目录

客户端技术编程

一、主要完成内容

1. 使用网页模板进行自定义设计和填充对应内容
2. 添加日间模式和夜间模式切换功能
3. 添加网站部分小功能，例如返回顶部和小部件缓慢切出等功能

二、主要功能实现

网站整体布局

1. 整体采用垂直布局的方式，从上至下依次是顶部导航栏、主要图片和内容的轮播图、主要信息部分、主要内容部分、特征部分、图片轮播图部分、视频展示部分、底部信息部分
2. 在整体下面，每个子部分使用宽度撑满的策略，使其能够依次排列

顶部菜单模块

1. 菜单使用弹性盒模型 `display: flex`，使得菜单排列在一行，并且控制每个的标签的 `padding` 属性，使其整齐排列
2. 网站名称使用行内盒模型 `display: inline-block;`，并设置 `white-space: nowrap;`，使其不换行，配合 `padding` 完成位置布局

展示图片的 gallery 模块

1. 通过 JavaScript 调整其图片的 `class` 为 `owl-item cloned active` 或 `owl-item cloned`，配合类选择器选择 `active` 来设置样式
2. 动画效果通过设置 `transform` 和 `transition` 来实现动画效果

三、额外功能实现

日间模式和夜间模式切换

1. 简要描述实现过程
    1. 在右下角设置一个切换按钮，样式设置成太阳和月亮的切换，太阳代表切换成日间模式，月亮代表切换成夜间模式
    2. 使用 JavaScript 获取到此时的模式，并设置在点击的时候实现切换功能
    3. 如果切换成夜间模式，主要将背景切换成黑色，并且设置一定的不透明度，再将字体设置为白色，形成一个高对比，然后将切换按钮的图片修改为太阳图标

4. 如果切换成日间模式，主要将样式修改成默认样式，再将切换按钮的图片修改为月亮图标

2. HTML 部分

```
<section class="right-nav" id="right-nav">
  <div class="black" id="black">
        <img src="img/night.png" alt="夜间模式">
  </div>
</section>
```

3. CSS 部分

```
.right-nav .black img{
    border-radius: 50%;
    width: 30px;
    height: 30px;
}
```

4. JavaScript 部分

    1. 点击事件

    ```
    $("#black").click(function () {
        //功能实现...
    }
    ```

    2. 获取模式信息

    ```
    //获取对应模式信息，夜间模式或日间模式
    var s = $("[alt$='间模式']").attr("alt");
    ```

    3. 设置对应模式样式

        1. 夜间模式

        ```
        $('body').css("background-color", "black") //设置背景颜色为黑色
        $('body').css("opacity", "0.7") //设置不透明度
        $('body').css("-moz-opacity", "0.7")
        $('body').css("filter", "alpha(opacity=50)") //设置滤镜
        $('body').append("<style>::selection {background-color: #d8d5da;color: #fff;}</style>") //设置选中的背景样式
        $('.mb-10,.pb-30,p,li,a,span.lnr').css("color", "white") //给对应字体设置颜色
        $('span.lnr-arrow-up,span.lnr-arrow-down').css("color", "#222") //给箭头设置颜色
        $('.project-area,.info-content,.single-feature,.gallery-area').css("background-color", "black") //给对应模块设
        ```

```
置背景样式
$('.header-btn').css("background-color", "transparent")
 //设置对应按钮背景颜色透明
$('.info-content').append("<style>.info-area .info-cont
ent::after{background-color: black}</style>") //给对应元
素添加样式
$('.active-works-carousel .owl-dots .owl-dot.active').c
ss("background", "#777777;") //设置背景颜色
```

2. 日间模式

```
$('body').css("background-color", "");  //设置为默认背景
颜色
$('body').css("opacity", "")    //取消不透明
$('body').css("-moz-opacity", "")
$('body').css("filter", "") //取消滤镜
//给对应部件设置样式
$('body').append("<style>::selection {background-color:
 #b01afe;color: #fff;}</style>")
$('.mb-10,.pb-30,p,li,a,span.lnr').css("color", "")
$('.project-area,.info-content,.single-feature,.gallery
-area').css("background-color", "")
$('.header-btn').css("background-color", "white")
$('.info-content').append("<style>.info-area .info-cont
ent::after{background-color: white}</style>")
```

4. 切换功能

    1. 夜间模式

    ```
    $("[alt='夜间模式']").attr({
        "src": "img/sum.png",
        "alt": "日间模式"
    });
    ```

