第5章 Razor Pages

Razor Pages 是在 ASP.NET Core 中创建基于页面或基于窗体的应用程序的首选方法。基于 Razor Pages 的编码方式比使用控制器和视图更为轻松和高效。本章主要介绍 Razor Pages 项目、模型对象、布局、Razor 语法的概念和应用。

本章主要学习目标如下：

- 掌握 Razor Pages 项目的创建。
- 掌握模型对象的概念和应用。
- 掌握布局的概念和应用。
- 掌握 Razor 语法的概念和应用。

5.1 Razor Pages 简介

Razor Pages 简化了传统的 MVC 模式，通过使用视图与模型来完成网页的渲染和业务逻辑的处理。模型里包含了数据和方法，通过绑定技术与视图建立联系，即 Razor Pages 将操作组合在一起，并在一个类中调用一个名为 PageModel 的 ViewModel 类，并将此类链接到名为 Razor 页面的视图，这往往会使 Razor Pages 及其处理程序更小、更集中，同时可以更轻松地查找和处理更改应用程序所需的所有文件。Razor Pages 页面是 MVC 框架的一种简化应用，可以使以页面为中心的编码方案更简单、高效。

所有 Razor Pages 都在 ASP.NET Core 项目根目录中的“页面”文件夹，即/Pages 目录中。Razor Pages 在此文件夹中根据其名称和位置使用路由约定。

ASP.NET Core Web Application 项目文件包括以下部分。

1. Pages 文件夹

Pages 文件夹包含 Razor 页面和支持文件。每个 Razor 页面都是一组文件。

(1) cshtml 文件：其中包含使用 Razor 语法的 C# 代码的 HTML 标记。

(2) cshtml.cs 文件：其中包含处理页面事件的 C# 代码。

支持文件的名称以下画线开头。例如，_Layout.cshtml 文件可配置所有页面通用的 UI 元素。此文件设置页面顶部的导航菜单和页面底部的版权声明。

2. wwwroot 文件夹

wwwroot 文件夹包含静态文件，如 HTML 文件、JavaScript 文件和 CSS 文件。

3. appSettings.json

此文件包含项目配置信息，如连接字符串。

4. Program.cs

此文件包含程序的入口点。

5. Startup.cs

此文件包含配置应用行为的代码。

5.2 创建 Razor Pages

【例 5-1】 创建 Razor Pages Web 项目。

① 打开 Visual Studio 2019 应用程序，选择“创建新项目”选项，如图 5-1 所示。

图 5-1 创建新项目

② 选择“ASP.NET Core Web 应用程序”选项，单击“下一步”按钮，如图 5-2 所示。

图 5-2 选择 ASP.NET Core Web 应用程序

③ 在“配置新项目”对话框中输入项目名称 RazorPagesdemo，单击“创建”按钮，如图 5-3 所示。

图 5-3　配置新项目

④ 在“创建新的 ASP.NET Core Web 应用程序”对话框中依次在下拉列表中选择 .NET Core 和 ASP.NET Core 3.1 选项，同时选择“Web 应用程序”选项，然后单击“创建”按钮，如图 5-4 所示。

图 5-4　选择“Web 应用程序”选项

⑤ 项目创建完成，如图 5-5 所示。

⑥ 单击 IIS Express 按钮运行程序，结果如图 5-6 所示。

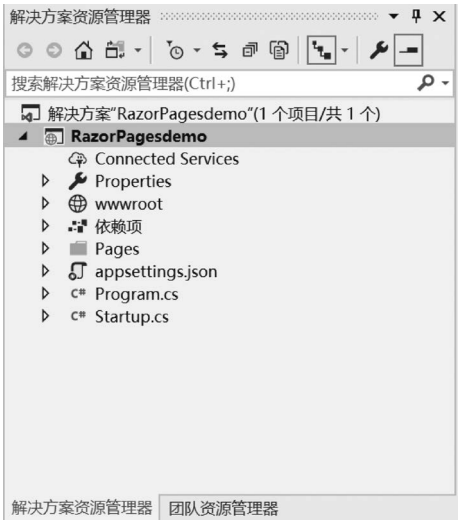

图 5-5 项目创建完成

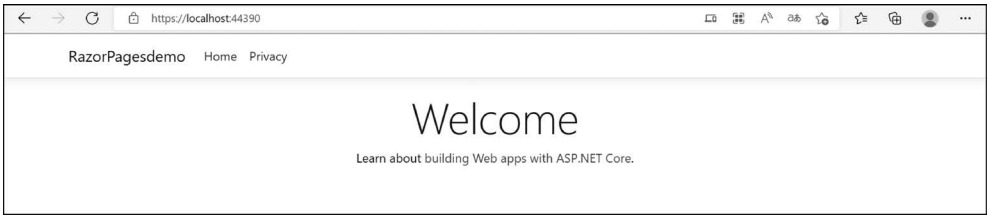

图 5-6 应用程序运行结果

5.3 Razor 基本语法

5.3.1 Razor 语法

Razor 是一种将服务器端的代码嵌入客户端网页中的标记语法。Razor 语法由 Razor 标记、C＃语言和 HTML 组成，Razor 文件的扩展名为.cshtml。Razor 默认的语言为 HTML，同时支持 C＃和 Visual Basic，可以使用@符号从 HTML 切换到 C＃。Razor C＃的内联表达式(变量和函数)以@开始，代码语句用分号结束。变量使用 var 关键字声明，字符串用引号括起来。

5.3.2 Razor 表达式

Razor 能够对 C＃表达式进行运算并渲染为 HTML 输出。当@符号后跟 Razor 保留关键字时为 Razor 特定标记，否则会表示为 C＃表达式。

若要对@标记中的符号进行转义，即在 HTML 中显示@，则需使用第二个@符号，例如：

```
<p>@@wswk2001@sina.com</p>
```

该代码在 HTML 中呈现单个@符号：

```
<p>@wswk2001@sina.com</p>
```

Razor 可以避免由于 HTML 内容中包含邮件地址@符号，而误将@符号处理为转义字符。以下示例中的电子邮件地址将通过分析而保持不变：

```
<a href="mailto:wswk2001@sina.com">wswk2001@sina.com</a>
```

