

.NET DDD 实战专题一：前期准备之EF CodeFirst - 文章 - 伯乐在线



一、前言

从去年已经接触领域驱动设计（Domain-Driven Design）了，当时就想自己搭建一个DDD框架，所以当时看了很多DDD方面的书，例如领域驱动模式与实战，领域驱动设计：软件核心复杂性应对之道和领域驱动设计C# 2008实现等书，由于当时只是看看而已，并没有在自己代码中进行实现，只是初步了解一些DDD分层的思想和一些基本概念，例如实体，聚合根、仓储等概念，今年有机会可以去试试面试一个架构岗位的时候，深受打击，当面试官问起是否在项目中使用过DDD思想来架构项目时，我说没有，只是了解它的一些基本概念。

回来之后，便重新开始学习DDD，因为我发现做成功面试一个架构师，则必须有自己的一个框架来代表自己的知识体系，而不是要你明白这个基本概念。此时学习便决定一步步来搭建一个DDD框架。但是这次的过程并不是像dax.net那样，一开始就去搭建框架，然后再用一个实际的项目来做演示框架的使用。因为我觉得这样对于一些初学者来学习的话，难度比较大，因为刚开始写框架根本看到什么，而且看dax.net的Apworks框架很多代码也不明白他为什么这么写的，从框架代码并不能看出作者怎么一步步搭建框架的，读者只能一下子看到整个成型的框架，对于刚接触DDD的朋友难度非常大，以至于学习了一段时间的DDD之后，就放弃了。

这个感觉本人学习过程中深有体会。所以本系列将直接把DDD的思想应用到一个实例项目中，完全实例项目后，再从中抽取一个DDD框架出来，并且会一步步介绍如何将DDD的思想应用到一个实际项目中

（dax.net中ByteartRetail项目也是直接给出一个完整的DDD演示项目的，并没有记录搭建过程，同样对于读者学习难度很大，因为一下子来吸收整个项目的知识，接受不了，读者自然就心灰意冷，也就没有继续学习DDD的动力了）。本文并没有开始介绍DDD项目的实际实现，而是一个前期准备工作，因为DDD项目中一般会使用的实体框架来完成，作为.NET阵营的人，自然首先会使EntityFramework。下面就具体介绍下EF中code First的实现，因为在后面的DDD项目实现中会使用到EF的CodeFirst。

二、EF CodeFirst的实现步骤

因为我之前没怎么接触EF的CodeFirst实现，所以在看dax.net的ByteartRetail项目的时候，对EF仓储的实现有疑惑，所以去查阅相关EF的教程发现，原来应用了EF中的CodeFirst。所以把过程记录下来。下面就具体介绍下使用EF CodeFirst的具体实现步骤。

步骤一：创建一个Asp.net MVC 4 Web项目，创建成功后，再添加一个Model类

CodeFirst自然是先写实体类了，这里演示的是一个Book实体类，具体类的实现代码如下：

```
1
2
3
4
5
6
7
8
9
public class Book
{
    public int BookId { get; set; }
    public string BookName { get; set; }
    public string Author { get; set; }
    public string Publisher { get; set; }
    public decimal Price { get; set; }
    public string Remark { get; set; }
}
```

将使用这个类表示数据库中的一个表，每个Book类的实例对应数据库中的一行，Book类中的每个属性映射为数据库中的一列。

步骤二：创建“BookDbContext”的类

使用Nuget安装Entity Framework，安装成功后，在Models文件夹下新建一个“BookDbContext”的类，将类派生自“DbContext”类（命名空间为System.Data.Entity，dll在EntityFramework），具体BookDbContext的实现如下：

```
1
2
3
4
5
6
using System.Data.Entity;
public class BookDbContext : DbContext
{
    public DbSet <Books> { get; set; }
}
```

BookDbContext代表EF中Book在数据库中的上下文对象，通过DbSet使实体类与数据库关联起来。Books属性表示数据中的数据实体，用来处理数据的存取与更新。

