Artigo

Struct e Class: Quando usar?

Por: Renato Guimarães
[Entre em contato com o autor]

Bacharel em Sistemas de Informação e trabalha com tecnologia da informação há mais de 15 anos.

Struct e Class: Quando usar?

Publicado em: 12/02/2007

Há dias tento reservar um tempinho para falar sobre Struct e Class. Lembro que uma pessoa fez esta pergunta para mim no MSN e achei interessante explicar este assunto em um documento para que sirva de referência para outros colegas da comunidade. Uma decisão errada pode impactar a performance da sua aplicação ou, por outro lado, afetar a manutenibilidade do código. Creio que você, em algum momento, já deve ter se perguntado quando usar um Struc ou quando usar um Class.

Para isso, é preciso uma explicação bem simples sobre as duas principais áreas de memória e, também, o que são os tipos por valor (Value Types) e tipos por referência (Reference Types) bem como eles são tratados pelo CRL (Common Language Runtime), principalmente no que diz ao gerenciamento de memória. Em seguida, uma visão geral sobre o Struc e Class, com foco nas características de cada tipo.

Stack e Heap

Cada thread tem uma área de memória reservada que é chamada de stack. Quando um método é executado ele pode adicionar os dados que necessita ao topo da stack (ou pilha), assim que concluir sua execução os dados armazenados na stack são removidos, de forma automática e livrando programador de qualquer preocupação. Esta área de memória tem um tamanho reduzido (alguns kilobytes), o que pode provocar problemas ou o tão conhecido "stack overflow". Outro ponto imporante é que os dados desta memória são removidos quando a função retorna, o que pode gerar várias cópias caso alguma outra função espera pelo valor de retorno. Resumindo, a stack é uma memória de tamanho limitado e a duração da alocação dependem da duração do método correspondente.

Por outro lado, a Heap é formado por uma grande área de memória que no primeiro momento está sem uso, cujo tamanho pode será determinado dinamicamente em tempo de execução. A duração da alocação não depende da duração de um método ou da stack. Esta área de memória é acessada indiretamente, ou seja, através de uma referência (o que a seguir será tratado como Reference Type). No Framework .NET esta área de memória é controlada pelo Garbage Collector (GC), que é responsável por sua limpeza e organização. Basicamente, o GC verifica se ainda existe alguma variável referenciando um determinado objeto e, quando não houver mais, pode-se dizer que a qualquer momento ele será desalocado da memória, ou melhor, coletado.

Value Types e Reference Types

Como já foi explicado, resumidamente podemos dizer que existe duas áreas de memórias onde os dados (objetos) de uma aplicação são armazenados: *Heap* e *Stack*. A principal diferença entre eles? Custo para criar os objetos na memória, pois a Heap tem um custo bem maior que a Stack. Pode-se resumir que os Value Types são

armazenados na Stack e os Reference Types na Heap. Entre os Value Types temos: tipos numéricos (int, long, short etc), ponto flutuante (float, double), decimal, booleanos (true e false) e estruturas definidas pelo usuário (struct). Já entre os Reference Types, temos: classes (class), interfaces (interface) e delegates (delegate).

Abaixo temos as principais características dos Value Types:

- Uma variável deste tipo contém o valor, e não um endereço de referência para o valor;
- Derivam de System.ValueTypes;
- Não é possível criar uma classe que herde deste tipo;
- Pode implementar interfaces da mesma forma que um Reference Type;
- Não pode ser atribuído o valor null, a não ser que se use nullable types (que é uma característica do .NET que permite que Value Types recebam o valor null, novidade do .NET Framework 2.0);
- Não pode conter o construtor default, ou seja, o construtor que não tem parâmetros. Mas fique tranquilo pois este construtor é criado implicitamente para que os membros da Struct sejam inicializados para seus valores padrão;
- Variáveis de escopo local precisam ser inicializadas antes de serem utilizadas;
- Atribuir o valor de variável a outra, implicitamente, é feita uma cópia do conteúdo da variável. Sendo assim, qualquer alteração no conteúdo de uma delas, não afetará a outra. Quanto maior for um objeto deste tipo mais custosa será sua cópia.