    2. 日间模式

    ```
    $("[alt='日间模式']").attr({
        "src": "img/night.png",
        "alt": "夜间模式"
    });
    ```

缓慢切入

1. 简要描述实现过程

    1. 使用选择器，选择到对应的元素后，设置 `transition: 5s`，将如果某个属性发生改变，则在 5s 内缓慢改变

2. 通过 JavaScript 实现在加载的时候，修改这个元素的位置，配合 css 的设置，则实现了缓慢切入的动画

2. css 部分

```
.right-nav {
    position: fixed;
    bottom: 20px;
    right: -30px;
    z-index: 999;
    transition: 5s; //设置 transition 后，right 从-30px 变化到 20px 的过程总共花费 5s
    cursor: pointer;
}
```

3. Javascript 部分

```
window.onload = function () {
    document.getElementById("right-nav").style.right = "20px";
}
```

四、主要结果截图

1. 日间模式和夜间模式的切换

    1. 夜间模式

2. 日间模式

2. 右下角的功能按钮

3. 整体实现效果

JavaScript 编程

一、主要完成内容

1. 随机生成 4 位的字母和数字组合的验证码

2. 实现通过滑块进行图片验证

二、主要功能实现

随机产生一个四位验证码

1. makeCode()：随机产生一个四位验证码的主要函数

```
//产生一个四位验证码
function makeCode() {
    string = "" //初始验证码为空
    var code = document.getElementById("code-main") //DOM 获取元素
    code.onclick = makeCode //设置 code 点击事件
    refresh.onclick = makeCode  //设置 refresh 点击事件
    for(let i = 0; i < 4; ++i) {     //总共产生 4 位
        string += createChar()  //每次添加一个随机位
    }
    code.innerHTML = string //修改 HTML 数据
}
```

2. createChar()：随机产生一个字符 a-z,A-Z,0-9

```
function createChar() {
    var num = Math.round(Math.random()) == 0     //随机确定是字母还是
数字
    if(num == true) {    //如果是数字，则返回 0-9
        return Math.round(Math.random()*9)
    }
    else {  //如果是字母
        var char =  String.fromCharCode(65 + Math.round(Math.random
() * 25))     //随机产生一个字母
        if(Math.round(Math.random()) == 0) {     //确定大小写
            return char.toLowerCase()   //返回小写
        }
        return char //返回大写
    }
}
```

验证四位验证码是否正确

1. testCode()：进行验证码验证

```
function testCode() {
    // console.log("success")
```

```
        console.log(char_code.value)
        if (char_code.value == string) {    //如果输入与随机产生相同
            alert("字符验证码验证成功！")
        } else {    //如果不同
            alert("字符验证码验证失败！")
        }
        location.reload()
    }
```

随机产生一个图片缺口

1. selectSection()：随机挑选图片中的一部分

```
    function selectSection() {
        var img = document.getElementById("img")    //DOM 获取 id 为 img
    的元素
        var height_string = img.getAttribute("height")  //获取元素的高度
        var img_height = height_string.substring(0, height_string.index
    Of("px"))    //获取实际像素值
        var img_url = img.getAttribute("src")   //获取图片地址

        //设置成功位置的位置
        success_section.style.top = success_position_y + "px"
        success_section.style.left = success_position_x + "px"
        //进行按钮位置微调
        section.style.top = success_position_y - img_height - 6 + "px"
        //获取图片缺省部分
        section.style.background = "url(" + img_url + ") -" + success_p
    osition_x + "px -" + success_position_y + "px/auto " + img_height +
     "px"
    }
```

随机产生图片

1. randomImage()：随机产生图片

```
    function randomImage() {
        var img = document.getElementById("img")    //获取图片元素
        var num = Math.ceil(Math.random()*5) + 1;   //随机产生图片序号
        img_num = (img_num + num)%5 + 2 //适配本地图片序号
        var img_src = "./img/img"+ img_num +".png"  //生成对应地址
        img.setAttribute("src",img_src) //修改图片
    }
```

通过滑块进行图片验证

1. buttonDown(e)：图片 button 点击事件

```
    function buttonDown(e) {
        //清除样式
        button.style.transition = ""
```

```
    background.style.transition = ""

    //获取button 开始位置
    var e = e || window.event || e.which;
    start_button_position = e.clientX

    document.onmousemove = buttonMove    //设置鼠标移动事件
    document.onmouseup = buttonUp    //设置鼠标松开事件
}
```

