Advanced DOTNet Tutorial

Published On: July 29, 2024

Advanced .Net Tutorial

Gain expertise to advanced .Net programming concepts such as parallel programming, asynchronous programming models, threading, memory management, and native interoperability, along with ADO.net, MVC3, Web Apps with ASP.Net, networking, remoting, and LINQ in this advanced .Net tutorial.

Advanced DOTNet Tutorial PDF

Introduction to Advanced .Net

Advanced DotNet programming is future-proof for developers, as it enables them to create more sophisticated web applications efficiently. Secure your future with these advanced .Net skills, as we cover the following concepts in this advanced .net tutorial:

- Asynchronous Programming
- Threading
- Parallel Programming
- Native Interoperability
- Memory Management
- MVC3
- ADO.Net
- LINQ

Benefits of Upgrading with Advanced .Net Skills

- 32 programming languages make up the .NET programming framework, which is extensively utilized in many different industries.
- The $197 billion gaming industry in 2022 will primarily rely on .NET programming languages like C#.
- Additionally, web, mobile, and iOS app development all heavily rely on .NET programming.
- Web developers and digital designers that work with.NET programming often make a median pay of around $78,000 annually, according to the Bureau of Labor Statistics.
- Top companies like Google, Netflix, and YouTube use.NET programming, which is widely employed in the game sector.

Advanced DOTNet Interview Questions

Understanding of Asynchronous Programming

Three patterns are available in advanced .NET for carrying out asynchronous operations:

- Task-based Asynchronous Pattern
- Event-based Asynchronous Pattern
- Asynchronous Programming Model

Task-based Asynchronous Pattern (TAP): TAP represents the start and finish of an asynchronous action using a single method.

- The .NET Framework 4 brought TAP.
- It's the method for asynchronous programming in .NET.
- The async and await keywords in C# and the async and await operators in Visual Basic give language support for TAP.

Event-based Asynchronous Pattern: The event-

ASP DOTNET Tutorial
Published On: July 31, 2024

ASP DOTNET Tutorial Microsoft created the web framework known as ASP.NET. It is employed in...

Artificial Intelligence Tutorial
Published On: July 30, 2024

Artificial Intelligence Tutorial Artificial intelligence (AI) is significant since it enhances many facets of society...

Appium Testing Tutorial
Published On: July 30, 2024

Appium Testing Tutorial Designed to make the UI automation of many app platforms easier, Appium...

based historical approach for asynchronous behavior is called Event-based Asynchronous Pattern (EAP).

- It needs one or more events, event handler delegate types, and EventArg-derived types, together with a function that ends in -sync.
- With the release of the .NET Framework 2.0 came EAP.
- It is not advised for use in any new developments.

Asynchronous Programming Model (APM): The IAsyncResult pattern, commonly known as the Asynchronous Programming Model (APM) pattern, is a heritage model that leverages the IAsyncResult interface to enable asynchronous functionality.

- Begin and End methods are needed for asynchronous operations in this design.
- **Example:** BeginWrite and EndWrite to implement an asynchronous write operation.
- It is no longer advised to use this pattern for new developments.

To quickly compare the three patterns' representations of asynchronous activities, have a look at the Read method, which reads a given amount of data into a buffer beginning at a given offset.

> *public class MyClass*
>
> *{*
>
> *public int Read(byte [] buffer, int offset, int count);*
>
> *}*
>
> This method's TAP equivalent would reveal the single ReadAsync method shown below:
>
> *public class MyClass*
>
> *{*
>
> *public Task<int> ReadAsync(byte [] buffer, int offset, int count);*
>
> *}*

> The EAP equivalent would reveal the subsequent group of members and types:
>
> ```
> public class MyClass
> {
>    public void ReadAsync(byte [] buffer, int offset, int count);
>    public event ReadCompletedEventHandler ReadCompleted;
> }
> ```
>
> Their APM counterpart would make visible the BeginRead and EndRead methods.
>
> ```
> public class MyClass
> {
>    public IAsyncResult BeginRead(
>       byte [] buffer, int offset, int count,
>       AsyncCallback callback, object state);
>    public int EndRead(IAsyncResult asyncResult);
> }
> ```

Advanced DOTNet Syllabus PDF

Understanding of Threads and Threading

Multithreading improves the responsiveness of your application and, if it runs on a multiprocessor or multi-core system, the throughput.