Abaixo temos as principais características dos Reference Types:

- Uma variável contém a referência(ou endereço) para o objeto que está na Heap;
- Atribuir o valor de uma variável para outra faz uma cópia da referência, e não do próprio objeto. Ou seja, não é feita a cópia do objeto, e sim do endereço de memória do objeto, o que não gera muito custo para objetos grandes;
- São alocados na Heap e seus objetos são coletados pelo Garbage Collector;
- São passados por referência, enquanto que Value Types são passados por valor. Ou seja, a alteração de um objeto afetará todas as instâncias que apontam para ele.

Para exemplificar a utilização de uma Struct, a **listagem 1** mostra a particularidade dos Value Types quando são passados como parâmetros. Perceba que neste exemplo a estrutura tem um construtor com parâmetros, lembre-se que não pode ter construtor sem parâmetros. Além disso, ilustra como é possível passar um Value Type por referência.

Listagem 1 – Exemplificação de um Value Type e seus detalhes quando passados como parâmetros

```
using System;

/// <summary>
/// Declaração da estrutura através da palavra-chave struct
/// </summary>
public struct Posicao
{
    public int X, Y;

    /// <summary>
    /// Declaração do construtor com parâmetros, ou seja, não o Default
    /// </summary>
    public Posicao(int px, int py)
    {
        X = px;
        Y = py;
    }
}

class Program
{
    static void Main(string[] args)
    {
        //Declaração de escopo local com duração até o método concluir.
        //Ainda é preciso inicializá-las, caso contrário ocorrerá erro.
        //A declaração e inicialização pode ser feita na mesma linha.
        Posicao pos1, pos2, pos3;

        // Inicialização com construtor padrão e com constutor
específico.
        //Desta forma os membros já foram inializados. Para inicializar
com
        //construtor padrão: pos1 = new Posicao()
        pos1 = new Posicao(30, 40);

        //Como a variável pos2 ainda não foi inicializada através da
execução
        //do construtor padrão ou um construtor específico. Sendo assim,
é
        //preciso inicalizar a variável antes de utilizá-la.
```

```
        pos2.X = 70;
        pos2.Y = 80;

        //No trecho abaixo é feita uma cópia da variável pos1 para pos2.
Com isso,
        //qualquer alteração nos valores de uma delas, não afetará a
outra.
        pos2 = pos1;
        pos2.X = 20;
        pos2.Y = 20;

        //Novamente é feita outra cópia. Se a Struct for grande e
complexa perceba
        //que haverá um consumo excessivo de memória devido as várias
cópias. Sendo assim,
        //tenha cuidado com o tamanho do Struct.
        pos3 = pos2;
        pos3.X = 50;
        pos3.Y = 50;

        //Quando passados por parâmetro, também é feita uma cópia, a não
ser que
        // você especifique que é por referência.O valor de pos3 não será
alterado
        Dobro(pos1);
        Console.WriteLine("Posição 1: x = {0}, y = {1}", pos1.X, pos1.Y);

        //Para que um ValueType seja utilizado como um ReferenceType,
faz-se
        //necessário o uso da palavra-chave ref, ou seja, por referência.
        //Neste caso não será feita uma cópia do valor da variável.
        //Tanto a declaração do método quanto sua chamada
        //devem ser feitas com a palavra-chave ref. OBS: Não esquece de
inicializar a
        //variável antes de passá-la como ref.
        Triplo(ref pos1);

        Console.WriteLine("Posição 1: x = {0}, y = {1}", pos1.X, pos1.Y);
        Console.WriteLine("Posição 2: x = {0}, y = {1}", pos2.X, pos2.Y);
        Console.WriteLine("Posição 3: x = {0}, y = {1}", pos3.X, pos3.Y);
        Console.ReadLine();
    }

    public static void Dobro(Posicao posicao)
    {
        posicao.X = posicao.X * 2;
        posicao.Y = posicao.Y * 2;
    }

    public static void Triplo(ref Posicao posicao)
    {
        posicao.X = posicao.X * 3;
        posicao.Y = posicao.Y * 3;
    }
}
```

A **listagem 2** exemplifica a utilização de um Reference Type bem como sua particularidade quando passado como parâmetro de um método.

