VB.NET - Box e Unboxing

Boxing e Unboxing é um conceito essencial no sistema de tipos do VB.NET (C# também). Mas o que vem a ser isto de boxing e unboxing ?

Nota: Podemos traduzir boxing como empacotar e unboxing como desempacotar mas creio que o melhor mesmo é nos referirmos aos termos originais box e unboxing pois será assim que serão referenciados na literatura e em artigos.

Para você entender estes conceitos vou dar uma pincelada no sistema de tipos usado no .NET.

Como você já deve saber o VB.NET agora é uma linguagem orientada a objetos e podemos dizer que tudo no VB.NET são objetos. Seguindo esta linha de raciocínio podemos dizer também que em .NET todo o tipo é derivado da classe System.Object.

O fato de tudo ser um objeto traz alguns problemas que podem afetar o desempenho. Vamos pensar na operação matemática da soma de dois números , 1 +2 , por exemplo.

Para realizar esta operação , se tudo for considerado um objeto , teremos que criar um objeto para representar o número 1 e outro objeto para representar o número 2 , somar os dois e colocar o resultado em outro objeto.

Um tanto complicado não é mesmo ???

A linguagem Java contornou este programa definindo os tipos por referência que são objetos e os tipos primitivos (int, char,boolean byte, etc..) que não são objetos e que guardam apenas os valores de seus dados.

Como não são objetos os tipos primitivos não possuem métodos . Isto levou a criação das classes invólucros(wrappers) destes tipos primitivos que transformam os tipos primitivos em tipos por referência.

Só tem um problema em tudo isto : quem tem que fazer este serviço é você , o programador.

E como o problema foi contornado na plataforma .NET ?

Se você respondeu box e unboxing , acertou !

Para resolver este problema foi organizado o sistema de tipos de duas formas:

- **Tipos Valor**: variáveis deste tipo são alocadas na pilha e têm como classe base System.ValueType, que por sua vez deriva de System.Object.
- **Tipos Referência**: variáveis deste tipo são alocadas na memória **heap** e têm a classe System.Object como classe base.

Organizando o sistema de tipos desta forma , apenas os tipos referência seriam alocados na memória heap, enquanto os tipos valor iriam para a pilha. Tipos primitivos como **int, float e char** não precisam ser alocados na memória heap, agilizando, assim, a sua manipulação.

Mas onde entra o tal de box e unboxing ?

A razão de se criar uma origem comum de tipos é para facilitar a interação entre tipos valor e referência.

O processo de conversão explícita de um tipo valor para um tipo referência é conhecido em .NET como **Boxing** (empacotar).

O processo contrário a **Boxing** é conhecido como **Unboxing**. Nesse caso, o compilador verifica se o tipo por valor a receber o conteúdo do tipo referência é equivalente a este último.

No processo de **Boxing**, o que de fato está acontecendo é que um novo objeto está sendo alocado na memória **heap** e o conteúdo da variável de tipo valor é copiado para a área de memória referenciada por esse objeto.

Exemplos:

```
Class Teste

    Shared Sub Main()
     Console.WriteLine(5.ToString())
End Sub

End Class
```

Perceba que neste exemplo eu estou chamando o método **ToString**() para um valor inteiro.
O que esta acontecendo por trás dos panos ???

Abaixo temos o exemplo mais claro de boxing e unboxing.

```
Class Teste2
```

```
Public Class Teste3
{
```

```
Shared Sub Main()
  Dim i As Integer = 1
  Dim o As Object = i          'aqui estou fazendo o boxing
  Dim j As Integer = Cint(o)   'aqui estou fazendo o unboxing
End Sub

End Class
```

Exemplo em VB.NET

```
 Static void Main()
 {
      int intNumero = 100;

      // Faz o boxing para o tipo referencia.
      Object objNumero = intNumero;

     // Faz o unboxing para o tipo valor
      int intValor = (int) objNumero;
  }
}
```

Exemplo em C#

A nova versão do Java (Tiger) traz como uma das novidades o box e unboxing.

Até mais ... 

José Carlos Macoratti