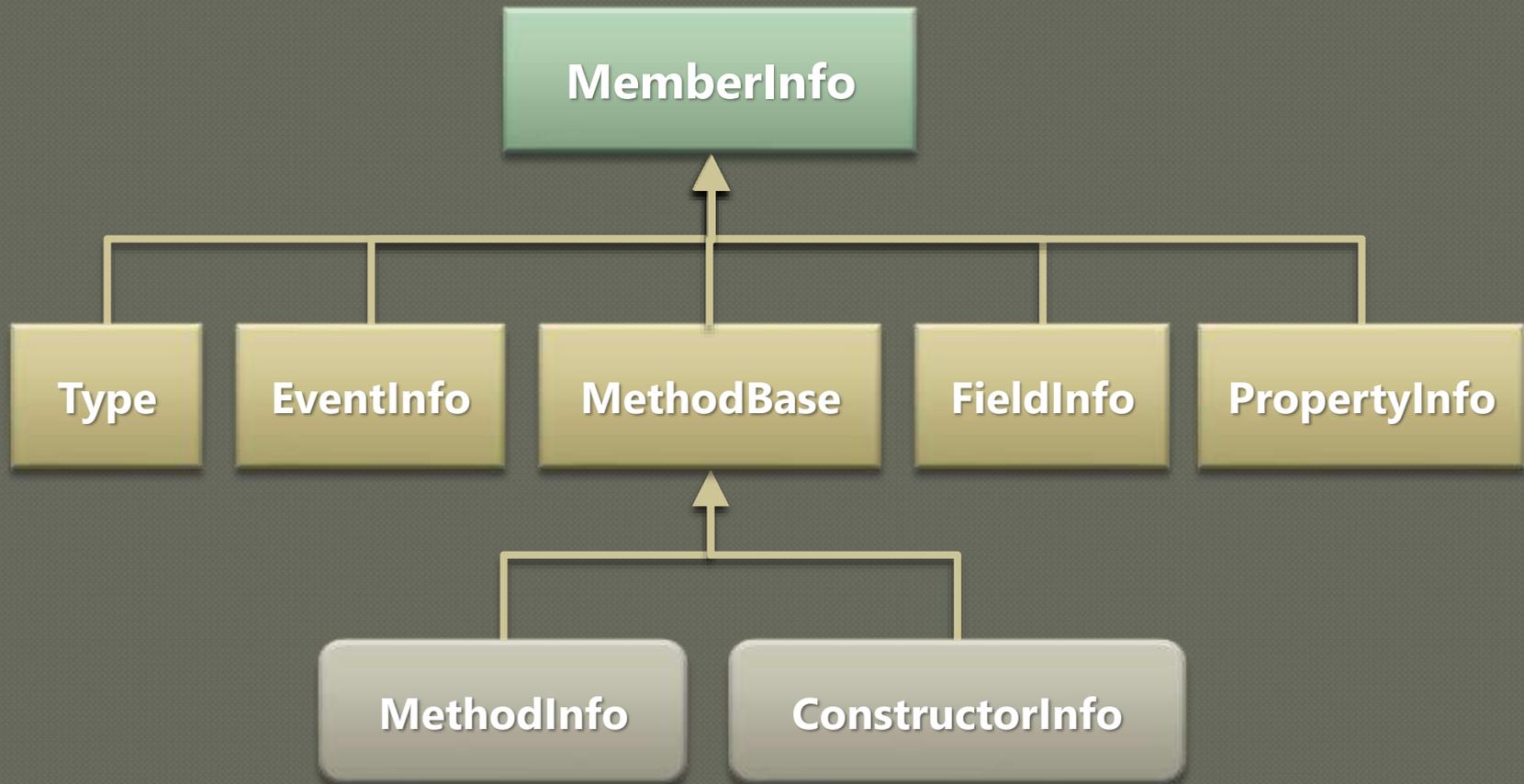
# Assembly

- Динамическая загрузка сборки (статические методы)
  - **Assembly.Load**("Имя сборки");
    - Assembly asm = Assembly.Load("Complex");
    - Assembly asm = Assembly.Load("Complex, Version=1.0.0.0, PublicKeyToken=null, Culture="");
    - Assembly asm = Assembly.Load ("SampleAssembly, Version=1.0.2004.0, Culture=neutral, PublicKeyToken=8744b20f8da049e3");
  - **Assembly.Load(AssemblyName assemblyRef)**
    - **AssemblyName** – представляет собой класс для удобного построения сильного имени сборки
  - **Assembly.LoadFile**("Полное\_имя\_файла")
    - Assembly asm = Assembly.LoadFile("c:\Complex.dll")
  - **Assembly.ReflectionOnlyLoad**("Имя сборки"); - загрузка только для анализа метаданных. Запрещено создавать типы и исполнять код
  - Сборки выгружаются только при закрытии домена приложения (автоматически)
- Важные методы и свойства:
  - Свойство **FullName** - Полное имя сборки:
  - Type[] **GetTypes()** – получение описания всех типов в сборке
  - Type **GetType**("ComplexClass.Complex"); - получение описания конкретного типа