1. 隐式 Razor 表达式

隐式 Razor 表达式以@开始，后面为 C# 代码：

```
<p>现在时间是:@DateTime.Now</p>
<p>今年是否为闰年:@DateTime.IsLeapYear(2022)</p>
```

2. 显式 Razor 表达式

显式 Razor 表达式由@带一对小括号组成。若要显示上个月的时间，则可以使用以下标记：

```
<p>上月的时间: @(DateTime.Now-TimeSpan.FromDays(30))</p>
```

Razor 将计算@后的小括号中的所有内容，并将其显示输出。

【例 5-2】 Razor 表达式。

① 创建 RazorPagedemo 项目后，打开 Pages 文件夹下的 Index.cshtml 页面，输入 Razor 表达式内容，代码如下。

```
<div class="text-center">
        <p>@@wswk2001@sina.com</p>
        <p>现在时间是:@DateTime.Now</p>
        <p>今年是否为闰年:@DateTime.IsLeapYear(2022)</p>
        <p>上月的时间: @(DateTime.Now - TimeSpan.FromDays(30))</p>
</div>
```

② 运行该程序，结果如图 5-7 所示。

图 5-7　Razor 表达式运行结果

隐式表达式通常不能包含空格，@后面必须紧跟有效的标识符或关键字、“(”或“{”等。

③ 在本例中再添加一行代码，如下。

```
<p>上月时间: @DateTime.Now - TimeSpan.FromDays(30)</p>
```

运行后结果如图 5-8 所示。

可以看出，因为去掉了括号，代码只输出了@DateTime.Now 表达式的值，@DateTime.Now 后面的语句原样输出了。

3. 表达式编码

C# 表达式计算结果是字符串时会采用 HTML 编码；如果为 IHtmlContent，则通过

图 5-8 结果对比

IHtmlContent.WriteTo 渲染到页面；如果不是 IHtmlContent，则通过 ToString 转换为字符串并在渲染前进行编码。

【例 5-3】 表达式编码。

① 新建 RazorPagedemo 项目，打开 Pages 文件夹下的 Index.cshtml 页面，输入如下代码。

```
<div class="text-center">
    @("<span>Razor Pages 是一种新型的 Web 框架</span>")
</div>
```

运行后显示结果如图 5-9 所示。

图 5-9 表达式编码运行结果

② 如果想以 HTML 标记进行输出，则需要加上 Html.Raw()方法，将上面代码修改如下。

```
<div class="text-center">
    @Html.Raw("<span>Razor Pages 是一种新型的 Web 框架</span>")
</div>
```

再次运行该程序，结果如图 5-10 所示。

图 5-10 使用 Html.Raw()方法运行结果

5.3.3 Razor 代码块

1. 代码块定义

Razor 代码块包含在@{…}中，代码块内的 C# 代码不会渲染到页面中。与表达式不同，Razor 页面中的代码块和表达式将共享同一个作用域并按顺序定义。

【例 5-4】 Razor 代码块。

① 创建 RazorPagedemo 项目，打开 Pages 文件夹下的 Index.cshtml 页面，输入 Razor

代码块内容，如下所示。

```
<div class="text-center">
    @{
        var quote = "课程名称: C++程序设计";
    }
        <p>@quote</p>
    @{
        quote = "课程名称:ASP.NET Core 程序设计";
    }
    <p>@quote</p>
</div>
```

② 运行该程序，结果如图 5-11 所示。

图 5-11　Razor 代码块运行结果

③ 在代码块中，可以使用标记将本地函数声明为模板化方法，代码如下所示。

```
<div class="text-center">
@{
    void RenderName(string name)
    {
        <p>姓名:<strong>@name</strong></p>
    }
    RenderName("李志伟");
    RenderName("赵宏杰");
}
</div>
```

④ 运行该程序，结果如图 5-12 所示。

图 5-12　函数模板化运行结果

2. 隐式转换

代码块中的默认语言是 C#，但 Razor 可随时将页面切换到 HTML，并且代码块内 HTML 标记能够被正确渲染和执行。

【例 5-5】 隐式转换。

① 创建 RazorPagedemo 项目，打开 Pages 文件夹下的 Index.cshtml 页面，输入如下代码。

```
<div class="text-center">
    @{
        var inScore = "A+";
        <p>王洪明同学在本次考试中的成绩为 @inScore</p>
    }
</div>
```

② 运行该程序，结果如图 5-13 所示。

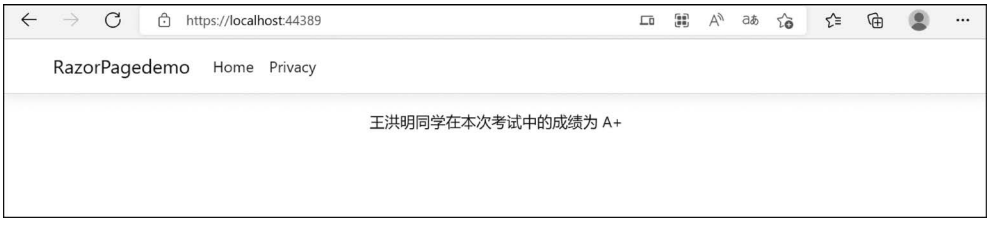

图 5-13 Razor 隐式转换运行结果

3. 带分隔符的显式转换

如果要在代码块中定义需要渲染的 HTML 的子区域，应用 Razor <text>标记包含要渲染的字符。使用此方法可渲染未使用 HTML 标记的 HTML 内容，用于在渲染内容时控制空格。此语句仅渲染标记之间的内容，标记之前或之后的空格不会显示在 HTML 中。

【例 5-6】 带分隔符的显式转换。

① 创建 RazorPagedemo 项目，打开 Pages 文件夹下的 Index.cshtml 页面，输入如下代码。

```
<div class="text-center">
        @{
            var tree1 = "杨树";
            var tree2 = "柳树";
            var tree3 = "槐树";
        }
        <text>北方常见的树种有:@tree1 @tree2 @tree3</text>
</div>
```