步骤三：添加数据库连接

在Web.config文件中，修改数据库连接字符串的配置，这里将数据库连接的name属性设置为BookDbContext，后面代码将会使用到该名字，并根据连接创建相应的数据库。

```
1
2
3
<connectionStrings>
    <add name="BookDbContext" connectionString="Data Source=(LocalDb)\v11.0;Initial
Catalog=BookDB;Integrated Security=True;AttachDBFilename=|DataDirectory|\BookDB.mdf"
providerName="System.Data.SqlClient"/>
</connectionStrings>
```

步骤四：为Book创建控制器和Index视图

首先创建一个控制器：在”Controlllers”上右键>添加>控制器，在打开的添加控制器对话框中，将控制器的名称改为”BookController”，模板选择”空控制器”。修改BookController的代码为如下所示：

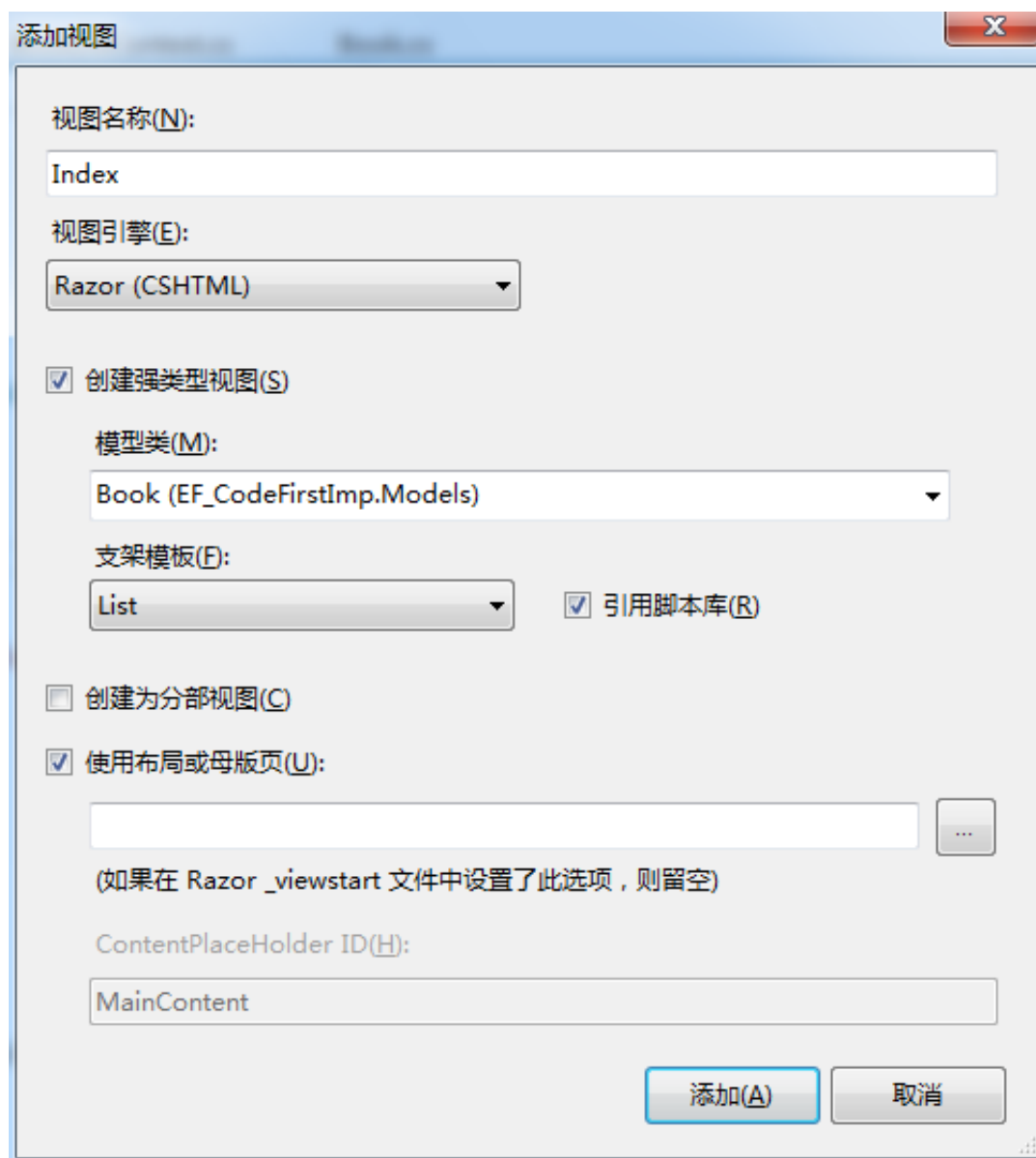
```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
public class BookController : Controller
{
    readonly BookDbContext _db = new BookDbContext();
    //
    // GET: /Book/
    public ActionResult Index()
    {
```

```
var books = from b in _db.Books
              where b.Author == "Learninghard"
              select b;

return View(books.ToList());
}

public ActionResult Create()
{
    return View();
}
}
```