2. buttonMove(e)：图片 button 移动事件

```
function buttonMove(e) {
    var button_move = e.clientX //获取移动的位置
    //获取移动的长度
    move_length = moveLength(button_move - start_button_position, 0,
 max_length)

    //修改移动过程中的样式
    button.style.left = move_length + "px"
    background.style.width = move_length + 20 + "px"
}
```

3. buttonUp()：图片 button 鼠标松开

```
function buttonUp() {
    var abs = Math.abs(move_length - success_length)    //获取移动的
距离和需要移动的距离的差值
    if (abs >= 5) { //如果超过5px 的偏差，则失败，重新调整按钮位置
        button.style.left = ""
        background.style.width = ""
        //控制其缓慢移动
        button.style.transition = "left 0.8s linear"
        background.style.transition = "width 0.8s linear"
    } else {    //否则就验证成功
        successMove()
    }
    //取消事件，重新开始
    document.onmousemove = null
    document.onmouseup = null
}
```

4. moveLength(length, min, max)：返回一个符合规范的移动长度

```
function moveLength(length, min, max) {
    if (length < min) { //如果小于最低要求，则返回最低要求
        return min
    } else if (length > max) {  //如果超过最高要求，则返回最高要求
```

```
            return max
        }
        return length   //正常返回
    }
```

5. successMove()：成功对齐执行函数

```
function successMove() {
    success = true  //设置状态为true
    text.innerHTML = "解锁成功" //修改HTML 内容
    background.style.backgroundColor = "lightgreen" //修改背景颜色
    // icon.className = "fas fa-thumbs-up"
    icon.className = "icon iconfont iconicon_xuanzhong" //更换图标
    icon.style.color = "green"  //设置颜色
    //取消对应事件
    button.onmousedown = null
    document.onmousemove = null

    //延时100ms 后弹出结果，并在确认后刷新
    setTimeout(function() {
        alert("图片解锁成功！")
        location.reload()
    }, 100)
}
```

三、额外功能实现

自动获取焦点

1. 简要描述：设置输入框的 autofocus 属性为 autofocus

2. 代码实现

```
<input type="text" id="char-input" autofocus="autofocus">
```

监听回车

1. 简要描述：在输入完毕后可以直接在输入框中输入回车，则进行验证字符验证码

2. 代码实现

```
char_code.onkeypress = listenDown
function listenDown(e) {
    var e = e || window.event || e.which;
    if(e.keyCode == 13) {   //如果是回车，进行验证码验证
        testCode()
    }
}
```

四、主要样式实现

1. 简要描述

    1. 布局问题：整体大概划分了两个部分，左边字符验证部分和右边的图片验证部分，主要使用的是设置一个主 div，和两个子 div，然后两个子 div 进行 `float: left` 浮动起来，形成两个同时水平排列

    2. 矢量图标问题：这次实现过程中，不仅使用了阿里的矢量图标库，也使用了 iconfont 的矢量图标库，两者既有差别又有很多相似之处

2. 主要实现部分

    1. 布局问题

```
.char-code {  //字符验证部分
    float: left;
    margin: 100px;
    width: 270px;
}

.captcha {    //图片验证部分
    float: left;
    width: 400px;
    margin: 0px auto;
    margin-top: 50px;
}
```

    2. 矢量图标问题

        1. 阿里矢量图标

```
<link rel="stylesheet" href="ali-font/iconfont.css">
  <!-- 引入阿里库 -->

<div class="icon iconfont iconicon-doubleright-line" id
="icon"></div>      <!-- 右移动图标 -->
<div class="icon iconfont iconicon_xuanzhong" id="icon">
</div>  <!-- 成功图标 -->
```

