



5. Обща Система от Типове / Common Type System



Съдържание

- Типът `System.Object`
- Създаване и преобразуване обекти
- Пространства от имена
- Примитивни типове
- Референтни и стойностни типове
- Опаковане и разопаковане
- Равенство и идентичност на обекти
- Сравняване на обекти
- Клониране на обекти



Какво е CTS?

- Дефинира поддържаните от CLR типове данни и операциите над тях
 - Примитивни типове
 - Структури
 - Изброени типове
 - Класове
 - Интерфейси
 - Масиви
 - Делегати
 - Указатели



Йерархия на типовете в CTS





Типът System.Object

- Типът System.Object е базов за всички типове
- При дефиниране на тип по подразбиране се наследява System.Object

```
class MyType { } ⇔ class MyType : System.Object { }
```

- Предоставя базова функционалност за всички типове



Членове на System.Object

- **public Type GetType();**
- **public virtual bool Equals(object);**
- **public virtual int GetHashCode();**
- **public virtual string ToString();**
- **protected virtual void Finalize();**
- **protected object MemberwiseClone();**
- **public static bool ReferenceEquals();**



GetType()

- Връща обект от тип `System.Type`, който представя типа на обекта
- Не може да се предефинира
- Използва се при мета-програмирането



Equals()

- Предназначение – сравняване по стойност
- Реализация по подразбиране – сравнява по:
 - референтни типове по референции
 - стойностни типове по стойност
- Може да се предефинира



GetHashCode()

- Предназначение – за извлечане на хеш кода (уникален идентификатор) на обекта
- Хеш кода е функционално зависим от член-данните (може и само от част от тях)
- Употреба – когато обекти от типа ще се използват като ключ в Хеш-таблици
- Може да се предефинира



ToString()

- По подразбиране връща стринг с пълното име на типа
- Не прави конвертиране към **String**
- Полезно е да връща стринговата интерпретация на обекта
- Може да се предефинира



Finalize()

- Служи за освобождаване на неуправляеми ресурси преди обекта да бъде изтрит
- Не трябва да се вика ръчно – Garbage Collector-а го вика, когато освобождава памет
- Може да се предефинира



Другите методи

- MemberwiseClone()
 - извършва плитко копиране на текущия обект
- ReferenceEquals(**object**, **object**)
 - статичен метод
 - сравнява два обекта по референция



Операторът **new**

- Служи за създаване на обект от даден тип
- Модел на работа
 - Изчислява паметта, която заема един обект като обхожда собствените и наследените член данни
 - Заделя нужната памет
 - Извиква конструктора, за да се иниц. паметта
 - Връща адреса на новия обект
- Няма съответстващ **delete** оператор



Преобразуване на типове / casting

- Типова безопасност – всеки тип знае типа си
- Преобразуване по подразбиране / Implicit cast
 - Използва се при преобразуване до базов тип
- Изрично преобразуване / Explicit cast
 - Използва се при преобразуване до наследен тип
 - Оператор ()
 - Оператори **is** и **as**



Операторът ()

- Прави проверка на типа
- Обхожда цялата йерархия до съвпадение
- Ако няма съвпадение хвърля [InvalidCastException](#)
- Може да се предефинира



Операторът **is**

- Прави проверка на типа
- Обхожда цялата йерархия до съвпадение
- Връща **true** или **false** в зависимост дали има съвпадение
- Не хвърля изключение
- Може да се предефинира



Операторът **as**

- Прави проверка на типа
- Обхожда цялата йерархия до съвпадение
- Ако няма съвпадение връща **null**
- Може да се предефинира



Употреба на (), is и as

- () се използва когато сме сигурни в типа и в противен случай искаме да възникне изключ.
- **is** и **as** се използват когато кода ни зависи от какъв точно е типът

```
Object o;                                Slower  
...  
if (o is MyType)  
{  
    MyType mt = o as MyType;  
    // use mt  
}
```

```
Object o;                                Faster  
...  
MyType mt = o as MyType;  
if (mt != null)  
{  
    //use mt  
}
```



Пространство от имена / Namespace

- Тповете и асемблитата
 - Физически типовете са разположени в асембли
 - Типа се определя уникално от името на асемблито и пълното име на типа
- Типовете и Namespace-ите
 - Логически типовете са организирани в namespace
 - Пълното име на тип се образува от пътя до него в йерархията от namespace-и и собственото му име
- Един namespace може да е в няколко асемблита



Директивите `using`

- Можем да работим с кратките имена на типа

```
public class Program {  
    public static void Main(string[] args) {  
        System.IO.FileStream fs = new System.IO.FileStream(...);  
        System.Text.StringBuilder sb = new System.Text.StringBuilder(...);  
    }  
}
```

```
using System.IO;  
using System.Text;  
  
public class Program {  
    public static void Main(string[] args) {  
        FileStream fs = new FileStream(...);  
        StringBuilder sb = new StringBuilder(...);  
    }  
}
```



Примитивни типове (1/2)