② 运行该程序，结果如图 5-14 所示。

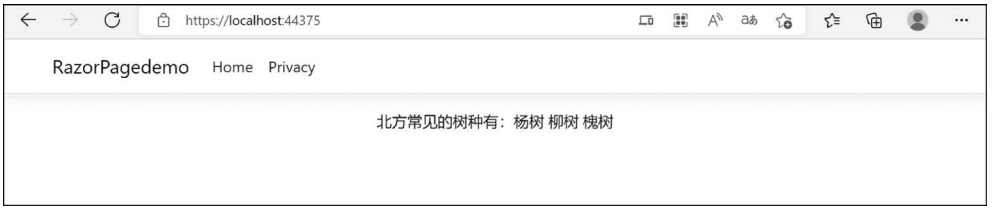

图 5-14 带分隔符的显式转换运行结果

5.3.4 Razor 控制语句

控制语句是对代码块的扩展，用于对程序流程的控制。

1. 条件结构

条件语句用到的命令包括 @if、else if、else 和@switch。

几种常见的条件结构如下。

1）标准 if 结构

语法如下：

```
@if(表达式)
    {
        代码块 1;
    }else
        {
        代码块 2;
        }
```

当表达式的值为真时，执行代码块 1 中的语句，否则执行代码块 2 中的语句。

2）嵌套 if 结构

语法如下：

```
@if(表达式 1)
    {
        代码块 1;
    }else if (表达式 2)
        {
            代码块 2;
        }
        else
        {
            代码块 3;
        }
```

当表达式 1 的值为真时执行代码块 1 中的语句，否则继续判断表达式 2 的值，当表达式 2 的值为真时执行代码块 2，否则执行代码块 3。

3）switch 结构

语法如下：

```
@switch (表达式)
    {
        case 1:表达式 1;break;
        case 2:表达式 2;break;
        ...
        case n:表达式 n;break;
        default:表达式 n+1;break;
    }
```

首先计算 switch 后面表达式的值，然后分别与 case 语句后面的表达式的值进行比较，如果相同则执行该 case 后面的语句，如果都不相同则执行默认的 default 后面的语句。该语句适合于三种及三种以上的多分支情况。

【例 5-7】 已知学生成绩，当成绩大于或等于 85 分时为优秀，当成绩大于或等于 60 分并且成绩小于 85 分时为合格，当成绩小于 60 分时为不合格，用 if 和 switch 两种方法编写代码并输出结果。

① 创建 RazorPagedemo 项目，打开 Pages 文件夹下的 Index.cshtml 页面，输入如下代码。

```
<div class="text-center">
        @{
            var cj = 85;
        }
        @if (cj < 60)
        {
            <p>该同学成绩为"不合格"</p>
        }
        else if (cj >= 60 && cj < 85)
        {
            <p>该同学成绩为"合格"</p>
        }
        else
        {
            <p>该同学成绩为"优秀"</p>
        }

        @switch (cj / 10)
        {
            case 10:
            case 9:
            case 8:<p>该同学成绩为"优秀"</p>; break;
            case 7:
            case 6:<p>该同学成绩为"合格"</p>; break;
            default: <p>该同学成绩为"不合格"</p>;break;
        }
</div>
```

② 运行该程序，结果如图 5-15 所示。

图 5-15 条件语句运行结果

2. 循环结构

循环结构用到的命令包括@for、@foreach、@while 和@do…while。

几种常见的循环结构如下。

1）for 结构

语法如下：

```
@for([初始化表达式];[条件表达式];[迭代表达式])
{
    语句块;
}
```

该结构语句运行时首先执行初始化表达式且只执行一次，接着执行条件表达式，如果为

真则执行语句块，语句块执行结束后执行迭代表达式，然后再回到条件表达式进行判断。如果为假则结束 for 循环。

2）foreach 结构

语法如下：

```
@foreach(类型 变量名 in 集合名)
{
    语句块;
}
```

该结构语句可以遍历集合中的所有元素。每次进行循环遍历时该语句就会从集合中取出一个新的元素值放到只读变量中去，然后执行语句块。当集合中的所有元素都已经被访问到，整个表达式的值即为假值，控制流程就会从 foreach 中退出。

3）while 结构

语法如下：

```
@while(条件表达式)
{
    语句块;
}
```

该结构语句首先判断条件表达式的值，如果该值为真则执行语句块，否则退出该 while 循环。

4）do…while 结构

语法如下：

```
@do
{
    语句块;
} while (条件表达式);
```

该结构语句首先执行语句块中的语句，然后判断条件表达式的值，如果该值为真则继续执行语句块，否则退出该 while 循环。

【例 5-8】 使用 for 结构语句求 1～100 中所有素数之和。

① 创建 RazorPagedemo 项目，打开 Pages 文件夹下的 Index.cshtml 页面，输入如下代码。

```
<div class="text-center">
        @{
            var sum = 0;
            var flag = true;
        }
        @for (var i = 2; i <= 100; i++)
        {
            flag = true;
            @for (var j = 2; j < i - 1; j++)
            {
                if (i % j == 0)
                {
                    flag = false;
                    break;
```

```
            }
        }
        if (flag)
        {
            sum = sum + i;
        }
    }
    <p>1～100 中素数的和为 @sum </p>
</div>
```

② 运行该程序，结果如图 5-16 所示。

图 5-16 for 循环语句运行结果

5.3.5 Razor 复合语句

在 C＃中，using 语句用于确保对象被正确释放。在 Razor 中，可使用相同的机制来创建包含附加内容的 HTML 帮助程序。

【例 5-9】 Razor 复合语句。

① 创建 RazorPagedemo 项目，打开 Pages 文件夹下的 Index.cshtml 页面，输入如下代码。

```
<div class="text-center">
    @using (Html.BeginForm())
    {
        <div>
            <p>
                用户名:<input type="text" id="user" value="">
            </p>
            <p>
                密 码:<input type="text" id="password" value="">
            </p>
            <p>
                <button>登录</button>
            </p>
        </div>
    }
</div>
```

在上面的代码中，HTML 帮助程序使用@using 语句呈现< form >标记。

② 运行该程序，结果如图 5-17 所示。

5.3.6 Razor 异常处理

Razor 异常处理与 C＃语法相似，使用@前缀声明即可。语法如下。

```
@try
```

图 5-17　复合语句运行结果

```
    {
        //可能导致异常的语句
    }
    catch (Exception ex)
    {
        //处理异常的语句
    }
    finally
    {
        //清理代码
}
```