        2. iconfont 矢量图标

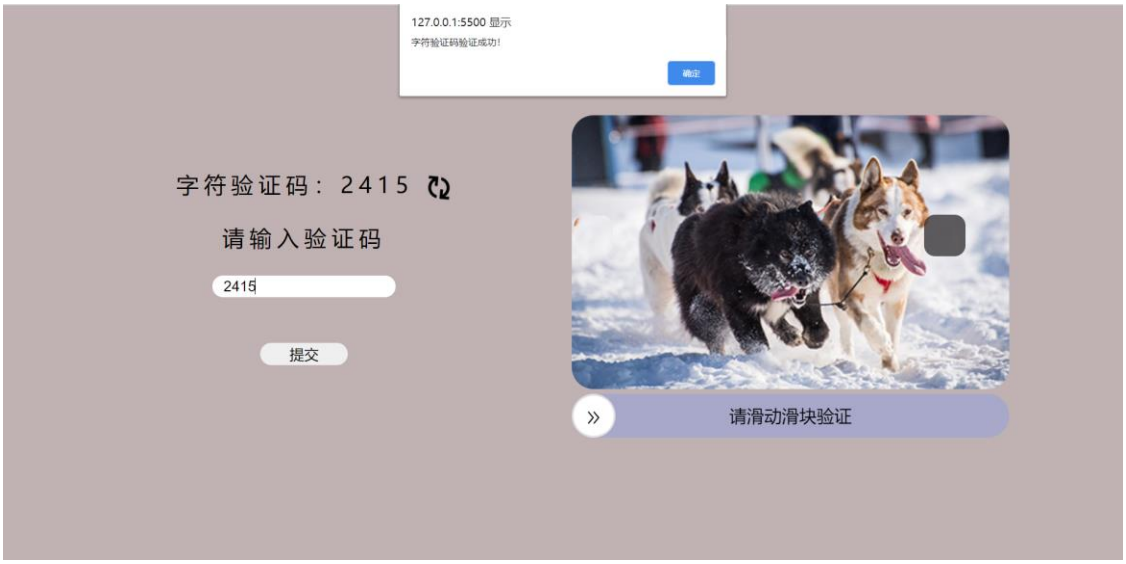
```
<!-- 引入 iconfont 库 -->
<link rel="stylesheet" href="./fa/css/all.min.css">
<link rel="stylesheet" href="./iconfont/iconfont.css">

<i class="fas fa-sync fa-spin" id="refresh"></i> <!--
刷新图标 -->
```

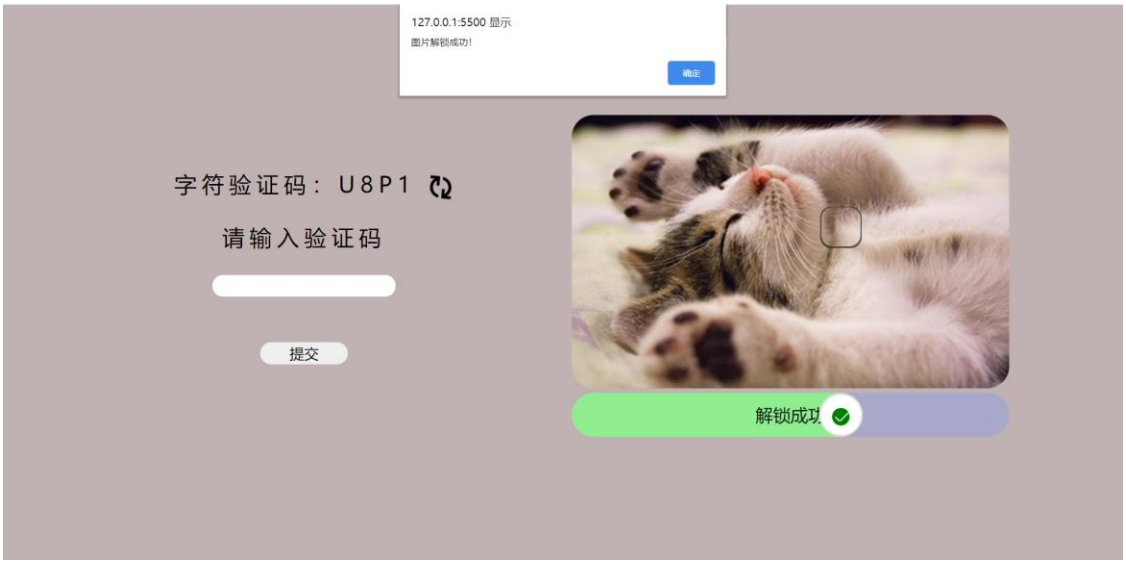
五、主要结果截图

1. 主界面

2. 字符验证成功界面



3. 图片验证成功界面



MVC 编程

一、主要完成内容

1. 使用 ASP.NET MVC + Code First 技术完成留言本系统
2. 具有普通用户和管理员用户两个角色
3. 普通用户具有登录，注册，发布个人留言，阅读全部留言，显示个人留言等功能
4. 管理员用户具有登录，注册，审核留言，删除留言，阅读留言等功能
5. 并且该留言本系统具有用户审核等功能

