ADO.NET 비동기 I/O Operation

2010년 8월 10일 화요일
오후 7:02

ADO.NET 2.0 이상부터 SQL Server 계열에 대한 SqlCommand 클래스가 비동기 IO 작업을 지원합니다.
비동기 IO 작업의 장점은 여러 문서를 보면, 알 수 있고, MSDN에서 찾아보면 됩니다.
여기서는 TPL (Task Parallel Library) 를 이용하여 아주 쉽게, ADO.NET에 대한 비동기 IO 작업을 구현할 수 있음을 보여드리겠습니다.

비동기 IO 작업을 위한 사전 작업

1. Connection String에 **"Asynchronous Processing=true;"** 항목을 추가합니다.
2. 비동기 IO 작업은 성능이 아주 좋아, Connection Pool을 많이 차지할 수 있으므로 최대 Pool Size도 늘려주면 좋습니다. Connection String에 **"Max Pool Size=200;"** 이렇 게 변경하십시요. 기본은 100입니다.

RCL.Data.Ado.Sql.SqlCommandAsync 정적 클래스는 다음과 같은 비동기 IO 작업을 위한 함수를 제공합니다.
ExecuteNonQuery, ExecuteReader, ExecuteScalar, ExecuteXmlReader 메소드에 대한 비동기 작업을 수행하는 확장 메소드를 정의했습니다.

```
public static Task<int> ExecuteNonQueryAsync(this SqlDatabase sqlDatabase, SqlCommand sqlCommand)
{
    sqlCommand.ShouldNotBeNull("sqlCommand");

    sqlCommand.Connection = (SqlConnection)sqlDatabase.CreateConnection();
    sqlCommand.Connection.Open();

    return
        Task<int>.Factory
            .FromAsync(sqlCommand.BeginExecuteNonQuery,
                       sqlCommand.EndExecuteNonQuery,
                       sqlCommand);
}
```

```
public static Task<SqlDataReader> ExecuteReaderAsync(this SqlDatabase sqlDatabase, SqlCommand sqlCommand)
{
    sqlCommand.ShouldNotBeNull("sqlCommand");

    if (IsDebugEnabled)
        log.Debug(@"ExecuteScalar를 비동기 실행합니다. CommandText=" + sqlCommand.CommandText);

    if (sqlCommand.Connection == null)
    {
        sqlCommand.Connection = (SqlConnection)sqlDatabase.CreateConnection();
        sqlCommand.Connection.Open();
    }
    return
        Task<SqlDataReader>.Factory
            .FromAsync(sqlCommand.BeginExecuteReader,
                       sqlCommand.EndExecuteReader,
                       sqlCommand);
}
```

```
public static Task<object> ExecuteScalarAsync(this SqlDatabase sqlDatabase, SqlCommand sqlCommand)
{
    sqlCommand.ShouldNotBeNull("sqlCommand");

    if (IsDebugEnabled)
        log.Debug(@"ExecuteScalar를 비동기 실행합니다. CommandText=" + sqlCommand.CommandText);

    sqlCommand.Connection = (SqlConnection)sqlDatabase.CreateConnection();
    sqlCommand.Connection.Open();

    return
        Task<SqlDataReader>.Factory
            .FromAsync(sqlCommand.BeginExecuteReader,
                       sqlCommand.EndExecuteReader,
                       sqlCommand)
            .ContinueWith<object>(task =>
                                  {
                                      try
                                      {
                                          using (var reader = task.Result)
                                          {
                                              if (reader.Read())
                                                  return reader.GetValue(0);
                                          }
                                          return null;
                                      }
                                      catch (Exception ex)
                                      {
                                          if (IsErrorEnabled)
                                              log.Error("비동기 ExecuteScalar 실행에 실패했습니다.", ex);
                                          return null;
                                      }
                                  },
                                  TaskContinuationOptions.OnlyOnRanToCompletion |
                                  TaskContinuationOptions.ExecuteSynchronously);
}
```

```
public static Task<XmlReader> ExecuteXmlReaderAsync(this SqlDatabase sqlDatabase, SqlCommand sqlCommand)
{
    sqlCommand.ShouldNotBeNull("sqlCommand");

    if (IsDebugEnabled)
        log.Debug(@"ExecuteXmlReader를 비동기 실행합니다. CommandText=" + sqlCommand.CommandText);

    if (sqlCommand.Connection == null)
    {
        sqlCommand.Connection = (SqlConnection)sqlDatabase.CreateConnection();
        sqlCommand.Connection.Open();
    }

    return
        Task<XmlReader>.Factory
            .FromAsync(sqlCommand.BeginExecuteXmlReader,
                       sqlCommand.EndExecuteXmlReader,
                       sqlCommand);
}
```

코드를 보면 TPL (Task Parallel Library) 의 FromAsync() 메소드를 이용하여, 아주 손쉽게 비동기 프로그래밍 모델에 대한 구현 작업을 하였음을 알 수 있죠.

아직 모든 단위 테스트를 수행한 것은 아니지만, ExecuteReaderAsync, ExecuteScalarAsync 를 수행했을 경우, 비동기 IO 작업이 아주 잘 이루어집니다. 좀 더 친절한 예제는 MSDN을 참고하여 더 추가하도록 하겠습니다.

```
[TestFixture]
[ThreadedRepeat(500)]
public class SqlCommandAsyncTestCase : AdoTestFixtureBase
{
    public SqlDatabase NWindDatabase...

    [Test]
    public void Can_ExecuteReaderAsync()
    {
        using (var command = (SqlCommand)NWindDatabase.GetSqlStringCommand("SELECT * FROM Customers"))
        {
            var readerTask = SqlCommandAsync.ExecuteReaderAsync(NWindDatabase, command);

            Console.WriteLine("비동기 실행했음...");

            using (var reader = readerTask.Result.ToAdoDataReader())
            {
                Assert.IsTrue(reader.Read());
                while (reader.Read())
                {
                    // Nothing to do.
                }
            }
        }
    }

    [Test]
    public void Can_ExecuteScalarAsync()
    {
        using (var command = (SqlCommand)NWindDatabase.GetSqlStringCommand("SELECT COUNT(*) FROM Customers"))
        {
            var readerTask = SqlCommandAsync.ExecuteScalarAsync(NWindDatabase, command);

            Console.WriteLine("비동기 실행했음...");

            var customerCount = RwConvert.DefValue(readerTask.Result, 0);
            Assert.IsTrue(customerCount > 0);
        }
    }
}
```

이렇게 비동기 IO 작업을 하게 되면, CPU 점유율도 높이지 않고, 동시에 많은 작업을 수행할 수 있습니다.