【例 5-10】 Razor 异常处理。

① 创建 RazorPagedemo 项目，打开 Pages 文件夹下的 Index.cshtml 页面，输入如下代码。

```
<div class="text-center">
    @try
    {
            throw new InvalidOperationException("异常操作");
        }
        catch (Exception ex)
        {
            <p>出现错误的信息: @ex.Message</p>
        }
        finally
        {
            <p>执行清理代码</p>
        }
</div>
```

② 运行该程序，结果如图 5-18 所示。

图 5-18　异常语句运行结果

5.3.7　Razor 注释语句

(1) Razor 支持 C# 和 HTML 语法的注释，如符合 C# 语法规定的单行和多行注释

方式：

```
@{
    // <p>单行注释</p>
    /* <p>
        多行
        注释
    </p>
    */
}
```

符合 HTML 语法规定的注释方式：

```
<!--HTML 语句注释-->
```

(2) Razor 自身特有的注释。

语法格式为：

```
@*注释的内容*@
```

该注释支持单行和多行，示例如下。

```
@{
    @* <p>单行注释</p> *@
    @* <p>
            多行
            注释
        </p>
    *@
}
```

5.4 Razor 页面实现

在本节中将通过一个图书管理的例子来说明如何创建基于 Razor 页面添加、修改和删除页面的实现过程。

5.4.1 添加模型

对于简单的 Razor 应用只要在文档中加入@page 指令，并将其放在 Pages 目录下就可以通过 URL 进行访问。如果涉及复杂的业务逻辑就需要创建页面模型类编写独立的代码，使得视图和代码分离。页面模型类需要从 Page Model 类派生，通过约定的方法与视图交互，页面视图通过 handler 的路由参数调用这些方法，方法的命名规则如下。

```
On<HTTP method><handler name>[Async]
```

- 方法以 On 开始。
- HTTP method 参数为 GET、POST、DELETE 等。
- handler name 为方法的正式名称，可以直接作为路由参数 handler 的值。
- Async 表示为异步方法，此项为可选项。

【例 5-11】 建立图书管理页面图书模型。

① 在 Visual Studio 2019 中新建 Razor Pages 项目，项目的名称为 books。

② 在项目名称下新建一个 data 目录，在该目录下新建 Bookinfo 实体类并添加以下代码。

```
public class Bookinfo
    {
        public int bid { get; set; }
        public string bname { get; set; }
        public string ISBN { get; set; }
        public string author { get; set; }
        public string press { get; set; }
        public DateTime pubtime { get; set; }
    }
```

③ 在 data 目录下新建 IBookinfoRepository 接口文件，添加 Get()、Add()、Update()和 Delete()四个方法，代码如下所示。

```
    public interface IBookinfoRepository
    {
        List<Bookinfo> List();
        Bookinfo Get(int id);
        bool Add(Bookinfo bookinfo);
        bool Update(Bookinfo bookinfo);
        bool Delete(int id);
    }
```

④ 在 data 目录下新建 BookinfoRepository 类，实现 IBookinfoRepository 接口，并且使用静态变量保存的数据，模拟数据库中的数据，代码如下所示。

```
public class BookinfoRepository:IBookinfoRepository
    {
        private static List<Bookinfo> books = new List<Bookinfo>{
                    new Bookinfo{ bid=1, bname="ASP.NET 程序设计基础教程", ISBN=
"9787302322108", author="陈长喜", press="清华大学出版社",pubtime=DateTime.Parse("2013-
08-01")},
                    new Bookinfo{ bid=2, bname="ASP.NET 程序设计高级教程", ISBN=
"9787302476351", author="陈长喜", press="清华大学出版社",pubtime=DateTime.Parse("2017-
10-01")}
        };
        public bool Add(Bookinfo bookinfo)
        {
            books.Add(bookinfo);
            return true;
        }
        public bool Delete(int id)
        {
            var book = books.FirstOrDefault(s => s.bid == id);
            if (book != null)
            {
                books.Remove(book);
            }
            return true;
        }
        public Bookinfo Get(int id)
        {
            return books.FirstOrDefault(s => s.bid == id);
```

```
        }
        public List<Bookinfo> List()
        {
            return books;
        }
        public bool Update(Bookinfo bookinfo)
        {
            var book = books.FirstOrDefault(s => s.bid == bookinfo.bid);
            if (book != null)
            {
                books.Remove(book);
            }
            books.Add(bookinfo);
            return true;
        }
    }
```

至此，books 模型文件建立完成，其目录结构如图 5-19 所示。

图 5-19 books 模型文件的目录结构

5.4.2 显示页面

【例 5-12】 在例 5-11 的基础上添加列表图书页面。

① 在项目 Pages 目录下新建 Bookinfo 文件夹。选择 Bookinfo 文件夹，右击，在弹出的快捷菜单中选择“添加”→“Razor 页面”命令，如图 5-20 所示。

② 在“添加已搭建基架的新项”对话框中选择 Razor 页面，单击“添加”按钮，如图 5-21 所示。

③ 在“添加 Razor 页面”对话框中添加 Razor 页面的名称为 List，并在下面的“选项”中选中“生成 PageModel 类”复选框，单击“添加”按钮，如图 5-22 所示。

④ 打开 List.cshtml 页面，填写如下代码。

```
<table class="table">
    <tr>
        <th>序号</th>
        <th>书名</th>
        <th>ISBN</th>
```