二、主要功能实现

非特定角色功能

1. 登录功能
    1. 登录验证简要逻辑
        1. 首先在视图中进行输入账号密码，在前端首先进行验证
        2. 如果前端输入出现异常，则前端进行简要格式验证，并将错误信息返还给用户，进行提示

1. 对应的输入使用 Razor，以便能够简单验证输入的数据
2. 保存错误信息使用`@Html.ValidationSummary()`，添加在页面中，但是进行隐藏，前端不直接显示，而通过 JavaScript 的 `alert` 进行简易的弹窗

3. 如果通过前端简要验证，则通过 POST 传输到后台 `Account/Login`，后台进行数据库查询
    1. 如果没有这条记录，则提示登录失败的信息，并返回登录视图，给用户重新登录
    2. 如果查询到对应记录，则辨别是管理员还是普通用户，如果是管理员则跳转到 `Admin/Index`，如果是普通用户则跳转到 `User/AllWords`，再进行后续操作

2. 登录功能前端代码

```
@model GuestBookSystem.Models.User
@{
    ViewBag.Title = "Login";
    Layout = "~/Views/Shared/_LayoutAccount.cshtml";
}

@using (Html.BeginForm("Login", "Account", new { ReturnUrl =
"" }, FormMethod.Post))
{
    <div class="user">
        @Html.TextBoxFor(a => a.Name, new { @placeholder = "
用户名" })
    </div>
    <div class="password">
        @Html.TextBoxFor(a => a.Password, new { @placeholder
= "密码", @type = "password" })
    </div>
    <div class="submit">
        <input type="submit" value="登录" id="submit">
    </div>
    <span hidden id="hidden">@Html.ValidationSummary()</span>
}
```

3. 登录验证后台代码

```
//在数据库中查找对应账号密码
var dbUser = db.Users.Where(
    a => a.Name == user.Name &&
    a.Password == user.Password).FirstOrDefault();
if (dbUser != null) //如果能找到对应账号密码
```

```
{
    ViewBag.STATE = true;   //记录登录状态
    if (dbUser.SRole.ToString() == "管理员")   //如果是管理员
    {
        Session["UserId"] = dbUser.UserId;  //记录UserId
        return RedirectToAction("Index", "Admin");  //跳转Adm
in/Index
    }
    else if (dbUser.SRole.ToString() == "普通用户") //如果是普
通用户
    {
        Session["UserId"] = dbUser.UserId;
        return RedirectToAction("AllWords", "User");    //跳
转User/AllWords
    }
}
```

2. 注册功能
    1. 注册功能简要逻辑
        1. 首先在视图中进行输入账号密码，在前端首先进行验证
        2. 如果前端输入出现异常，则前端进行简要格式验证，并将错误信息返还给用户，进行提示

           实现提示逻辑和登录功能相似，都是使用 `@Html.ValidationSummary()`
        3. 如果通过前端简要验证，则通过 POST 传输到后台 `Account/Register`，后台进行注册验证
            1. 如果注册失败，则提示注册失败的信息，并返回注册视图，给用户重新注册
            2. 如果注册成功，则跳转到登录界面，便于用户的登录
    2. 注册功能前端代码

```
@using (Html.BeginForm("Register", "Account", FormMethod.Pos
t))
{
    <div class="email">
        @Html.TextBoxFor(a => a.Email, new { @placeholder = "
注册邮箱" })
    </div>
    <div class="name">
        @Html.TextBoxFor(a => a.Name, new { @placeholder = "
注册用户名" })
```

```
        </div>
        <div class="password">
            @Html.TextBoxFor(a => a.Password, new { @placeholder
    = "注册密码" })
        </div>
        <div class="srole">
            <span class="text">
                注册角色:
            </span>
            <select id="SRole" name="SRole">
                <option value="0">管理员</option>
                <option value="1" selected>普通用户</option>
            </select>
        </div>
        <div class="submit">
            <input type="submit" value="注册" id="submit">
        </div>
        <span hidden id="hidden">@Html.ValidationSummary()</span>
    }
```

3. 注册功能后台代码

```
[HttpPost]
public ActionResult Register(User user)
{
    if (ModelState.IsValid) //如果状态通过
    {
        db.Users.Add(user); //在数据库中添加这个记录
        db.SaveChanges();   //保存数据库
        return RedirectToAction("Login");   //跳转到/Login
    }
    return View();  //如果不通过，则返回注册视图
}
```

