

Разработка приложений на платформе .NET

Лекция 10

Атрибуты
Сериализация

Сегодня

- Атрибуты
- Сериализация

Сегодня

- Атрибуты
- Сериализация

Атрибуты

- **Метаданные:**
 - Стандартные
 - Расширенные (атрибуты)
- **Атрибуты:**
 - Дополнительная метаинформация о сборке, типе, методе, свойстве, и т.д.
 - Добавляется декларативным образом
 - Доступ к атрибутам можно получить через Reflection
- **Использование**
 - Может учитываться во время компиляции
 - Может учитываться во время исполнения
 - Может учитываться средой разработки

Применение атрибутов

- В квадратных скобках перед целевым объектом:
 - Например, перед типом
`[Serializable]`
`public class Complex {...}`
 - Или перед методом:
`[System.Runtime.InteropServices.DllImport("user32.dll")]`
`extern static void SampleMethod();`
- Может быть применено несколько атрибутов одновременно
`[Serializable]`
`[Obsolete]`
`[DefaultMember("Re")]`
`[MyAttrib("im", myval = 12345)]`
`public struct Complex { ... }`
 - Несколько атрибутов можно объединить в одни []
`[XmlIgnore, Obsolete]`
`public double Re { get; set; }`
- Некоторые атрибуты могут быть применены сразу несколько раз
`[Conditional("DEBUG"), Conditional("TEST1")]`
`void TraceMethod(){...}`

Применение атрибутов

- Параметры атрибутов:

- Могут задаваться позиционно или по имени
- Позиционные, как обычно, задаются в определённом порядке и не могут быть пропущены
- Именованные параметры могут следовать в произвольном порядке и могут быть пропущены

```
[DllImport("user32.dll")]  
[DllImport("user32.dll", SetLastError=false, ExactSpelling=false)]  
[DllImport("user32.dll", ExactSpelling=false, SetLastError=false)]
```

.....

- Целью атрибутов могут быть:

- Сборка, Модуль, Тип, Поле, Свойство, Метод, Параметр (метода или свойства), Возвращаемое значение (метода или свойства), Событие

- [assembly: AssemblyTitle("Production assembly 4")] // Атрибут сборки

- [module: CLSCompliant(true)] // Атрибут модуля

- [SomeAttr] // Атрибут метода

- int Method1() { return 0; }

- [method: SomeAttr] // Можно и конкретно указать, что это атрибут метода

- int Method2() { return 0; }

[return: SomeAttr]

// Атрибут возвращаемого значения

Создание собственного атрибута

- Класс, должен быть унаследован от абстрактного класса `Attribute`

```
public class ColumnAttribute : Attribute {...}
```

- Можно задавать область применимости атрибута с помощью атрибута `AttributeUsage`

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct)]
public class TableAttribute : Attribute
{
    public TableAttribute(string tableName)
    {
        this.tableName = tableName ?? String.Empty;
    }

    private readonly string tableName;
    public string TableName { get { return tableName; } }
}
```

- Применение такого атрибута (`Attribute` можно опускать)

```
[Table("Customers")]
public class Customer {...}
```

```
[TableAttribute("Orders")]
public class Order {...}
```

- Класс `Attribute` предоставляет набор статических методов для работы с атрибутами

Получение атрибута

• Через информацию о члене:

- `bool MemberInfo.IsDefined(Type attrType, bool inherit)` – нет доступа к атрибутному объекту
- Только проверка о существовании атрибута

• Через атрибут:

- `Attribute[] GetCustomAttributes(MemberInfo mi)`
- `Attribute Attribute.GetCustomAttribute(MemberInfo mi, Type attrType)`
- Создают экземпляры атрибутов (вызывают конструкторы классов атрибутов, задают свойства и т. д.)

```
if (entity.GetType().IsDefined(typeof(TableAttribute), false))
{
    TableAttribute tableAttribute = (TableAttribute)Attribute.GetCustomAttribute(entityType,
                                                                                typeof(TableAttribute));
    Console.WriteLine(tableAttribute . TableName );
}
```

- Атрибут может наследоваться от предка к потомку. Например, применив атрибут к базовому классу, класс наследник тоже будет иметь этот атрибут. Наследование может быть задано с помощью `AttributeUsageAttribute` на конкретном классе атрибута

