Log in / create account

Fluent NHibernate wiki

Database configuration

From Fluent NHibernate

Fluent NHibernate provides an API for configuring your database, this can be used in combination with the fluent configuration or on it's own. I'm going to show some of the various ways you can use this API.

All of the following examples are best used when placed inside the Database call of the fluent configuration.

```
Fluently.Configure()
  .Database(/* examples here */)
  .Mappings(...)
  .BuildSessionFactory();
```

NHibernate has a provider and driver design for it's database interaction, these drivers handle the various different peculiarities of each database and allow NHibernate to remain agnostic; however, it does mean you need to pick the right ones for your application. This is where Fluent NHibernate comes in, we currently provide six of the most popular database providers, and wrap them up in a nice, easy to use, API.

All the database providers are located in the `FluentNHibernate.Cfg.Db` namespace, and are reasonably discoverable. Each configuration starts with a static property that specifies which version you want to use, if there's only one available then you should use the `Standard` property.

For example, if you're using MS SQL Server 2005, then you should use `MsSqlConfiguration.MsSql2005` to begin your configuration, alternatively if you're using SQLite (http://www.sqlite.org) you can use `SQLiteConfiguration.Standard`.

Each configuration instance has several options available on it, some which are shared across all providers, and others that are specific to a version or database.

The most common thing you'll need to set is the connection string for the database. There are several available options for setting your connection string; if you've got it in a string you can supply that, you can have it read from an `appSetting` or a `connectionString` from the App.config, or you can build it up yourself.

The ConnectionString property on the configurations takes a lambda that alters the conection string, or a string if you're just providing a pre-prepared connection string, here are some examples:

```
// raw string
MsSqlConfiguration.MsSql2005
```

```
  .ConnectionString("a raw string");

// from appSettings
MsSqlConfiguration.MsSql2005
  .ConnectionString(c => c
    .FromAppSetting("appSettingKey"));

// manually constructed
MsSqlConfiguration.MsSql2005
  .ConnectionString(c => c
    .Server("my-server")
    .Database("database-name")
    .Username("jamesGregory")
    .Password("password1"));
```

Similar to ConnectionString is Cache which, as the name suggests, configures NHibernate's cache. It's fairly self explanatory again, so I'll just show a simple example.

```
MsSqlConfiguration.MsSql2005
  .Cache(c => c
    .UseQueryCache()
    .UseMinimalPuts());
```

This example sets the NHibernate `cache.use_query_cache` and `cache.use_minimal_puts` properties to `true`.

There are several other methods available in addition to ConnectionString and Cache, but they're all self explanatory. Here's an example of a couple of calls:

```
MsSqlConfiguration.MsSql2005
  .UseOuterJoin()
  .DefaultSchema("dbo");
```

Using Fluent NHibernate does not require you to sacrifice portability across RDBMSes. You can use the traditional approach to NHibernate configuration (which uses the standard .NET configuration mechanisms) to determine database drivers and connection strings this way:

```
var cfg = new NHibernate.Cfg.Configuration();
cfg.Configure(); // read config default style
Fluently.Configure(cfg)
    .Mappings(
      m => m.FluentMappings.AddFromAssemblyOf<Entity>())
    .BuildSessionFactory();
```

Bringing it all together

Here's an example of a complete MS SQL 2005 configuration:

```
MsSqlConfiguration.MsSql2005
  .ConnectionString(c => c
    .FromAppSetting("connectionString"))
  .Cache(c => c
    .UseQueryCache()
    .ProviderClass<HashtableCacheProvider>())
  .ShowSql();
```

When using the fluent configuration API, you would combine it like so:

```
Fluently.Configure()
  .Database(MsSqlConfiguration.MsSql2005
    .ConnectionString(c => c
      .FromAppSetting("connectionString"))
    .Cache(c => c
      .UseQueryCache()
      .ProviderClass<HashtableCacheProvider>())
    .ShowSql())
  .Mappings(m => m
    .FluentMappings.AddFromAssemblyOf<Entity>())
  .BuildSessionFactory();
```

However, if you're not using the FluentConfiguration api, then you can call either ConfigureProperties with an NHibernate Configuration instance, or ToProperties to create an `IDictionary<string, string>` of the properties.

Some shortcuts

Depending on which database you're using, there may be some shortcuts available for common configurations. For example, SQLite can be used as an in-memory database, or it can be pointed at a specific file; in these cases we've provided shortcuts that build up the connection string and settings for these configurations.

```
SQLiteConfiguration.Standard
  .InMemory();

SQLiteConfiguration.Standard
  .UsingFile("database.db");
```

Using SQLite in ASP.NET

Using a SQLite database for your web application is convenient because it almost gives you the performance and capabilities of an advanced DBMS (like MS-SQL or Oracle) but saves you from having to host such a database while also providing very easy x-copy deploy and backup possibilities.

Usually you would want to place the SQLite databases in the App_Data folder of your ASP.NET web project while a base class library isolates the functionality of using Fluent NHibernate to provide object mapping, connection configuration and POCO entities.

To do this, first, place any SQLite databases you may have in the '~/App_Data/' folder of your web application project.

Then, add a normal connection string in the web.config of your application:

```
<connectionStrings>
    <add name="MyConnectionString" connectionString="data source=|DataDirectory|MySQLite.db;" providerNam
</connectionStrings>
```

Note that the |DataDirectory| keyword neatly maps to '~/App_Data/' for web applications.

Create a class in your base project (really anywhere but that's the best place for it, it's nice to keep your web project clean for mostly GUI related stuff) containing the following code:

```
public static class MySQLiteSessionFactory
{
  public static ISessionFactory CreateSessionFactory()
  {
      return Fluently.Configure()
          .Database(
          SQLiteConfiguration.Standard.ConnectionString(
          c => c.FromConnectionStringWithKey("MyConnectionString")))
          .Mappings(x => x.FluentMappings.AddFromAssembly(Assembly.GetExecutingAssembly()))
          .BuildSessionFactory();
  }
}
```

Adjust this code to your liking, like for example writing out the method calls more, but the important thing here is that you are telling the Fluent API to go look in the configuration file (which chooses the web.config as main source, NOT the app.config in your base class library) for the "MyConnectionString" connectionstring.

Also, this code includes the "Mappings" method. Since this code is in my base class library project, that assembly would also contain all Fluent NHibernate mappings so in this code it configures the connection to just use all objects in the current assembly.

And that's it! Check out the Getting started guide to do some selections and updates on your database.

Retrieved from "http://wiki.fluentnhibernate.org/Database_configuration"

Page Discussion View source History

Search