Processes and Threads

- A process is a running program. An operating system employs processes to separate the programs that are being run.
- A thread is the basic unit through which an operating system allocates processor time.

When to Use Multiple Threads

You employ numerous threads to improve the responsiveness of your program and to make use of a multiprocessor or multi-core system to increase the application's throughput.

- Consider a desktop program in which the primary thread manages user interface components and responds to user interactions. Use worker threads to conduct time-consuming tasks that might otherwise dominate the principal thread and render the user interface unresponsive.
- If your program performs tasks that may be done in parallel, you can reduce the total execution time by doing them in separate threads and running it on a multiprocessor or multicore system. Multithreading on such a system may boost both throughput and responsiveness.

How to Use Multithreading in .NET

Starting with.NET Framework 4, the Task Parallel Library (TPL) and Parallel LINQ (PLINQ) are the preferred methods for implementing multithreading.

- Both TPL and PLINQ rely on ThreadPool threads. The *System.Threading.ThreadPool* gives a .NET application a pool of worker threads. You can also use thread pool threads.
- You may use the System.Threading.Thread to represent a controlled thread.
- Multiple threads may require access to a common resource. To retain the resource in an uncorrupted state and avoid race situations, synchronize thread access to it.
- You may also want to coordinate the interaction of several threads. .NET supports a variety of types for synchronizing access to a shared resource or coordinating thread interaction.

Exceptions in Managed Thread

Some unhandled exceptions that are utilized to regulate program flow have a backstop provided using the common language runtime:

- Because Abort was called, a

ThreadAbortException is raised in that thread. This is specific to applications built with the.NET Framework.

- A thread throws an AppDomainUnloadedException when it detects that the application domain it is running in needs to be unloaded.
- The host process or the common language runtime raises an internal exception to terminate the thread.

Using Threads and Threading

Because processor-intensive processes run on distinct threads while the user interface remains active, multithreaded applications respond to user input more quickly.

How to create and start a new thread

By making a new instance of the *System.Threading.Thread* class, you can start a new thread. The constructor receives the name of the method you wish to run on the new thread. Invoke the *Thread.Start* function to begin a newly generated thread.

How to stop a thread

Use the System.Threading.CancellationToken to put an end to a thread's execution. It offers a standardized method for collaboratively stopping threads.

The .NET Framework provides the Thread.Abort method for forcing the termination of a thread's execution. When that method is called on a thread, a ThreadAbortException is raised on that thread.

How to pause or interrupt a thread

The Thread.Interrupt method allows you to set a time limit for the current thread's pause. By using Thread.Interrupt method, you can break the block on a blocked thread.

Thread Properties

Some of the Thread properties are shown in the following table:

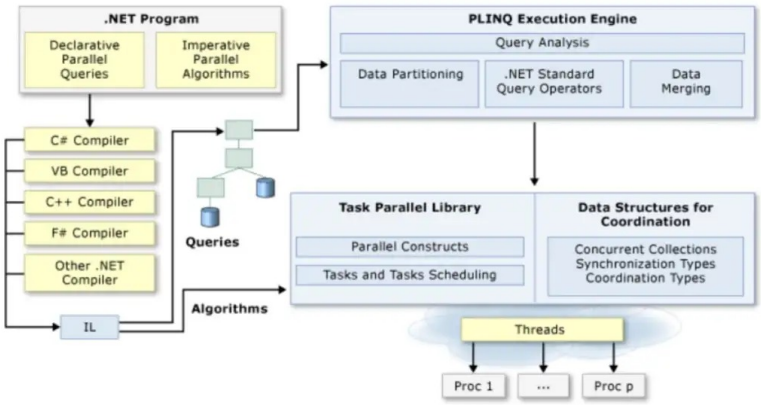
Thread Property	Description
IsAlive	Returns true if a thread has begun and hasn't yet aborted or ended regularly.
IsBackground	Obtains or modifies a Boolean value indicating whether a thread is in the background.
Name	It retrieves or modifies a thread's name. Most commonly used in debugging to identify specific threads.
Priority	It obtains or modifies a ThreadPriority value that determines how the operating system will order the scheduling of threads.
ThreadState	Obtains a ThreadState value that holds the state of a thread at that moment.

