ASP.NET 비동기 프로그래밍

리얼웹	배성혁	2010-08-19 오후 1:29

비동기 프로그래밍 방식은 여러 분야에서 처리량(throughput) 증대, 확장성 향상에 좋은 방식임에는 틀림없습니다. 다만 사용 방법이 직관적이지 못해, 개발자들이 쉽게 수용하지 못하는 경우가 많습니다. 이에 몇 가지 자료와 함께 비동기 프로그래밍을 아주 쉽게 적용할 수 있는 방법을 알려 드리도록 하겠습니다.

우선 참고자료로는

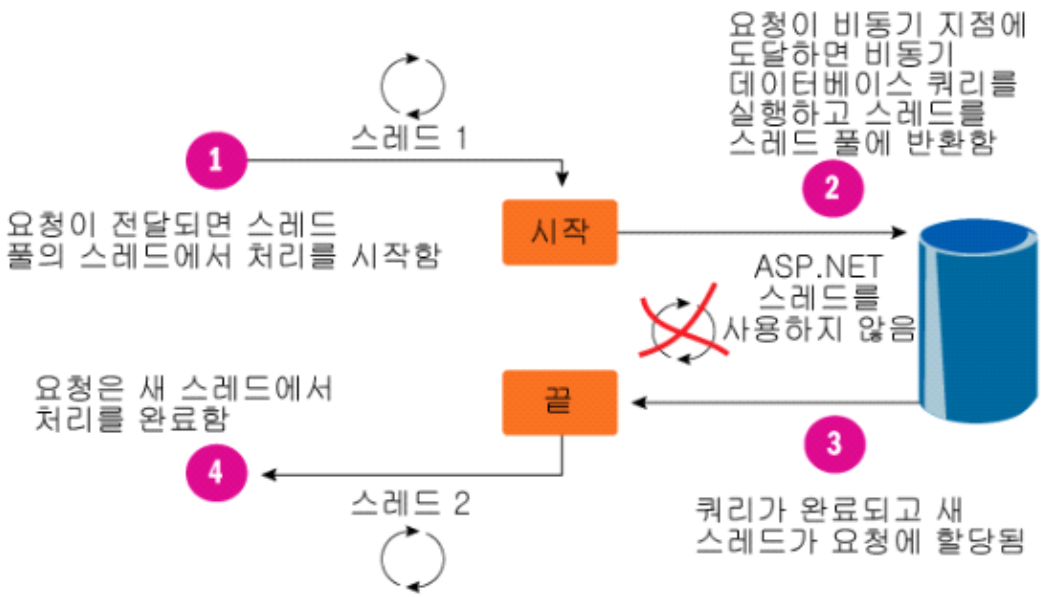
ASP.NET에서 비동기 프로그래밍을 활용하여 확장성이 우수한 응용프로그램 작성

MSDN : PageAsyncTask

를 보시면 좋습니다. 다만 위의 예제는 사용하지 않을 것이고, 방법도 전혀 다르므로, 개념만 이해하시면 됩니다.

1. ASP.NET Web Form 에서 비동기 프로그래밍

ASP.NET에서 비동기 프로그래밍의 개념을 가장 잘 나타낸 그림입니다.
보시다시피, 1번 요청 스레드에서 비동기로 작업을 시작하면, 스레드가 없어지고, 비동기 IO 작업으로 DB에 대한 처리를 담당합니다. 그 후 작업이 완료되면, 새로운 스레드가 만들어져, 비동기 작업을 마무리 합니다.



화면 캡처: 2010-08-19 오후 1:39

위의 방식은 일반적인 Computed-bounded 비동기 방식과는 달리 IO-bounded 비동기 방식이라 합니다. IO-bounded 비동기 방식은 처리율이 현저히 떨어지는 파일, Network, DB등의 IO 작업 시에 작업 스레드가 멍청히 기다리면서, 계속된 context switching에 대한 부담을 가지지 않도록 해주고, 동시에 여러 요청이 오더라도, 동시 활성화된 스레드의 수가 작으므로, 멀티스레드 관리에 드는 부담이 현저히 줄어들게 됩니다.