在Index方法内右键>”添加视图”，在打开的”添加视图“对话框，勾选”创建强类型视图“，在模型类列表中选择”Book“（如果选择列表为空，则需要首先编译下项目），在支架模板列表中选择”List“，具体如下图所示：



点击添加按钮，VS为我们创建了Index.cshtml文件，修改Index.cshtml代码为如下所示的代码：

1
2

- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
- 26
- 27
- 28
- 29
- 30
- 31
- 32
- 33
- 34
- 35
- 36
- 37
- 38
- 39
- 40
- 41
- 42
- 43
- 44

45
46
47
48
49
50
51
52
53
54
55
56
57

```
@model IEnumerable<EF_CodeFirstImp.Models.Book>
@{
    ViewBag.Title = "图书列表-EF CodeFirstImp";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("添加图书", "Create")
</p>
<table>
    <tr>
        <th>
            图书名称
        </th>
        <th>
            作者
        </th>
        <th>
            出版社
        </th>
        <th>
            价格
        </th>
        <th>
            备注
        </th>
    </tr>
    @foreach (var item in Model) {
        <tr>
```

```
<td>

    @Html.DisplayFor(modelItem => item.BookName)

</td>

<td>

    @Html.DisplayFor(modelItem => item.Author)

</td>

<td>

    @Html.DisplayFor(modelItem => item.Publisher)

</td>

<td>

    @Html.DisplayFor(modelItem => item.Price)

</td>

<td>

    @Html.DisplayFor(modelItem => item.Remark)

</td>

<td>

    @Html.ActionLink("编辑", "Edit", new { id=item.BookId }) |
    @Html.ActionLink("查看", "Details", new { id=item.BookId }) |
    @Html.ActionLink("删除", "Delete", new { id=item.BookId })

</td>

</tr>

}

</table>
```

编译并运行程序，在浏览器中输入地址：<http://localhost:2574/Book>，得到的运行结果如下：



尽管没有数据，但EF已经为我们创建了相应的数据库了。此时在App_Data文件夹下生成了BookDB数据库，在解决方案点击选择所有文件，将BookDB数据库包括在项目中。

步骤五：添加Create视图

在BookController中的Create方法右键添加视图来添加Create视图，此时模型类仍然选择Book，但支架模板选择” Create”。添加成功后，VS会在Views/Book目录下添加一个Create.cshtml文件，由于这里选择了Create支架框架，所以VS会为我们生成一些默认的代码。在这个视图模板中，指定了强类型Book作为它的模型类，VS检查Book类，并根据Book类的属性，生成对应的标签名和编辑框，我们修改标签使其显示中文，修改会的代码如下所示：

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
```


36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

```
@model EF_CodeFirstImp.Models.Book
```

```
@{
```

```
    ViewBag.Title = "添加图书";
```

```
}
```

```
<h2>添加图书</h2>
```

```
@using (Html.BeginForm()) {
```

```
    @Html.AntiForgeryToken()
```

```
    @Html.ValidationSummary(true)
```

```
    <fieldset>
```

```
<legend>图书</legend>
<div class="editor-label">
    图书名称:
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.BookName)
    @Html.ValidationMessageFor(model => model.BookName)
</div>
<div class="editor-label">
    作者:
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Author)
    @Html.ValidationMessageFor(model => model.Author)
</div>
<div class="editor-label">
    出版社:
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Publisher)
    @Html.ValidationMessageFor(model => model.Publisher)
</div>
<div class="editor-label">
    价格:
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Price)
    @Html.ValidationMessageFor(model => model.Price)
</div>
<div class="editor-label">
    备注:
</div>
<div class="editor-field">
    @Html.EditorFor(model => model.Remark)
    @Html.ValidationMessageFor(model => model.Remark)
</div>
<p>
    <input type="submit" value="添加" />
</p>
</fieldset>
}
<div>
```