- Типове, които се поддържат директно от компилатора
- Типове, които не са съставени от други типове
- Дефинирани са специални правила за преобразуване



ПРИМИТИВНИ ТИПОВЕ (2/2)

C# Primitive Types (keywords)	CLR Primitive Types
byte	System.Byte
short	System.Int16
int	System.Int32
long	System.Int64
char	System.Char
float	System.Single
double	System.Double
bool	System.Boolean
decimal	System.Decimal
object	System.Object
string	System.String



Референтни типове

- Всеки типо, който е дефиниран с думата **class**
- **Копират се по референция при присвояване**
- Могат да приемат стойност **null**
- Може няколко променливи от референтен тип да сочат към един и същи обект в паметта
- Потребителски референтни типове – класове, интерфейси, масиви, делегати, указатели



Стойностни типове

- Наследяват абстрактния тип `System.ValueType`
- **Копират се по стойност при присвояване**
- Не могат да се наследяват (те са `sealed`)
- Не могат да приемат стойност `null`
- Оптимизирани от CLR-а
- Потребителски стойностни типове
 - Структури
 - Изброени типове
 - `Nullable` типове



Оптимизация на стойностните типове

- Динамичната памет е скъп ресурс
 - Сложен механизъм за освобождаване на паметта
 - Голям капацитет
- Статичната памет е бърза
 - Прост механизъм за освобождаване на паметта
 - Малък капацитет
- Когато е възможно обектите от стойностен тип се пазят в статичната памет



Структури

- Дефинират се със запазената дума **struct**
- Могат да имат полета, свойства и методи
- Могат да имплементират интерфейси
- Необходими условия, за да деф. структура
 - Няма методи, които променят член данните
 - Не трябва да наследява друг тип
 - Няма да бъде наследяван от други типове
 - Обектите са ще са малки (до 16 байта)



Изброени типове

- Дефинират се със запазената дума **enum**
- Наследяват типа **System.Enum**
- Дефинират статични, константи полета
- Не могат да имат свойства и методи
- Не могат да имплементират интерфейси



Nullable типове

- Надграждат примитивните стойностни типове като позволяват да имат стойност **null**
- Имат read-only свойство `HasValue`, което показва имат ли стойност
- Имат read-only свойство `Value`, което дава стойността, когато я има или хвърля изключение `InvalidOperationException`
- Има метод `GetValueOrDefault()` за улеснено ползване



Опаковане и разопаковане на стойностните типове

- Употреба на стойностен тип като референтен
- CLR-ът поддържа опаковане и разопаковане на стойностните типове
- Спестява дефинирането на wrapper класове за стойностните типове



Опаковане / Boxing

- Опаковането е действие, което преобразува стойностен тип в референтен
- Процедура на опаковане:
 - Заделя се динамична памет за създаване на обект
 - Копира се съдържанието на променливата от стека в заделената динамична памет
 - Връща се референция към създадения обект в динамичната памет



Разопаковане / Unboxing

- Разопаковането е действие, което преобразува опакован стойностен тип в обикновен стойностен тип
- Процедура на разопаковане:
 - Ако референцията е **null** се предизвиква **NullReferenceException**
 - Ако референцията сочи към обект от различен тип, се предизвиква **InvalidCastException**
 - Стойността се извлича от динамичната памет



Равенство и идентичност на обекти

- Неконсистентност в .NET Framework
- Методът Equals()
 - Предназначение – за проверка на равенство
 - Базова имплементация:
 - Стойности типове – равенство
 - Референтни типове – идентичност
 - Базовата имплементация на == ползва Equals()
- Методът `Object.ReferenceEquals()`
 - Предназначение и базова имплементация – идентичност



Предефиниране на Equals()

- Рефлексивност, симетричност и транзитивност
- Предефиниране и на GetHashCode()
- Предефиниране и на == и !=
- Силно препоръчително за структури



Интерфейсът `IEquatable<T>`

- Имплементиране на равенство на обекти
- Дефинира типово безопасен метод `Equals()`
- Предефиниране на `Equals()` да вика типово безопасния метод `Equals()`
- Предефиниране на операторите `==` и `!=` да викат типово безопасния метод `Equals()`



Интерфейсът IComparable

- Имплементиране на сравняване на обекти
- Дефинира метода CompareTo() който връща:
 - число < 0, ако подаденият обект е по-голям от текущия обект
 - 0, ако подаденият обект е равен на текущия обект
 - число > 0, ако подаденият обект е по-малък от текущия обект



Интерфейсът ICloneable

- Имплементиране на същинско клониране на обекти
- Дефинира метода `Clone()`, който връща идентично копие на обекта



Въпроси?



Ресурси

- “Програмиране за .NET Framework”, Светлин Наков и колектив, Глава 5: Обща система от типове
- “CLR via C#”, Jeffrey Richter, Chapter 4: Type fundamentals, Chapter 5: Primitive, Reference and Value Types
- Erik Lippert blog post:
 - [The Stack Is An Implementation Detail, Part One](#)
 - [The Stack Is An Implementation Detail, Part Two](#)
 - [The Truth About Value Types](#)