3. 主页阅读留言功能

    1. 留言功能实现逻辑

        1. 通过数据库进行查询状态为通过的留言，并更加创建时间进行倒序，并将数据传送给前端

        2. 前端通过@foreach，进行遍历输出对应字段

    2. 留言功能前端代码

```
@foreach (var gb in Model)
{
    <tr>
        <td>
```

```
                <span class="username">@gb.User.Name :</span>
                <span class="context">@gb.Content</span>
                <span class="date">@gb.CreatedOn</span>
            </td>
        </tr>
    }
```

3. 留言功能后台代码

```
GBSDBContext db = new GBSDBContext();   //数据库
// GET: GuestBook
public ActionResult Index()
{
    var gb = db.Guestbooks.Where(a => a.isPass == true);    //如果查询到状态为通过的留言
    return View(gb.OrderByDescending(a=>a.CreatedOn).ToList()); //则根据创建留言时间进行倒序
}
```

普通用户功能

1. 发布个人留言

    1. 发布个人留言简要逻辑

        1. 首先在前端进行个人留言的输入，如果出现格式错误，则会在前端就进行提醒，使用的逻辑还是@Html.ValidationSummary()，和前面相同

        2. 前端验证格式成功之后，通过 POST 方式，提交到后台的 User/CreateWords，再进行处理

        3. 后台首先获取登录的 ID，自动添加到留言本中，再将留言本的这条记录保存到数据库中，最后将个人留言界面返回给用户

    2. 发布个人留言前端代码

```
<form action="/User/CreateWords" method="post">
    <table class="second">
        <tr>
            <td>标题：</td>
            <td>@Html.TextBoxFor(g => g.Title, new { @autocomplete = "off", autofocus="autofocus"})</td>
        </tr>
        <tr>
            <td>内容：</td>
            <td>@Html.TextAreaFor(g => g.Content, new { cols = 50, rows = 10 })</td>
        </tr>
        <tr>
```

```
            <td></td>
            <td class="button"><input type="submit" value="发
布" /></td>
        </tr>
    </table>
</form>
```

3. 发布个人留言后台代码

```
if (ModelState.IsValid)
{
    gb.User = db.Users.Where(a => a.UserId == UserId).FirstOr
Default(); //找到登录用户的ID，自动添加到留言本中
    //gb.CreatedOn = System.DateTime.Now;
    db.Guestbooks.Add(gb);  //在数据库中添加这条记录
    db.SaveChanges();   //保存数据库
    return RedirectToAction("MyWords"); //跳转到/User/MyWords，
显示个人留言
}
```

2. 显示个人留言

1. 显示个人留言简要逻辑

    1. 后台通过登录用户的 ID，在数据库中进行查找，查找到的结果通过创建时间逆序传递给前端

    2. 前端通过 foreach 进行遍历，将结果进行输出，特别地是在是否通过审核这一栏，进行 if 判断，如果通过则显示审核通过，否则显示正在审核

2. 显示个人留言前端代码

```
@foreach (var gb in Model)
{
<tr class="clearfix">
    <td><span>@gb.User.Name</span></td>
    <td><span>@gb.CreatedOn</span></td>
    <td><span>@gb.Title</span></td>
    <td><span>@gb.Content</span></td>
    <td>
        <span class="pass">
            @if (gb.isPass == true)
            {@Html.DisplayName("审核通过") }
            else
            {@Html.DisplayName("正在审核")}
        </span>
    </td>
```

```
</tr>
}
```

3. 显示个人留言后台代码

```
var gb = db.Guestbooks.Where(a => a.User.UserId == UserId);
return View(gb.OrderByDescending(a=>a.CreatedOn).ToList());
```

3. 显示全部留言

    1. 显示全部留言简要逻辑

        1. 使用的整体逻辑和主界面阅读全部留言类似
        2. 特别地是在普通用户界面可以较为详细地查看留言，例如可以多查看到留言本标题

    2. 显示全部留言前端代码

```
@foreach (var gb in Model)
{
<tr class="clearfix">
    <td><span>@gb.User.Name</span></td>
    <td><span>@gb.CreatedOn</span></td>
    <td><span>@gb.Title</span></td>
    <td><span>@gb.Content</span></td>
</tr>
}
```

    3. 显示全部留言后台代码

```
public ActionResult AllWords()
{
    var gb = db.Guestbooks.Where(a => a.isPass == true);
    return View(gb.OrderByDescending(a=>a.CreatedOn).ToList
());
}
```