```
@Html.ActionLink("Back to List", "Index")
</div>
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

分析上面的代码：

- `@model EF_CodeFirstImp.Models.Book`：指定该视图模板中的“模型”强类型化是一个Book类。
- `@using (Html.BeginForm()) { }`：创建一个Form表单，在表单中包含了对于Book类所生成的对应字段。
- `@Html.EditorFor(model => model.BookName)`：根据模型生成模型中BookName的编辑控件（生成一个Input元素）
- `@Html.ValidationMessageFor(model => model.BookName)`：根据模型生成模型中BookName的验证信息。

编译项目，在浏览器中输入`http://localhost:2574/Book`，然后点击”添加图书“来查看添加图书界面，运行结果如下图所示：



步骤六：添加Create的Postback方法

添加Create视图后，此时仅仅是把添加图书界面显示出来，并不能实际的完成图书的添加操作，因

为我们还没有添加按钮的处理方法，没有实际的处理添加事件，为了完成数据的添加，在BookController类添加一个Create的Postback方法，完整的BookController代码如下所示：

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
public class BookController : Controller
{
    readonly BookDbContext _db = new BookDbContext();
    //
    // GET: /Book/
```

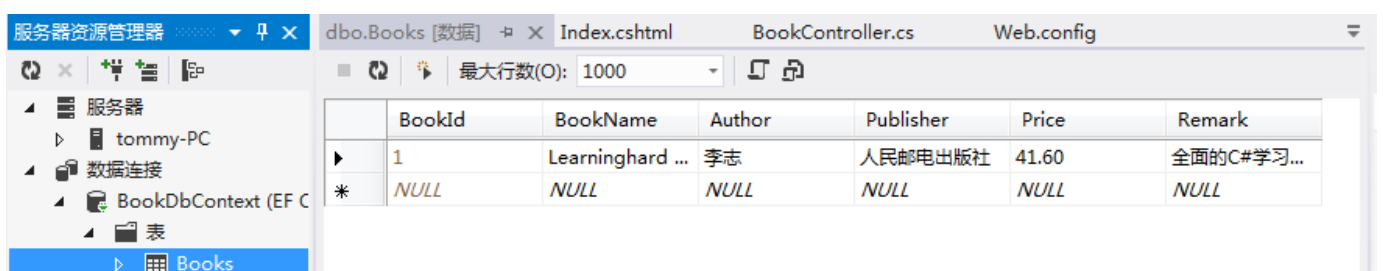
```
public ActionResult Index()
{
    var books = from b in _db.Books
                 where b.Author == "Learninghard"
                 select b;

    return View(books.ToList());
}

public ActionResult Create()
{
    return View();
}

[HttpPost]
public ActionResult Create(Book book)
{
    if (ModelState.IsValid)
    {
        _db.Books.Add(book);
        _db.SaveChanges();
        return RedirectToAction("Index");
    }
    else
        return View(book);
}
```

这时，我们在添加图书界面中输入数据，并点击”添加“按钮时，数据库中就会添加一行记录。打开数据库，我们将看到如下截图的数据：



	BookId	BookName	Author	Publisher	Price	Remark
▶	1	Learninghard ...	李志	人民邮电出版社	41.60	全面的C#学习...
*	NULL	NULL	NULL	NULL	NULL	NULL

步骤七：设置视图模型的数据验证

我们可以在模型类中显式地追加一个验证规则，使得对输入数据进行验证。修改之前的Book类为如下：