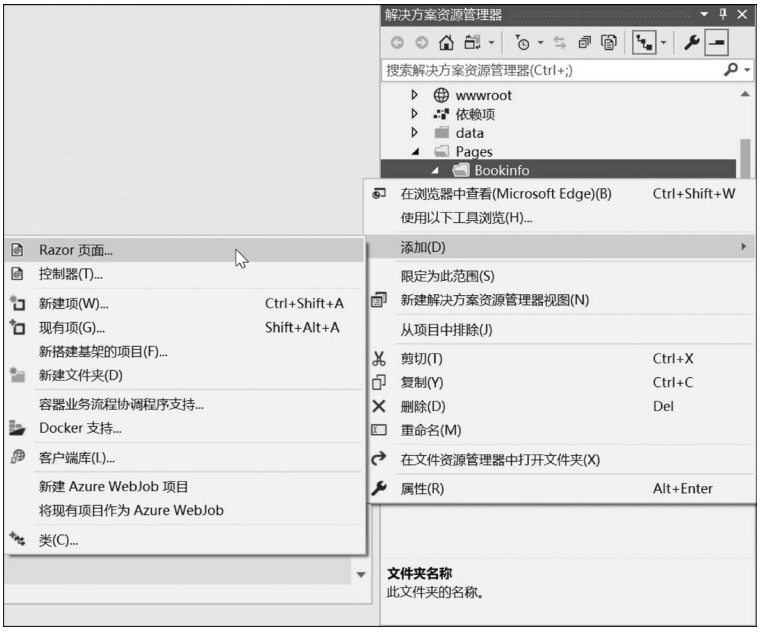

图 5-20　选择“添加”→“Razor 页面”命令

图 5-21　选择 Razor 页面

图 5-22　添加 Razor 页面

```
            <th>作者</th>
            <th>出版社</th>
            <th>出版时间</th>
            <th></th>
        </tr>
        @foreach (var book in Model.bookinfo)
        {
    <tr>
        <td>@book.bid</td>
        <td>@book.bname</td>
        <td>@book.ISBN</td>
        <td>@book.author</td>
        <td>@book.press</td>
        <td>@book.pubtime</td>
        <td>
            <a class="btn btn-primary" asp-page="Add">添加 </a>
            <a class="btn btn-primary" asp-page="Update" asp-route-id="@book.bid">修改</a>
            <a class="btn btn-danger" href="/bookinfo/delete?id=@book.bid">删除</a>
        </td>
    </tr>
        }
    </table>
```

⑤ 打开 List.cshtml.cs 文件，修改 ListModel 类代码，如下所示。

```
public class ListModel : PageModel
    {
        public void OnGet()
        {
            bookinfo = _bookinfoRepository.List();
        }
        private readonly IBookinfoRepository _bookinfoRepository;
        public List<Bookinfo> bookinfo { get; set; }
        public ListModel(IBookinfoRepository bookinfoRepository)
        {
            _bookinfoRepository = bookinfoRepository;
        }
    }
```

⑥ 打开 Startup.cs 文件，在 ConfigureServices()方法中注册 repository，代码如下所示。

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages();
            //注册 repository
            services.AddScoped<IBookinfoRepository, BookinfoRepository>();
        }
```

⑦ 调试、运行该程序，在现有浏览器地址后面输入/bookinfo/list 并按 Enter 键，结果如图 5-23 所示。

List.cshtml.cs 文件即 ListModel 类，该类具备 MVC 中的 Controller 和 Model 的概念，其包含的数据可以用来生成 Razor 视图；包含方法可以用来处理业务逻辑，此方法可以认为是 Controller 中的 Action。

图 5-23　图书信息显示页面

以上示例中，使用 Razor Pages 的 asp-page 属性实现页面跳转，如：

```
<a class="btn btn-primary" asp-page="Add">添加 </a>
```

在 a 元素上添加了 asp-page="Add"，表示单击添加链接会跳转到 Add 页面。

使用 Razor Pages 的 asp-route-id 属性可以完成页面间的参数传递。例如“修改”按钮，需要跳转至 Update 页面并且传递一个 id 参数，代码如下所示。

```
<a class="btn btn-primary" asp-page="Update" asp-route-id="@book.bid">修改</a>
```

5.4.3　添加页面

【例 5-13】 在例 5-12 的基础上添加图书信息页面。

① 打开例 5-12 项目后在 Bookinfo 文件夹中添加 Razor 页面，页面名称为 Add。

② 打开 Add.cshtml 添加页面并填写如下代码。

```
<form method="post">
    <div class="form-group">
        <label>序号</label>
        <input type="number" asp-for="bookinfo.bid" class="form-control" />
    </div>
    <div class="form-group">
        <label>书名</label>
        <input type="text" asp-for="bookinfo.bname" class="form-control" />
    </div>
    <div class="form-group">
        <label>ISBN</label>
        <input type="number" asp-for="bookinfo.ISBN" class="form-control" />
    </div>
    <div class="form-group">
        <label>作者</label>
        <input type="text" asp-for="bookinfo.author" class="form-control" />
    </div>
    <div class="form-group">
        <label>出版社</label>
        <input type="text" asp-for="bookinfo.press" class="form-control" />
    </div>
    <div class="form-group">
        <label>出版时间</label>
        <input type="text" asp-for="bookinfo.pubtime" class="form-control" />
    </div>
    <div class="form-group">
```

```
        <button type="submit" class="btn btn-primary" asp-page-handler="Save">保存
</button>
        <a asp-page="list" class="btn btn-dark">取消</a>
    </div>
</form>
```

③ 打开 Add.cshtml.cs 文件，修改 AddModel 类代码，如下所示。