Understanding of Parallel Programming

Numerous workstations and home computers are equipped with multiple CPU cores, which allow for the simultaneous execution of numerous threads. You can parallelize your code to spread work across

numerous processors and make use of the technology.

A high-level summary of .NET's parallel programming architecture may be found in the following image.



Advanced DOTNet Tutorial

Task Parallel Library: It gives the System.Threading.Tasks.Parallel class documentation contains For and ForEach loop parallel variants, as well as for the System.Threading.Tasks.Task class is also a recommended manner of expressing asynchronous actions.

Parallel LINQ: LINQ to Objects is implemented in parallel, which greatly boosts speed in many instances.

Data Structures for Parallel Programming: It links to the documentation of types for lazy initialization, lightweight synchronization, and thread-safe collection classes.

Parallel Dignostic Tools: It provide links to the Concurrency Visualizer's documentation as well as

that for the jobs and parallel stacks debugger windows in Visual Studio.

Custom Partitioners for PLINQ and TPL: It explains the operation of partitioners and how to set up new partitioners or modify the built-in ones.

Task Schedulers: It explains the operation of schedulers and possible configurations for the default schedulers.

Lambda Expressions in PLINQ and TPL: It explains lambda expressions in C# and Visual Basic and demonstrates their use in PLINQ and the Task Parallel Library in brief.

Example: Iterating Files using Parallel Class

```
using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

using System.Security;

using System.Threading;

using System.Threading.Tasks;

class Program

{

   static void Main()

   {

      try

      {

         TraverseTreeParallelForEach(@"C:\Program Files", (f) =>

         {

            try

            {

               byte[] data = File.ReadAllBytes(f);

            }
```

```
            catch (FileNotFoundException) { }
            catch (IOException) { }
            catch (UnauthorizedAccessException) { }
            catch (SecurityException) { }
            Console.WriteLine(f);
        });
    }
    catch (ArgumentException)
    {
        Console.WriteLine(@"The directory 'C:\Program Files' does
not exist.");
    }
    Console.ReadKey();
  }
  public static void TraverseTreeParallelForEach(string root,
Action<string> action)
  {
    int fileCount = 0;
    var sw = Stopwatch.StartNew();
    int procCount = Environment.ProcessorCount;
    Stack<string> dirs = new Stack<string>();
    if (!Directory.Exists(root))
    {
      throw new ArgumentException(
        "The given root directory doesn't exist.", nameof(root));
    }
    dirs.Push(root);
    while (dirs.Count > 0)
    {
      string currentDir = dirs.Pop();
      string[] subDirs = { };
      string[] files = { };
```

```
try
{
    subDirs = Directory.GetDirectories(currentDir);
}
catch (UnauthorizedAccessException e)
{
    Console.WriteLine(e.Message);
    continue;
}
catch (DirectoryNotFoundException e)
{
    Console.WriteLine(e.Message);
    continue;
}
try
{
    files = Directory.GetFiles(currentDir);
}
catch (UnauthorizedAccessException e)
{
    Console.WriteLine(e.Message);
    continue;
}
catch (DirectoryNotFoundException e)
{
    Console.WriteLine(e.Message);
    continue;
}
catch (IOException e)
{
    Console.WriteLine(e.Message);
```

```
    continue;
}
try
{
    if (files.Length < procCount)
    {
        foreach (var file in files)
        {
            action(file);
            fileCount++;
        }
    }
    else
    {
        Parallel.ForEach(files, () => 0,
            (file, loopState, localCount) =>
            {
                action(file);
                return (int)++localCount;
            },
            (c) =>
            {
                Interlocked.Add(ref fileCount, c);
            });
    }
}
catch (AggregateException ae)
{
    ae.Handle((ex) =>
    {
        if (ex is UnauthorizedAccessException)
```

```
            {
                Console.WriteLine(ex.Message);
                return true;
            }
            return false;
        });
    }
    foreach (string str in subDirs)
        dirs.Push(str);
  }
  Console.WriteLine("Processed {0} files in {1} milliseconds",
fileCount, sw.ElapsedMilliseconds);
  }
}
```