管理员功能

1. 审核留言

    1. 审核留言简要逻辑

        1. 后台首先获取到所有未通过审核的留言，将这些留言按照创建时间逆序传递给前端
        2. 前端将这些留言进行显示，并在每一条留言后面都添加一个审核通过的按钮

3. 点击审核通过的按钮，则跳转到 Admin/CheckMessageToFind，进行确认是否通过审核

4. 再显示确认通过审核的界面，让管理员用户进行进一步操作

2. 审核留言前端代码

    1. 显示未通过审核的留言

```
@foreach (var gb in Model)
{
<tr class="clearfix">
    <td><span>@gb.User.Name</span></td>
    <td><span>@gb.CreatedOn</span></td>
    <td><span>@gb.Title</span></td>
    <td><span>@gb.Content</span></td>
    <td><span class="button">@Html.ActionLink("审核通过",
 "CheckMessageToFind", new { id = gb.GuestbookId })</sp
an></td>
</tr>
}
```

    2. 确认通过审核

```
@using (Html.BeginForm())
{
    <div class="info">
        <table class="confirmed">
            <tr>
                <td>确认是否通过</td>
            </tr>
            <tr>
                <td>留言人：@Model.User.Name</td>
            </tr>
            <tr>
                <td>内容：@Model.Content</td>
            </tr>
            <tr>
                <td>时间：@Model.CreatedOn</td>
            </tr>
            <tr>
                <td class="button"><input type="submit"
 value="确认"></td>
            </tr>
        </table>
        @Html.ActionLink("X", "CheckIndex", new { }, ne
w { @class = "close" })
    </div>
}
```

3. 审核留言后台代码

    1. 显示所有未通过审核的留言

```
public ActionResult CheckIndex()
{
    GetUser();
    var gb = db.Guestbooks.Where(a => a.isPass == fals
e);   //获取没有通过审核的留言本
    return View(gb.OrderByDescending(a=>a.CreatedOn).To
List()); //通过创建时间逆序，并传递给前端
}
```

    2. 确认是否通过审核

```
public ActionResult CheckMessageToFind(int id)
{
    GetUser();
    var gb = db.Guestbooks.Find(id);     //查找到留言本ID
    return View(gb);     //返回确认界面，给管理员用户确认是
否通过审核
}
```

    3. 如果审核通过

```
[HttpPost, ActionName("CheckMessageToFind")]
public ActionResult CheckMessage(int id)
{
    GetUser();
    var gb = db.Guestbooks.Find(id);     //获取留言本ID
    gb.isPass = true;    //将通过状态设置为true
    db.SaveChanges();    //保存数据库
    return RedirectToAction("CheckIndex");  //返回到Adm
in/CheckIndex，显示还未通过的留言
}
```

2. 删除留言

    1. 删除留言简要逻辑

        1. 后台首先获取到所有可删除的留言，将这些留言按照创建时间逆序传递给前端

        2. 前端将这些留言进行显示，并在每一条留言后面都添加一个删除留言的按钮

        3. 点击删除留言的按钮，则跳转到`Admin/Delete，进行确认是否删除

4. 再显示确认删除的界面，让管理员用户进行进一步操作

2. 删除留言前端代码

    1. 显示所有能删除的留言

```
@foreach (var gb in Model)
{
<tr class="clearfix">
    <td><span>@gb.User.Name</span></td>
    <td><span>@gb.CreatedOn</span></td>
    <td><span>@gb.Title</span></td>
    <td><span>@gb.Content</span></td>
    <td><span class="button">@Html.ActionLink("删除留言",
 "Delete", new { id = gb.GuestbookId })</span></td>
</tr>
}
```

    2. 确认是否删除留言

```
@using (Html.BeginForm())
{
    <div class="info">
        <table class="confirmed">
            <tr>
                <td>确认是否删除</td>
            </tr>
            <tr>
                <td>留言人：@Model.User.Name</td>
            </tr>
            <tr>
                <td>内容：@Model.Content</td>
            </tr>
            <tr>
                <td>时间：@Model.CreatedOn</td>
            </tr>
            <tr>
                <td class="button"><input type="submit"
 value="确认"></td>
            </tr>
        </table>
        @Html.ActionLink("X","Index", new { }, new { @c
lass="close"})
    </div>
}
```

3. 删除留言后台代码

    1. 显示所有可删除的留言

```
public ActionResult Index()
{
    GetUser();
    return View(db.Guestbooks.OrderByDescending(db => d
b.CreatedOn).ToList());  //通过创建时间逆序，传递给前端
}
```

