

Remoting Prerequisites

Tomáš Matoušek
&
Ladislav Prošek

tmd.havit.cz/teaching/csharp.htm

Process

- “a running program”
- fundamental property: virtual address space
 - provides isolation
- represented by `System.Diagnostics.Process` class
 - just a wrapper around process-related Win32 API

```
foreach (Process process in Process.GetProcesses())  
{  
    Console.WriteLine(process.ProcessName);  
    if (process.VirtualMemorySize > 1000000000) process.Kill();  
}
```

```
Process.Start("cmd.exe");
```

Application Domains

- “light-weight” processes
- represented by `System.AppDomain` class
- provide isolation between applications running in the same process
 - cheaper than processes
 - enabled by verifiability of managed code
- architectural elements responsible for loading assemblies
 - it is impossible to unload individual assemblies or types
 - `AppDomain` is the smallest unloadable entity in .NET
- static fields are allocated per-`AppDomain`, not per-process

Special AppDomains

- three special well-known AppDomains in every process
 1. system domain
 - bootloader for system types
 - loads and maintains a single assembly – mscorlib
 - hidden, cannot be unloaded
 2. shared domain
 - loads domain-neutral assemblies (see the next slide)
 - hidden, cannot be unloaded
 3. default domain
 - loads all assemblies that are not system or domain-neutral (unless another AppDomain is explicitly created)
 - cannot be unloaded

Domain-neutrality

- assembly can be loaded as domain-neutral
 - assembly's code is then shared by all domains that reference it
 - advantage: lower memory consumption (JITted code and EE data structures are shared)
 - disadvantage: slower access to static data (JITted code must contain additional logic to access the appropriate copy of static data)
- three options for domain-neutral assemblies
 1. SingleDomain: only mscorlib is loaded as domain-neutral (default)
 2. MultiDomain: all assemblies are loaded as domain-neutral
 3. MultiDomainHost: all strong-named assemblies are loaded as domain-neutral (ASP.NET)
 - Framework 2.0: all assemblies that are in GAC and whose transitive binding closure is in GAC are loaded as domain-neutral
 - The option is selected when loading a CLR host into a process or by decorating the Main method with `[LoaderOptimization(...)]`.

Instance-agility

- AppDomains within one process share one common managed heap and garbage collector
- isolation enforced by the fact that instances are created, live and die in a single AppDomain
 - ordinary (non-agile) instances cannot be referenced by more AppDomains
 - only so called agile instances can freely cross AppDomain boundary
 - strings, security objects, localization tables, components that are part of the remoting infrastructure, ...
 - must be of types that were loaded as domain-neutral
 - must not hold references to non-agile instances

Threads

- preemptively scheduled execution unit within an address space
- represented by `System.Threading.Thread` class
- What is the relationship between threads and application domains?
 - particular thread is not confined to a single application domain
 - however, at any given time, every thread is executing in one application domain
 - setting up a new `AppDomain` does not create any threads
- threads provide a “local store” for thread-specific data
 - similar to Win32 TLS (Thread Local Storage)
 - achieve the same effect as with the `[ThreadStatic]` attribute
- `System.Threading.ThreadPool` is an easy-to-use pool of worker threads
 - just pass a delegate to a method to `ThreadPool.QueueUserWorkItem()` and the method will eventually run in a background worker thread

Synchronization

- [MethodImpl(MethodImplOptions.Synchronized)]
 - puts the entire method body into `lock(this)` or `lock(typeof(...))`

- System.Threading.Monitor

`lock(obj) { /* critical section */ }` is equivalent to

```
Monitor.Enter(x);  
try  
{  
    /* critical section */  
}  
finally  
{  
    Monitor.Exit(obj);  
}
```

- System.Threading.Mutex
 - can be given a name to support interprocess synchronization
- AutoResetEvent, ManualResetEvent, ReaderWriterLock
 - all in System.Threading
- System.Threading.Interlocked
 - Increment, Decrement, Exchange, CompareExchange atomic operations

Stack

- thread's stack contains much more than just method invocation state
- stack is annotated with execution engine frames
 - data annotations – for example **GCFRAME** whose purpose is to protect references on the stack from being garbage collected
 - transition markers – for example **PInvokeCalliFrame** whose purpose is to mark transition into native code via P/Invoke
- EE must have a knowledge of what is on the stack in order to
 - track and update object references during garbage collection
 - find the correct handler and unwind the stack during an exception
 - generate human-readable call traces for debugger and exception support
 - enforce code access security
 - ...
- stack can be walked from managed code using `System.Diagnostics.StackTrace`
 - only the calling thread's stack or a suspended thread's stack can be walked

Serialization

- serialization – process of converting an object (or object graph) into a contiguous stream of bytes
- deserialization – process of converting a contiguous stream of bytes back into the object graph
- What is it good for?
 - persistence
 - objects can be saved to a file or to a database and restored later
 - ASP.NET session state
 - remoting
 - cross-AppDomain, cross-process, cross-machine transfer over network, shared memory, ...
 - parameters of a remote call
- components that performs the conversion are called formatters
 - .NET Framework (BCL) comes with two formatters
 - `System.Runtime.Serialization.Formatters.Binary.BinaryFormatter`
 - `System.Runtime.Serialization.Formatters.Soap.SoapFormatter`

Default Serialization

- all types whose instances constitute the object graph must be marked with [Serializable] attribute
 - subtypes do not inherit base class's [Serializable] attribute
- all fields are serialized except for those marked with [NonSerialized] attribute
 - [NonSerialized] fields are deserialized as null (reference types) or as "zero value" (value types)

```
[Serializable]
class A
{
    public object X;
    public object Y;

    [NonSerialized]
    public object Z;
}

A a = new A();
// ...
new BinaryFormatter().Serialize(stream, a);
```

Custom Serialization

- if a class wishes to control the serialization process, it implements `ISerializable` interface (in addition to being `[Serializable]`)
 - `ISerializable` contains one method: `GetObjectData` but also implies a “deserializing” constructor with `(SerializationInfo, StreamingContext)` signature
- `ISerializable.GetObjectData`
 - called by the formatter in order to obtain data to be serialized
- deserializing constructor
 - used by the formatter when deserializing an instance
- custom serialization of a class may be delegated to another class
 - useful when it is impossible to fix the original class (part of a library, etc.)
 - the surrogate class provides its services via `ISerializationSurrogate` interface containing two methods: `GetObjectData` and `SetObjectData`
 - must be registered with a `SurrogateSelector` which in turn must be known to the formatter

Advanced Serialization

- **IDeserializationCallback**
 - contains one method: `OnDeserialization`, which is called by the formatter after the entire object graph has been deserialized
 - useful when it is necessary to access fields during deserialization
 - example: `Hashtable`
- **IObjectReference**
 - contains one method: `GetRealObject`, which is called by the formatter during deserialization in order to obtain the “real object” that should be deserialized
 - useful when deserializing singletons
- **SerializationBinder**
 - when registered with the formatter, redirects types of objects being deserialized
 - useful when types have been moved between assemblies, assemblies have been upgraded, etc.
- Web services use `System.Xml.Serialization.XmlSerializer` class to convert object to/from XML
 - different approach than the aforementioned serialization infrastructure