Understanding of Native Interoperability

Calling into native code would be beneficial for the following reasons:

- There are many APIs included with operating systems that aren't found in managed class libraries.

Example: having access to an operating system or hardware management features.

- Collaborate with other components that have native ABIs (C-style ABIs) or are capable of developing them, such as managed languages capable of producing native components or Java code made available through the Java Native Interface (JNI).
- The majority of installed Windows software, including the Microsoft Office suite, registers COM components, which are representations of their programs and enable developers to use or automate them. Moreover, native interoperability is needed for this.

P/Invoke or Platform Invoke

Using P/Invoke, you can invoke functions, structs, and callbacks in unmanaged libraries directly from your managed code.

One namespace, System, contains the majority of the P/Invoke API.

Duration of action.Collaboration Services.

You can specify how you want to interact with the native component by using these two namespaces.

Let's display a command-line application's message box:

```
using System;

using System.Runtime.InteropServices;

namespace PInvokeSamples

{

  public static partial class Program

  {

    [LibraryImport("libSystem.dylib")]

    private static partial int getpid();

    public static void Main(string[] args)

    {

      int pid = getpid();

      Console.WriteLine(pid);

    }

  }

}
```

Type Marshalling

The process of changing types when they have to transition between managed and native code is known as marshalling.

Example

```
[LibraryImport("somenativelibrary.dll")]

static extern int
MethodA([MarshalAs(UnmanagedType.LPStr)]
string parameter);

// or

[LibraryImport("somenativelibrary.dll",
StringMarshalling = StringMarshalling.Utf8)]

static extern int MethodB(string parameter);
```

Understanding of Memory Management

Among the features the Common Language Runtime offers during managed execution is automatic memory management. The Common Language Runtime garbage collector manages the memory allocation and release of a program.

Allocating Memory

Compared to unmanaged memory allocation, memory allocation from the managed heap is quicker. Allocating memory for an object through the runtime is nearly as quick as allocating memory from the stack because it involves simply appending a value to a pointer.

Releasing Memory

The runtime allocates memory for huge items in a separate heap in order to increase performance. For huge items, the garbage collector releases the RAM automatically. However, this memory is not compressed to prevent moving big items around in it.

Generations and Performance

Three generations (0, 1, and 2) make up the managed heap, which is separated to maximize garbage collector speed. The computer software industry has experimented with garbage collection systems and found numerous generalizations that serve as the foundation for the runtime's garbage

collection process.

Releasing Memory for Unmanaged Resources

You may rely on the garbage collector to take care of memory management automatically for most objects that your application creates. Unmanaged resources, however, need specific cleansing.

You can allow users to expressly release memory from your object when they are done using it by including a Dispose method. Take note of Dispose and call it when needed while using an object that contains an unmanaged resource.

Advanced Dotnet Developer Salary

Understanding of MVC5

ASP.NET MVC 5 is a framework that leverages the power of ASP.NET and the.NET Framework along with well-established design patterns to create scalable, standards-based online applications.