# System.Type

- Описание типа
- Получение экземпляра типа `Type`:
  - `typeof (float), typeof (MyClass)`
  - `obj.GetType()`
  - `Type.GetType("полное_имя_типа")`
  - `assembly.GetType("Complex")`
- Методы (экземпляра)
  - `ConstructorInfo[] ci = t.GetConstructors()` – возвращает описания конструкторов
  - `MemberInfo[] mi = t.GetMembers();` – возвращает описания членов
  - `MethodInfo[] mi = t.GetMethods();` – возвращает описания методов
  - `PropertyInfo[] pi = t.GetProperties();` – возвращает описания свойств
  - `FieldInfo[] fi = t.GetFields();` – возвращает описания полей
  - `EventInfo[] fi = t.GetEvents();` – возвращает описания событий
  - И т.д.
  - Получение конкретного члена – без `-s` и с указанием имени члена
    - `FieldInfo mi = t.GetField("Real");`
  - `MemberInfo[] mi = t.FindMembers()` – поиск членов по критериям
- Свойства:
  - `Name, Namespace`
  - `IsAbstract, IsPublic, IsNotPublic, IsClass, IsArray, IsInterface, IsGenericType, IsNested, IsSealed, IsPrimitive, IsVisible`

# Информация о членах



# MemberInfo

---

- ◎ Базовый класс для описание членов типа
- ◎ Свойства
  - `string Name` – имя члена
  - `Type DeclaringType` – объявляющий тип
  - `Type ReflectedType` – тип, через который получен доступ (может быть типом наследником)
  - `MemberTypes MemberType` – тип члена. Перечисление `Method`, `Constructor`, `Event` и .т.д.

# FieldInfo

---

- Описание для полей и констант
- Получение статической информации
  - Свойство `Type` **FieldType** - Тип поля
  - **Name** – имя поля
  - **IsPublic**, **IsPrivate**, ... - доступ
  - **IsInitOnly** - поле объявлено как `readonly`)
  - **IsStatic** - статичность поля
  - И т.д.
- Работа со значениями на объектах
  - `object` **GetValue**(`object o`) – получение значения свойства у экземпляра типа
  - `void` **SetValue**(`object obj`, `object value`) – установка значения свойства для экземпляра типа

# MethodBase

- Базовый класс для:
  - **ConstructorInfo**
  - **MethodInfo**
- Информация о доступности и т.д.
  - **Name** – имя метода
  - **IsPublic, IsPrivate, ...** - доступ
  - **IsStatic** - статичность метода
  - **IsVirtual, IsAbstract, IsFinal** – виртуальный, абстрактный или не переопределяемый метод
  - **IsConstructor** – конструктор
  - И.т.д.
- Информация о параметрах
  - **ParameterInfo[] GetParameters()** – получение информации о входных параметрах
  - Тип **ParameterInfo**
    - **Name** – имя параметра
    - **IsIn / IsOut** – входной/выходной параметр
    - **IsRetVal** – возвращаемый параметр
    - **ParameterType** – тип параметра

# MethodInfo и ConstructorInfo

---

- ◎ ConstructorInfo - наследник от MethodBase
  - **ConstructorName** – имя метода конструктора класса
  - **TypeConstructorName** – имя типа конструктора класса
  
- ◎ MethodInfo - наследник от MethodBase
  - Тип **ReturnType** - Тип возвращаемого значения
  - ParameterInfo **ReturnParameter** – ParameterInfo возвращаемого параметра

# Демонстрации

---

Получение метаданных сборки

# Динамическое создание объекта

---

- Динамическое создание типа

## **Activator.CreateInstance()**

```
object o = Activator.CreateInstance(Тип, параметры  
конструктора);
```

```
Type type =
```

```
myAssembly.GetType("ComplexNameSpace.Complex");
```

```
object o = Activator.CreateInstance(type, 5, 7);
```

# Динамический вызов метода

---

- Динамический вызов метода

```
MethodInfo.Invoke(object obj, object [] parameters);  
MethodInfo mi = myType.GetMethod("ToString");  
object retValue = mi.Invoke(o, null);
```

- Динамический вызов статического метода

```
MethodInfo.Invoke(null, object [] parameters);
```

- Динамический вызов переопределенной операции – как вызов метода

```
MethodInfo miadd = type.GetMethod("op_Addition");  
object o3 = miadd.Invoke(null, new object[] { o1, o2 });
```

# Динамическая работа

---

## ○ Работа с полями и свойствами

- object **GetValue**(object o) – получение значения свойства у экземпляра типа
- void **SetValue**(object obj, object value) – установка значения свойства для экземпляра типа

## ○ Работа с событиями

```
EventInfo ei = type.GetEvent("MyEvent");  
ei.AddEventHandler(obj, myDelegate);  
ei.RemoveEventHandler(obj, myDelegate);
```

# Демонстрации

---

Динамическое создание типа и вызов его методов

# dynamic

- Dynamic Runtime Language (DLR)
- Перенос проверки типов и наличие членов типа с момента компиляции в момент времени выполнения
- Ключевое слово **dynamic**

```
myAssembly.GetType("ComplexNamespace.Complex");  
dynamic d = Activator.CreateInstance(type, 5, 7);  
d.Print();  
Console.WriteLine(d + d * d.Re);
```



- Удобно работать с Reflection, WinAPI, COM объектами и т.д.
- Удобно работать с динамическими языками программирования
- Есть примеры, которые возможны только при использовании **dynamic**

- Возможны ошибки в Runtime, а не в момент компиляции
- Тяжело отлаживать



- При разработке нет подсказок IntelliSense
- Загружаются доп. Сборки (DLR)
- Чуть медленнее

- Появилось в .NET 4

# Демонстрации

---

dynamic

# Сегодня

---

- ◎ Сборки
- ◎ Reflection