이에 Data-Centric 한 응용프로그램이거나 메신저 서버 등 IO 작업이 많은 서버 쪽에서도 비동기 방식으로 구현하게 되면, 많은 수의 동시 요청을 수용할 수 있게 됩니다.(이런 걸 확장성이라 합니다.)

그럼 기존 ASP.NET 2.0에서는 Web Form에 대해 두 가지 방식의 비동기 프로그래밍을 지원합니다. 첫 번째가 Page.AddOnPreRenderCompleteAsync 이고, 두 번째가 PageAsyncTask 입니다.

근데, 두 방식 모두 다 상당히 복잡합니다...
그래서 .NET 4.0 TPL (Task Parallel Library)의 TaskFactory 의 FromAsync 메소드를 이용하여 아주 쉽게, 비동기 Web Form을 구현하는 방법에 대해 알려드리겠습니다.

```
12      public partial class AsyncPage : System.Web.UI.Page
13      {
14          private static readonly Random Rnd = new ThreadSafeRandom();
15
16          protected void Page_Load(object sender, EventArgs e)
17          {
18              Response.Clear();
19
20              // 비동기로 작업할 함수를 할당한다.
21              Action action = new Action(DoProcessTask);
22
23              // 비동기 작업을 생성한다
24              var task = Task.Factory.FromAsync(action.BeginInvoke,
25                                                action.EndInvoke,
26                                                null);
27
28              // 다른 작업...
29              Thread.Sleep(Rnd.Next(10, 20));
30              Response.Write("..............................<br/>");
31
32              // 작업이 완료되기를 기다린다. (Task.ContinueWith() 메소드를 사용하여, 뒷처리도 가능하다)
33              task.Wait();
34          }
35
36          /// <summary>
37          /// 실제 작업을 처리하는 함수.
38          /// </summary>
39          protected virtual void DoProcessTask()
40          {
41              Thread.Sleep(Rnd.Next(100, 200));
42
43              Response.Write("DoProcessTask를 수행하였습니다.<br/>");
44          }
45      }
```

비동기 방식의 Web Form 수행

예제 코드는 보시다시피 비동기 작업에 대한, 시뮬레이션을 위해 Thread.Sleep() 함수를 이용합니다.
실제 작업은 DoProcessTask() 로 기존 개발과 똑같이 구현하시면 됩니다. 그 후 비동기 방식으로 실행하기 위해, TPL의 TaskFactory.FromAsync() 메소드를 이용하는 것입니다. Line 21에서 실제 작업하고자 하는 함수의 인스턴스를 받아, line 24 처럼 FromAsync 메소드를 호출하기만 하면 됩니다.

물론 FromAsync 메소드는 overload가 상당히 많으므로, 작업함수(DoProcessTask())의 다양한 signature를 수용할 수 있습니다.
응용을 위해서는, 위의 DoProcessTask() 메소드 내에서, DB에 접속하여 정보를 가져오는 작업을 구현하면 됩니다.

TPL을 활용해서 더 좋은 장점은 FromAsync로 만드는 작업을 여러 개 만들면, 병렬로 수행할 수 있다는 것입니다. 즉 여러 개의 비동기 IO-Bounded 작업을 병렬로 수행할 수 있다는 뜻입니다.

```
12      public partial class AsyncPage : System.Web.UI.Page
13      {
14          private static readonly Random Rnd = new ThreadSafeRandom();
15
16          protected void Page_Load(object sender, EventArgs e)
17          {
18              Response.Clear();
19
20              var tasks = new List<Task>();
21
22              // 비동기로 작업할 함수를 할당한다.
23              Action action = new Action(DoProcessTask);
24
25              // 비동기 작업을 생성한다
26              var task = Task.Factory.FromAsync(action.BeginInvoke,
27                                                action.EndInvoke,
28                                                null);
29              tasks.Add(task);
30
31              var action2 = new Action<object>(state => DoProcessTask2(state));
32              var task2 = Task.Factory.FromAsync(action2.BeginInvoke,
33                                                 action2.EndInvoke,
34                                                 "Second Task",
35                                                 null);
36
37              tasks.Add(task2);
38
39              // 다른 작업...
40              Thread.Sleep(Rnd.Next(10, 20));
41              Response.Write("................................<br/>");
42
43              // 작업이 완료되기를 기다린다. (Task.ContinueWith() 메소드를 사용하여, 뒷처리도 가능하다)
44              Task.WaitAll(tasks.ToArray());
45          }
46
47          /// <summary>
48          /// 실제 작업을 처리하는 함수.
49          /// </summary>
50          protected virtual void DoProcessTask()
51          {
52              Thread.Sleep(Rnd.Next(100, 200));
53
54              Response.Write("DoProcessTask를 수행하였습니다.<br/>");
55          }
56
57          /// <summary>
58          /// 실제 작업을 처리하는 함수.
59          /// </summary>
60          protected virtual void DoProcessTask2(object state)
61          {
62              Thread.Sleep(Rnd.Next(100, 200));
63
64              Response.Write("DoProcessTask2를 수행하였습니다.  state=" + state + "<br/>");
65          }
66      }
```