Features of MVC5

- An extensible integrated scaffolding system through NuGet HTML 5 project templates
- The new Razor View Engine is one of the more expressive views.
- Strong hooks feature global action filters and dependency injection
- Rich JavaScript support includes JSON binding, jQuery validation, and non-intrusive JavaScript.

Adding a New Controller

Depending on the incoming URL, ASP.NET MVC calls distinct controller classes (and various action methods inside them). The default URL routing mechanism of ASP.NET MVC uses this format to determine which code to call:

public static void RegisterRoutes(RouteCollection

routes)

```
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}")
    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
    );
}
```

Adding a View

As of right now, the string that the Index method returns contains the hard-coded message for the controller class. As demonstrated in the following code, modify the index function to call the controllers' View method:

public ActionResult Index()

```
{
    return View();
}
```

Adding a New Model

Example: Movie Class

using System;

namespace MvcMovie.Models

```
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
```

```
        public decimal Price { get; set; }
    }
}
```

Example: MovieDBContext Class

```
using System;

using System.Data.Entity;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }
    public class MovieDBContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
    }
}
```

Understanding of ADO.Net

Consistent access to data sources like OLE DB and ODBC, as well as data sources like SQL Server and XML, is made possible via ADO.NET. Consumer data-sharing apps can use ADO.NET to connect to several data sources and retrieve, control, and update the data.

This page's code listings show you how to use the following ADO.NET technologies to obtain data from a database:

ADO.NET data providers:

- SqlClient (System.Data.SqlClient)
- OleDb (System.Data.OleDb)
- Odbc (System.Data.Odbc)
- OracleClient (System.Data.OracleClient)

ADO.NET Entity Framework:

- LINQ to Entities
- Typed ObjectQuery
- EntityClient (System.Data.EntityClient)

Understanding of LINQ: Language-Integrated Query

Developers no longer need to use a separate query language to create set-based queries in their application code because of Language-Integrated Query (LINQ).

XML documents, SQL databases, DataSet objects, in-memory data structures, and other enumerable data sources (i.e., data sources that implement the IEnumerable interface) can all be queried using LINQ.

LINQ to Dataset

The DataSet is a fundamental component of the widely-used disconnected programming model upon which ADO.NET is based. By applying the same query formulation methodology that is available for many other data sources, LINQ to DataSet allows developers to incorporate more sophisticated query capabilities into DataSet.

LINQ to SQL

For developers who don't need mapping to a conceptual model, LINQ to SQL is a helpful tool. You can easily apply the LINQ programming model to an existing database schema by utilizing LINQ with SQL. Developers can create .NET Framework classes that represent data by using LINQ to SQL.

LINQ to Entities

An application can interact with data as objects by modeling the data in a specific domain using the Entity Data Model, a conceptual data model.

Conclusion

We hope you have gotten fundamental ideas through this advanced .net tutorial for creating complex applications easily using advanced .Net programming concepts. Learn them comprehensively with hands-on exposure in our **advanced .Net training in Chennai.**

Share on your Social Media

Softlogic Academy

Softlogic Systems

KK Nagar [Corporate Office]

No.10, PT Rajan Salai, K.K. Nagar, Chennai – 600 078.
Landmark: Karnataka Bank Building
Phone: +91 86818 84318
Email: enquiry@softlogicsys.in
Map: Google Maps Link

OMR

No. E1-A10, RTS Food Street
92, Rajiv Gandhi Salai (OMR),
Navalur, Chennai - 600 130.

Landmark: Adj. to AGS Cinemas
Phone: +91 89256 88858
Email: info@softlogicsys.in
Map: Google Maps Link

Courses

- Python
- Software Testing
- Full Stack Developer
- Java
- Power BI
- Clinical SAS
- Data Science
- Embedded
- Cloud Computing
- Hardware and Networking
- VBA Macros
- Mobile App Development
- DevOps

Social Media Links

Review Sources

- Google
- Trustpilot
- Glassdoor
- Mouthshut
- Sulekha
- Justdial
- Ambitionbox
- Indeed
- Software Suggest
- Sitejabber