```
[AttributeUsageAttribute(AttributeTargets.All, Inherited = true, AllowMultiple = false)]
public abstract class AuthorAttribute : Attribute {...}
```


Использование атрибутов

- Сериализация – что и как сериализовать
- Работа с БД
 - Автоматическое чтение/запись объектов в БД
 - Авто генерация таблиц по атрибутам
- WCF, веб-сервисы
 - Пометить метод как доступный удаленно. Задание контрактов
- Аспектно-ориентированное программирование
- Задание метаданных сборке (версию, описание, торговую марку и т. д.)
[assembly: AssemblyVersion("1.0.0.0")]
- Вызов неуправляемого кода
[DllImport("user32.dll", CharSet = CharSet.Unicode)]
public static extern int MessageBox(IntPtr hWnd, String text, String caption, uint type);
- Взаимодействие со средой разработки
[Obsolete("Use Method2 instead")]
- Описание свойств, методов и т.д. для COM объектов
- Взаимодействие с системой безопасности .NET

Демонстрации

Работа с атрибутами

Построение SQL запроса для произвольной Entity

Сегодня

- Атрибуты
- Сериализация

Сериализация

- Сериализация – процесс сохранения состояния объекта в потоке
- Сохраняются данные и необходимая информация для реконструкции объекта – десериализации
- При сериализации
 - Сохраняется граф объектов
 - Сохраняются данные и всех базовых классов
- Использование
 - Сохранение данных программы (например, настройки)
 - Передача объектов по сети
 - “Персистентные” объекты (время жизни объекта больше времени работы программы)

Формат сериализации

- Двоичный
 - Компактный
 - Для десериализации нужен .NET
- SOAP
 - Стандарт W3C.org
 - Не зависит от платформы
 - Многословен
- XML
 - Произвольный XML формат
 - Не зависит от платформы
 - Многословен
 - Полностью настраиваемый

Настройка сериализации

- Только для двоичного и SOAP форматов
- С помощью атрибутов
- **[Serializable]** – указывает на то, что тип можно сериализовать
[Serializable] class Complex {....}
- Что сериализуется
 - Все публичные и приватные поля (и автоматические свойства)
 - Базовые и используемые типы тоже должны быть сериализуемыми, т.е. иметь атрибут [Serializable]
- **[NonSerialized]** – помечается поле или автоматическое свойство, если что-то сериализовать не нужно
[Serializable] struct Complex {
 double im, re;
 [NonSerialized] double mod, arg; }
}

Как сериализовать

○ Форматеры

- **BinaryFormatter** – для бинарной сериализации
 - (System.Runtime.Serialization.Formatters.Binary.BinaryFormatter)
- **SoapFormatter** – для сериализации в SOAP формате
 - (System.Runtime.Serialization.Formatters.Soap.SoapFormatter)
- **XmlSerializer** – для сериализации в произвольный XML формат
 - (System.Xml.Serialization.XmlSerializer)

Как сериализовать

• Для BinaryFormatter и SoapFormatter

- **Serialize**(stream, obj) – сериализация объекта в поток

```
Complex complex = new Complex(10, 20);
using (FileStream stream = new FileStream("store.bin", FileMode.Create))
{
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    binaryFormatter.Serialize(stream, complex);
}
```

- object **Deserialize**(stream) – для десериализации объекта

```
Complex complex;
using (FileStream stream = new FileStream("store.bin", FileMode.Open))
{
    BinaryFormatter binaryFormatter = new BinaryFormatter();
    complex = (Complex)binaryFormatter.Deserialize(stream);
}
```

• Для XmlSerializer необходимо указать сериализуемый тип(-ы)

- XmlSerializer serializer = new XmlSerializer(typeof(Complex));
- serializer.Serialize(fileStream, complex);
- XmlSerializer serializer = new XmlSerializer(typeof(Car), new Type[] { typeof(Radio) });
- Car c = (Car)ser.Deserialize(fileStream);

Особенности XML сериализации

- Сериализация только публичных полей и свойств (приватных полей, стоящих за ними)
- Должен быть конструктор без параметров
- Атрибуты настройки вида XML
 - [XmlIgnore]
 - [XmlAttribute]
 - [XmlElement]
 - [XmlText]
 - И др.
- Необходимо задавать граф сериализуемых объектов
`XmlSerializer serializer = new XmlSerializer(typeof(Car), new Type[] { typeof(Radio) });`
- Не умеет сериализовать `ArrayList` и `List<T>`

Демонстрации

Сериализация

Сегодня

- Атрибуты
- Сериализация