Listagem 2 – Exemplificação de um Value Type e seus detalhes quando passados como parâmetros

```
using System;

/// <summary>
/// Declaração de uma classe simples com construtor padrão, construtor
com
/// parâmetros, propriedades etc.
/// </summary>
public class Cliente {

    private string _matricula;
    private string _nome;

    public Cliente() { }

    public Cliente(string matricula, string nome) {
        this.Matricula = matricula;
        this.Nome = nome;
    }

    public string Matricula{
        get { return _matricula; }
        set { _matricula = value; }
    }
```

```
    public string Nome{
        get { return _nome; }
        set { _nome = value; }
    }
}
class Program{

    static void Main(string[] args){
        Cliente cli1, cli2;

        //Inicialização de uma instância da classe Cliente.
        //Com isso, foi criado um objeto na memória Heap e a variável
cli1 contém
        //o endereço de memória deste objeto, ou seja, a referência. Como
já foi comentado,
        //não acessamos a Heap diretamente.
        cli1 = new Cliente();
        cli1.Matricula = "1345-9";
        cli1.Nome = "Renato Guimarães";

        //Como cli2 ainda não foi inicializado, seu valor é null. Ao
tentar
        //utilizá-la ocorrerá uma exceção do tipo NullReferenceException
        //No código abaixo a váriavel cli2 recebe a mesma referÊncia de
cli1.
        //Agora as duas variáveis apontam para o mesmo objeto. Qualquer
alteração que
        //for realizada em umas das variáveis afetará a outra. Para o GC,
agora existem
        //duas referências apontando para o objeto, desta forma não
poderá ser removido
        //da memória.
        //Para remover as referências, pode-se atribuir o valor null as
duas variáveis ou
        //ao final da execução do método. As referências estão na Stack e
os objetos estão na Heap.
        cli2 = cli1;

        //OBS: Quando passamos um ReferenceType como parâmetro, na
verdade ainda é
        //feita uam cópia do valor da variável, só que não do objeto
todo. Até porque
        //a variável só contém o endereço de memória do objeto.
        MostrarObjeto(cli1);
        MostrarObjeto(cli2);

        //A alteração aferatá as duas variáveis, pois apontam para mesmo
objeto.
        Maiuscula(cli2);
        MostrarObjeto(cli1);
        MostrarObjeto(cli2);

        Console.ReadLine();
    }

    //Como é um Reference Type, qualquer alteração no parâmetro afetará
todas
    //as variáveis que apontam para esta referência.
    public static void Maiuscula(Cliente cli) {
        cli.Nome = cli.Nome.ToUpper();
    }

    public static void MostrarObjeto(Cliente cli) {
        Console.WriteLine("Matrícula: {0}", cli.Matricula);
        Console.WriteLine("Nome.....: {0}", cli.Nome);
    }
   }
```

Conclusão

Use um Value Type quando você tiver certeza de que as instâncias daquele tipo serão pequenas e terão uma duração curta ou que serão somente utilizadas embutidas em outros objetos. Um Value Type deve ser definido para tipos cujo tamanho seja de 16 bytes ou menos. Nada impede que sejam maiores que 16 bytes, mas se o for, que não sejam utilizados como parâmetros de métodos ou usados em coleções (lembre-se do *boxing* e *unboxing*, outro dia eu explico o que é isso). Ideal que você só utilize estes tipos dentro de métodos e que os utilize com cuidado como retorno do método (por exemplo, um Struct muito grande pode provocar problemas de performance). Se precisar utilizar um Value Type com coleções, por favor, não deixe de utilizar o recurso de Generics (que resolve o problema de boxing e unboxing), novidade do .NET Framwork 2.0.

Como os Value Types são limitados no que diz respeito a herança, com certeza, já podem ser descartados caso você esteja montando uma hierarquia de objetos. Caso seja um objeto complexo no qual passado entre camadas, manipulado através de coleções e associados a controles, não perca tempo e o declare como uma classe.