```
public class AddModel : PageModel
    {
        public void OnGet()
        {
        }
        private readonly IBookinfoRepository _bookinfoRepository;
        public AddModel(IBookinfoRepository bookinfoRepository)
        {
            _bookinfoRepository = bookinfoRepository;
        }
        [BindProperty]
        public Bookinfo bookinfo { get; set; }
        public IActionResult OnPostSave()
        {
            _bookinfoRepository.Add(bookinfo);
            return RedirectToPage("List");
        }
    }
```

④ 调试、运行该程序，在现有浏览器地址后面输入/bookinfo/add 并按 Enter 键，结果如图 5-24 所示。

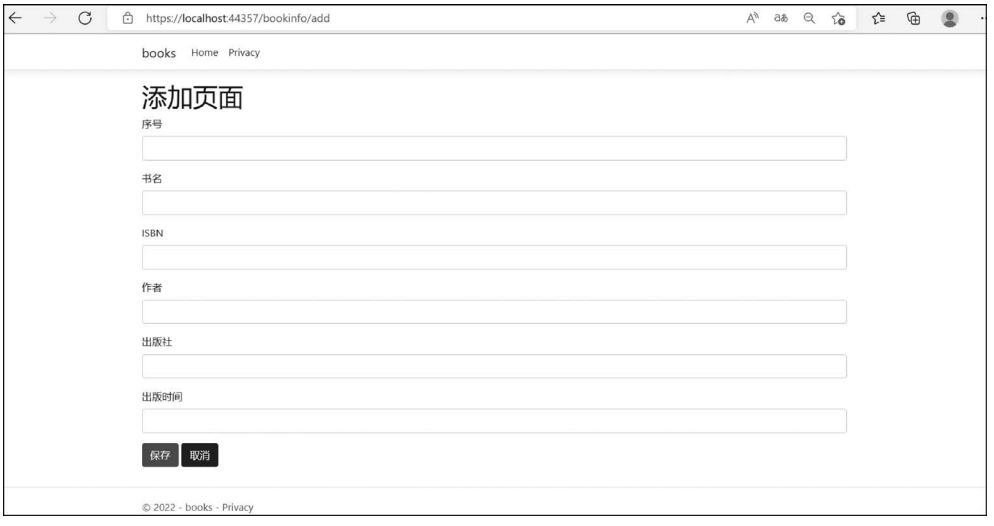

图 5-24 添加图书信息页面

在“添加页面”页面中输入相关数据后单击“保存”按钮，系统将保存当前数据并返回到 List 显示页面。

在 Add.cshtml 页面中通过使用 asp-page-handler="Save"映射模型中的 OnPostSave() 方法。

```
public IActionResult OnPostSave()
        {
            _bookinfoRepository.Add(bookinfo);
            return RedirectToPage("List");
        }
```

添加的图书信息需要从前端传递到后端并进行提取，使用 BindProperty 来完成提交的表单数据与模型属性之间的映射，实现简单的前后端绑定。其代码如下。

```
[BindProperty]
        public Bookinfo bookinfo { get; set; }
```

5.4.4 修改页面

【例 5-14】 在例 5-13 的基础上实现修改图书信息页面。

① 打开例 5-13 项目后在 Bookinfo 文件夹中添加 Razor 页面，页面名称为 Update。

② 打开 Update.cshtml 修改页面并填写如下代码。

```
<form method="post">
    <div class="form-group">
        <label>序号</label>
        <input type="number" asp-for="book.bid" class="form-control" />
    </div>
<div class="form-group">
    <label>书名</label>
    <input type="text" asp-for="book.bname" class="form-control" />
</div>
<div class="form-group">
    <label>ISBN</label>
    <input type="number" asp-for="book.ISBN" class="form-control" />
</div>
<div class="form-group">
        <label>作者</label>
        <input type="text" asp-for="book.author" class="form-control" />
</div>
<div class="form-group">
    <label>出版社</label>
    <input type="text" asp-for="book.press" class="form-control" />
</div>
<div class="form-group">
    <label>出版时间</label>
    <input type="text" asp-for="book.pubtime" class="form-control" />
</div>
<div class="form-group">
        <button type="submit" class="btn btn-primary" asp-page-handler="Edit">保存</button>
        <a asp-page="list" class="btn btn-dark">取消</a>
    </div>
</form>
```

③ 打开 Update.cshtml.cs 文件，修改 UpdateModel 类代码，如下所示。

```
public class UpdateModel : PageModel
{
    [BindProperty]
```

```
        public Bookinfo book { get; set; }
        public void OnGet(int id)
        {
            book = _bookinfoRepository.Get(id);
        }
        private readonly IBookinfoRepository _bookinfoRepository;
        public UpdateModel(IBookinfoRepository bookinfoRepository)
        {
            _bookinfoRepository = bookinfoRepository;
        }
        public IActionResult OnPostEdit()
        {
            _bookinfoRepository.Update(book);
            return RedirectToPage("list");
        }
    }
```

④ 调试、运行该程序，在现有浏览器地址后面输入/bookinfo/list 并按 Enter 键，显示图书列表页面，在该页面中单击序号为 1 的记录中的“修改”按钮跳转到修改页面，结果如图 5-25 所示。

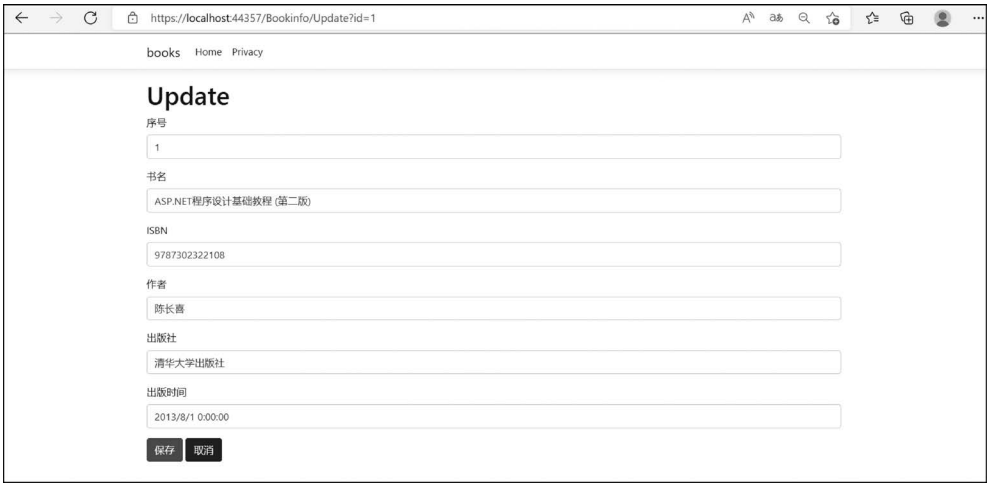

图 5-25 修改图书信息

⑤ 在图书修改页面将书名“ASP. NET 程序设计基础教程”修改为“ASP. NET 程序设计基础教程（第二版）”，然后单击“保存”按钮，程序将返回图书列表页面，此时会看到已完成序号为 1 的记录中的书名被修改，如图 5-26 所示。

5.4.5 删除页面

【例 5-15】 在例 5-14 的基础上实现删除图书信息页面。

① 打开例 5-14 项目后在 Bookinfo 文件夹中添加 Razor 页面，页面名称为 Delete。

② 打开 Delete. cshtml 删除页面并填写如下代码。

