theburningmonk.com
serverless, AWS, functional programming, cloud

.Net Tips – making a serializable immutable struct

2 Comments / .Net, C#, Programming, Tips / April 25, 2010

Presently sponsored by Blue Matador: You've automated your infrastructure with serverless. Now automate your monitoring alerts with Blue Matador. Stop configuring alerts for your Lambda functions, just use Blue Matador. We're so confident you'll love it that we're offering $100 to demo it with our team.

As you might know already, an object is **immutable** if its state doesn't change once it has been created.

In C# the most used immutable type is *string*, this means every time you modify the value of a string variable you are actually creating a new string object and updating the reference of the variable to point to the new object.

Class vs Struct

When creating a new type, you have the choice of either a *class* or a struct. The general rule of thumb is to go with a class except for lightweight types smaller than 16 bytes in which case it is more efficient to use a struct. The reason a struct can be more efficient is because a *struct* is a value type and therefore goes straight onto the stack so we don't have the overhead of having to hold the reference to the object itself (4 bytes in a 32bit system).

Mutable vs Immutable

In addition, you also have to consider whether your type should be mutable or immutable. In general, a *struct* should always be immutable because a *struct* usually represents some fundamental value – such as the number 5 – and whilst you can change a variable's value you don't logically change the value itself.

Also, data loss is far too easy with mutable *structs*, consider the following:

```
Foo foo = new Foo(); // a mutable struct
foo.Bar = 27;
Foo foo2 = foo;
foo2.Bar = 55;
```

Now *foo.Bar* and *foo2.Bar* is different, which is often unexpected.

Here are some [illegible]

- **Easier validation** – if you validate the parameters used to construct your object, your object will never be invalid as its state can never be changed.
- **Thread safety** – immutable types are inherently thread-safe because there is no chance for different threads to see inconsistent views of the same data if the data can never be changed.
- **Better encapsulation** – immutable types can be exported from your objects safely because the caller cannot modify the internal state of your objects.
- **Better for hash-based collections** – the value returned by *Object.GetHashCode()* must be an instance invariant, which is always true for immutable types.

Deserializing an Immutable Struct

To create an immutable *struct*, you usually have no setters on properties and in all likelihood the private variables that the getters return will be made readonly too to enforce the write-once rule. The lack of public setters on properties, however, represents a challenge when serializing/deserializing the immutable *structs*.

The easiest way to get around this in my experience is to simply implement the ISerializable interface and providing a constructor which takes a SerializationInfo and a StreamingContext object:

```
[Serializable]
public struct MyStruct: ISerializable
{
    private readonly int _x;
    private readonly int _y;

    // normal constructor
    public MyStruct(int x, int y) : this()
    {
        _x = x;
        _y = y;
    }

    // this constructor is used for deserialization
    public MyStruct(SerializationInfo info, StreamingContext text) : this()
    {
        _x = info.GetInt32("X");
        _y = info.GetInt32("Y");
    }

    public int X { get { return _x; } }
    public int Y { get { return _y; } }

    // this method is called during serialization
    [SecurityPermission(SecurityAction.Demand, SerializationFormatter = true)]
    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        info.AddValue("X", X);
        info.AddValue("Z", Y);
    }
}
```

Reference:

StackOverflow thread on immutability of structs

Patrick Smacchia's article on Immutable Types: understand their benefits and use them