```
1
2
3
4
5
6
7
```

```
8
9
10
11
12
13
14
using System.ComponentModel.DataAnnotations; //需要额外添加该命名空间
public class Book
{
    public int BookId { get; set; }
    [Required(ErrorMessage = "必须输入图书名称")]
    [StringLength(maximumLength: 100, MinimumLength = 1, ErrorMessage = "最多允许输入100个字符")]
    public string BookName { get; set; }
    [Required(ErrorMessage = "必须输入作者名称")]
    public string Author { get; set; }
    [Required(ErrorMessage = "必须输入出版社名称")]
    public string Publisher { get; set; }
    public decimal Price { get; set; }
    public string Remark { get; set; }
}
```

此时重新运行，并打开添加图书页面，当不输入任何数据的时候，点击”添加“按钮时，界面会出现一些提示信息，并阻止我们进行数据的提交操作，具体的结果界面如下所示：



The screenshot shows a web application interface for adding a book. At the top, there is a header with the text '将你的徽标放置在此处' and navigation links for '注册', '登录', '主页', '关于', and '联系方式'. The main content area is titled '添加图书' (Add Book). Below the title, there are five input fields with corresponding labels and validation messages:

- 图书名称:** (Book Name) - Validation message: 必须输入图书名称 (Must input book name).
- 作者:** (Author) - Validation message: 必须输入作者名称 (Must input author name).
- 出版社:** (Publisher) - Validation message: 必须输入出版社名称 (Must input publisher name).
- 价格:** (Price) - Validation message: Price 字段是必需的。 (Price field is required).
- 备注:** (Remark) - No validation message.

At the bottom of the form, there is a '添加' (Add) button and a 'Back to List' link.





另外，EF创建数据库除了在第三步中添加连接字符串的方式外，还可以定义defaultConnectionFactory中添加parameters节点的方式来完成，dax.net中ByteartRetail项目中就是采用了这种方式。下面注释

掉connectionStrings节点，在defaultConnectionFactory添加如下parameters节点：

```
1
2
3
4
5
6
7
8
9
10
<entityFramework>
    <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework" >
    <parameters>
        <parameter value="Data Source=(LocalDb)\v11.0;Initial
Catalog=BookDB_2;Integrated Security=True;AttachDBFilename=|DataDirectory|\BookDB_2.mdf"/>
    </parameters>
    </defaultConnectionFactory>
    <providers>
        <provider invariantName="System.Data.SqlClient"
type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />
    </providers>
</entityFramework>
```

entityFramework节点是使用Nuget添加Entity Framework后自动添加的节点。下面测试下这种方案是否可以成功生成BookDB_2数据库呢？

运行项目，在浏览器中输入地址：<http://localhost:2574/Book>，显示界面成功后，你将在你的App_Data目录下看到如下截图：

 BookDB.mdf	2015/4/26 12:47	SQL Server Data...	3,136 KB
 BookDB_2.mdf	2015/4/26 12:53	SQL Server Data...	2,112 KB
 BookDB_2_log.ldf	2015/4/26 12:53	SQL Server Data...	1,024 KB
 BookDB_log.ldf	2015/4/26 12:47	SQL Server Data...	1,024 KB

从上图可以发现，这种方式同样成功生成了数据库。

三、总结

到这里，领域驱动设计实战系列的前期准备就结束了，本文主要介绍了如何使用EF CodeFirst的功能自动生成数据库、以及实体的添加、查看操作等。这里也简单介绍了MVC相关内容。下一专题将介绍如何利用DDD的思想来构建一个简单的网站，接下来的系列就逐一加入DDD的概念来对该网站进行完善。

本文所有源码下载：[EFCodeFirstImp.zip](#)

加入伯乐在线专栏作者。扩大知名度，还能得赞赏！详见《[招募专栏作者](#)》

1 赞 收藏 [评论](#)



合作联系

Email: bd@jobbole.com

QQ: 2302462408 (加好友请注明来意)

更多频道

[小组](#) - 好的话题、有启发的回复、值得信赖的圈子

[头条](#) - 分享和发现有价值的内容与观点

[相亲](#) - 为IT单身男女服务的征婚传播平台

[资源](#) - 优秀的工具资源导航

[翻译](#) - 翻译传播优秀的外文文章

[文章](#) - 国内外的精选文章

[设计](#) - UI, 网页, 交互和用户体验

[iOS](#) - 专注iOS技术分享

[安卓](#) - 专注Android技术分享

[前端](#) - JavaScript, HTML5, CSS

[Java](#) - 专注Java技术分享

[Python](#) - 专注Python技术分享