```
<h2 class="text-danger">
    确定删除?
</h2>
```

图 5-26　完成图书信息修改

```
<form method="post">
    <div class="form-group">
        序号: @Model.Bookinfo.bid
    </div>
    <div class="form-group">
        书名:@Model.Bookinfo.bname
    </div>
    <div class="form-group">
        ISBN: @Model.Bookinfo.ISBN
    </div>
    <div class="form-group">
        作者: @Model.Bookinfo.author
    </div>
    <div class="form-group">
        出版社: @Model.Bookinfo.press
    </div>
    <div class="form-group">
        出版时间: @Model.Bookinfo.pubtime
    </div>
    <div class="form-group">
        <button type="submit" class="btn btn-primary" asp-page-handler="Delete" asp-route-id
="@Model.Bookinfo.bid">删除</button>
        <a asp-page="list" class="btn btn-dark">取消</a>
    </div>
</form>
```

③ 打开 Delete.cshtml.cs 文件，修改 DeleteModel 类代码如下。

```
public class DeleteModel : PageModel
    {
        public Bookinfo Bookinfo { get; set; }
        public void OnGet(int id)
        {
            Bookinfo = _bookinfoRepository.Get(id);
        }
        private readonly IBookinfoRepository _bookinfoRepository;
        public List<Bookinfo> bookinfo { get; set; }
        public DeleteModel(IBookinfoRepository bookinfoRepository)
        {
            _bookinfoRepository = bookinfoRepository;
        }
        public IActionResult OnPostDelete(int id)
        {
```

```
            _bookinfoRepository.Delete(id);
            return RedirectToPage("list");
        }
    }
```

④ 调试、运行该程序，在现有浏览器地址后面输入/bookinfo/list 并按 Enter 键，显示图书列表页面，在该页面中单击序号为 1 的记录中的“删除”按钮跳转到删除页面，结果如图 5-27 所示。

图 5-27 删除图书信息页面

单击“删除”按钮将删除当前记录并返回列表页。

5.5 Razor 中的布局

5.5.1 布局文件

大多数 Web 应用都具有一个通用布局，在页面间切换时为用户提供一致性体验。该布局通常包括应用导航或菜单元素以及页脚等常见的用户界面元素。应用中的页面也经常使用常见的 HTML 结构，如脚本和样式表。所有这些共享元素都可以在布局文件中定义，然后在 Web 应用内使用的视图都可以引用该布局文件，减少视图中的重复代码。项目解决方案中可以定义多个布局，其中不同的视图可以指定不同的布局。

Razor Pages 应用程序的默认布局文件名为 _Layout.cshtml，该文件存在于项目根目录中 Pages/Shared 文件夹下，如图 5-28 所示。

图 5-28 Razor 布局文件

5.5.2 布局规则

1. 指定布局

指定的布局可以使用完整路径和部分名称。例如/Pages/Shared/_Layout.cshtml 或_Layout。使用部分名称时，Razor 视图引擎首先搜索处理程序方法（或控制器）所在的文件夹，然后搜索 Shared 文件夹。

HTML 可以将某个位置指定的布局模板应用到整个项目的多个页面中。布局模板中会使用标签 @RenderBody()和@RenderSection()，RenderBody 是一个占位符，它可以显示子页面的全部内容，其代码如下所示。

```
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>
```

布局可以通过调用 RenderSection 来选择引用一个或多个节。节通过方法来组织某些页面元素放置的位置，即在母版页中占一个位置来解决页面不同布局的问题。每次调用 RenderSection()时可通过 required 指定该部分是必需还是可选，其中 required 默认为 true，表示引用这个布局页的所有 View 必须含有该 Section，若设为 false 则表示可以有也可以没有，示例代码如下。

```
<script type="text/javascript" src="~/scripts/global.js"></script>
@RenderSection("Scripts", required: false)
```

2. 导入共享指令

在视图和页面中可以使用 Razor 指令导入命名空间。由多个视图共享的操作可以在_ViewImports.cshtml 文件中进行指定，代码如下所示。

```
@using layout
@namespace layout.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

_ViewImports 文件不支持函数和节定义，但是支持以下指令：@addTagHelper、@removeTagHelper、@tagHelperPrefix、@using、@model、@inherits、@inject，代码如下。

```
@using WebApplication1
@using WebApplication1.Models
@using WebApplication1.Models.AccountViewModels
@using WebApplication1.Models.ManageViewModels
@using Microsoft.AspNetCore.Identity
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

_ViewImports.cshtml 文件一般放在 Pages 或 Views 文件夹中。该文件只能应用于所在文件夹及其子文件夹中的页面或视图。如果在文件层次结构中找到多个_ViewImports.cshtml 文件，则指令的规则如下。

@addTagHelepr、@removeTagHelper：按顺序全部运行；

@tagHelperPrefix：距离视图最近的 tagHelperPrefix 会覆盖任何其他 tagHelperPrefix；

@model：距离视图最近的 model 会覆盖任何其他 model；

@inherits：距离视图最近的 inherits 会覆盖任何其他 inherits；

@using：全部包括在内，忽略重复项；

@inject：针对每个属性，最接近视图的属性会替代具有相同属性名的其他属性。

3. 先于视图文件之前执行

_ViewStart.cshtml 会在所有视图被执行之前运行，如一些不便或不能在母版页中进行的统一操作可以将这些代码置于_ViewStart.cshtml 文件中。通常这些应用级别版本的文件应直接放置在 /Pages(或/Views)文件夹中。

_ViewStart.cshtml 与_ViewImports.cshtml 文件类似，也采用分层结构。如果在 View 的某个目录下(例如 Home 目录)存在一个同名的_ViewStart.cshtml 文件，那么这个_ViewStart 文件也会被调用，但最先调用根目录下的_ViewStart 文件，然后才是 Home 目录下的_ViewStart 文件。

5.5.3 使用布局

【例 5-16】 使用布局。

① 打开例 5-15 项目，选择 Pages/Shared 目录，右击，在弹出的快捷菜单中选择“添加”→“新建项”命令，如图 5-29 所示。

图 5-29 添加新建项

② 选择“Razor 布局”选项，文件名称默认为_Layout1.cshtml，单击“添加”按钮，如图 5-30 所示。

③ 修改_Layout1.cshtml，代码如下。