복수의 비동기 작업을 병렬로 수행하는 예

이렇게 사용한다면, Bottleneck도 걸리지 않고, 병렬로 작업을 수행하게 되므로, 처리율과 속도 모두 만족스러운 향상을 가져올 것입니다.

TPL 사용이 .NET 4.0이상에서만 가능하다는 편견은 버리십시요. Reactive Extensions for .NET 3.5.1에 System.Threading.dll 이 TPL을 제공하니, .NET 3.5.1에서도 병렬 프로그래밍이 가능합니다. (RCL도 마찬가지구요)

2. IHttpAsyncHandler를 구현한 비동기 프로그래밍

우선 MSDN의 비동기 HTTP 처리기 만들기 를 보십시요. 설명서대로 구현하려면, 여러 Http 처리기를 만들어야 하는 사람의 입장에서, 상당히 부담이 될 것입니다.

HttpAsync...

이에 IHttpAsyncHandler 를 구현한 기본클래스를 제작하여, 일반 개발자는 Web Form 비동기 방식 구현과 마찬가지로, 전혀 비동기에 대한 어떠한 구현도 하지 않도록 해줍니다. (첨부파일 참고)

예제는 다음과 같습니다.

```
 6      /// <summary>
 7      /// AsyncEcho의 요약 설명입니다.
 8      /// </summary>
 9      public class AsyncEcho : HttpAsyncHandlerBase
10      {
11          /// <summary>
12          /// 비동기 방식의 HttpHandler의 Main 함수
13          /// </summary>
14          /// <param name="context"></param>
15          protected override void DoProcessRequest(HttpContext context)
16          {
17              base.DoProcessRequest(context);
18
19              WriteEcho(context);
20
21              context.Response.Flush();
22              context.Response.End();
23          }
24
25          /// <summary>
26          /// 사용자 구현 함수
27          /// </summary>
28          /// <param name="context"></param>
29          private static void WriteEcho(HttpContext context)
30          {
31              context.Response.ContentType = "text/plain";
32              context.Response.Write("Hello World.");
33          }
34      }
```

IHttpAsyncHandler 를 구현한 예제

위 예제 코드에서는 비동기 관련 코드는 전혀 보이지 않습니다. 다만, 개발자는 DoProcessRequest() 메소드를 재 정의하여, 자신이 원하는 코드를 작성하면 됩니다. DB 작업을 기존 방식대로 작업해도 됩니다.

어떻습니까? MSDN 에 나온 방식은 제가 봐도, 구현 자체가 문제가 아니라 확산이 문제였는데, 위의 두 가지 방식은 아주 손 쉽게 작업할 수 있도록 했습니다.

물론 HttpAsyncHandlerBase의 경우에는, 상속 체계를 바꾸는 문제가 있을 수 있지만, 많은 개발자들이 HttpHandler를 도입하는 단계라면, 도입 효과가 상당하리라 예상됩니다.^^

3. 참고 자료

a. CLR via C# 2nd Ed. Chapter 23, 24

b. CLR via C# 3rd Ed. Chapter 25 ~ 29
c. Pro .NET Parallel Programming in C#
d. Pattern Of Parallel Programming (svn/public/research 에 parallel 관련)