```
<html>
    <head>
        <meta name="viewport" content="width=device-width" />
```

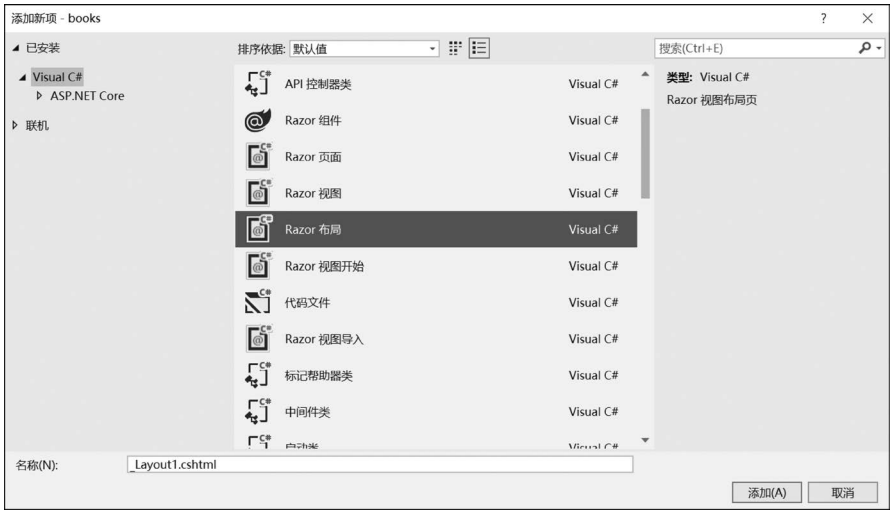

图 5-30　添加 Razor 布局

```
        <title>@ViewBag.Title</title>
    </head>
<body>
        <h1>图书信息</h1>
        <div id="mainDiv">
            @RenderBody()
        </div>
</body>
</html>
```

④ 设置所有页面使用_Layout1 布局。打开_ViewStart.cshtml 文件，修改 Layout 并设置为_Layout1。代码如下：

```
@{
    Layout = "_Layout1";
}
```

⑤ 调试、运行该程序，在现有浏览器地址后面输入/bookinfo/list 并按 Enter 键，显示图书列表页面，如图 5-31 所示。

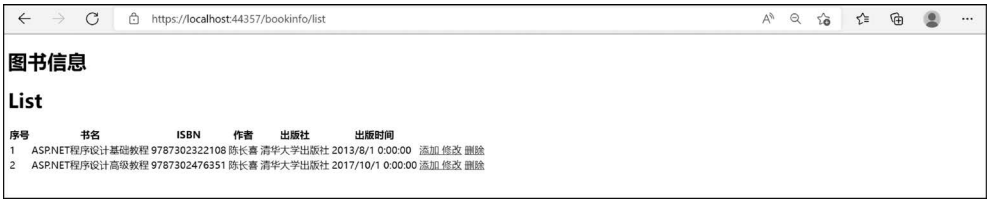

图 5-31　列表页面

⑥ 单击“添加”“修改”“删除”链接后结果如图 5-32～图 5-34 所示。

⑦ 如果只有项目列表页面使用_Layout1 布局，打开 List.cshtml 文件，添加代码如下。

```
@{
    ViewData["Title"] = "List";
    Layout = "_Layout1";
}
```

图 5-32 添加页面

图 5-33 修改页面

图 5-34 删除页面

⑧ 调试、运行该程序,发现只有列表页面布局发生了改变,其他页面并没有变化。读者可自行调试、运行。

小 结

本章首先介绍了 Razor Pages 的基本概念,通过一个简单的示例介绍了 Razor Pages 应用程序的建立过程,然后介绍了 Razor 基本语法的使用,并通过一个示例介绍了 Razor Pages 中数据模型的建立及页面的添加、修改和删除过程,最后介绍了布局的概念及其应用。本章应重点掌握 Razor Pages 应用程序的建立。

习 题

一、作业题

1. 简述什么是 Razor Pages 技术。
2. ASP.NET Core Web Application 项目文件包含哪几部分?
3. ASP.NET 的布局方式有哪几种?
4. 简述 Razor 语法的特点。

二、上机实践题

使用 Razor Pages 技术创建 ASP. NET Core Web 应用程序，可以显示学生个人信息，包含学生的学号、姓名、性别、家庭地址和联系方式，实现对学生个人信息的显示、添加、修改和删除，结果如图 5-35～图 5-38 所示。

图 5-35　显示学生信息

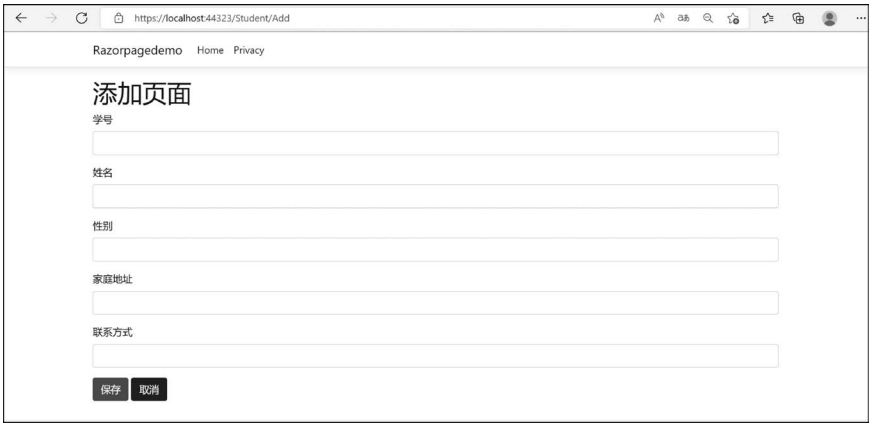

图 5-36　添加学生信息

图 5-37　修改学生信息


https://localhost:44323/student/delete?id=1
Razorpagedemo
Home
Privacy
删除页面
确定删除?
学号: 1
姓名:张新
性别: 男
家庭地址: 北京市
联系方式: 010-12345678
删除
取消

图 5-38 删除学生信息