2. 是否删除留言

```
public ActionResult Delete(int id)
{
    GetUser();
    var gb = db.Guestbooks.Find(id);    //找到这条留言的
ID
    return View(gb);    //返回确认删除的视图
}
```

3. 确认删除这条留言

```
[HttpPost, ActionName("Delete")]
public ActionResult DeleteConfirmed(int id)
{
    GetUser();
    var gb = db.Guestbooks.Find(id);    //找到这条留言的
ID
    db.Guestbooks.Remove(gb);   //在数据库中删除这条留言
    db.SaveChanges();   //保存数据库
    return RedirectToAction("Index");   //返回/Admin/In
dex
}
```

三、部分额外功能

超长文本省略处理

1. 功能实现简要描述

    1. 超长文本进行省略处理，鼠标移动上去显示全文
    2. 在 CSS 中设置空白处理方式：不换行，超出部分隐藏和文本超出使用省略号
    3. 在 JavaScript 中获取到所有 span 标签，遍历所有的 span 如果其中文字的宽度大于元素的宽度，则将该 span 标签的 title 属性设置为其内容

2. CSS 部分（以主页留言本显示为例）

```
.guestbook .main .list tr td span {
    display: inline-block;
    padding-left: 20px;
```

```
    height: 50px;
    text-align: center;
    line-height: 50px;
    overflow: hidden;
    text-overflow: ellipsis;
    white-space: nowrap;
}
```

3. JavaScript 部分

```
var spans = document.getElementsByTagName("span")
for (let i = 0; i < spans.length; ++i) {
    if (spans[i].scrollWidth > spans[i].offsetWidth) {
        spans[i].setAttribute("title", spans[i].innerText)
    }
}
```

用户验证

1. 功能实现简要描述
    1. 必须登录后才可以使用普通用户和管理员操作
    2. 在普通用户控制器 User 和管理员控制器 Admin 中添加[Authorize]
    3. 如果在未登录的情况下，直接使用 User 或 Admin 中的方法，则会重定向到登录界面
    4. 只有在对应角色登录后，才可以使用对应角色的方法
    5. 在使用完对应角色的方法后，可以使用退出功能，进行账号退出
2. 功能实现后台代码
    1. Web.confi 设置为表单验证

```
<authentication mode="Forms">
    <forms loginUrl="/Account/LoginFirst" defaultUrl="/" time
out="30" path="/">
    </forms>
</authentication>
```

    2. Login 方法中进行设置

```
FormsAuthenticationTicket ticket = new FormsAuthenticationTic
ket(1, user.UserId.ToString(), DateTime.Now, DateTime.Now.Add
Minutes(30), false, user.SRole.ToString());
string hashTicket = FormsAuthentication.Encrypt(ticket);
HttpCookie cookie = new HttpCookie(FormsAuthentication.FormsC
ookieName, hashTicket);
```

```
cookie.HttpOnly = true;
Response.Cookies.Add(cookie);
```

3. 在普通用户控制器 User 和管理员控制器 Admin 中添加[Authorize]，使其进行验证

4. 账号退出

```
public ActionResult Logout()
{
    FormsAuthentication.SignOut();
    return RedirectToAction("Index", "GuestBook");
}
```

四、其他实现

数据库实现

1. 数据库上下文

```
public class GBSDBContext : DbContext
{
    public DbSet<Guestbook> Guestbooks { get; set; }
    public DbSet<Admin> Admins { get; set; }
    public DbSet<User> Users { get; set; }
}
```

2. 用户角色信息

```
public class User
{
    public int UserId { get; set; }
    [System.ComponentModel.DataAnnotations.Required(ErrorMessage =
"邮箱不能为空")]
    [EmailAddress(ErrorMessage ="邮箱格式不对")]
    public string Email { get; set; }
    [Unique]
    [System.ComponentModel.DataAnnotations.Required(ErrorMessage =
"用户名不能为空")]
    [MaxLength(10,ErrorMessage ="用户名长度不能大于 10 个字符")]
    public string Name { get; set; }
    [System.ComponentModel.DataAnnotations.Required(ErrorMessage =
"密码不能为空")]
    [MaxLength(20, ErrorMessage = "密码长度不能大于 20 个字符")]
    [MinLength(6,ErrorMessage ="密码长度不能少于 6 个字符")]
    [DataType(DataType.Password)]
    public string Password { get; set; }
    public SystemRole SRole { get; set; }
    public virtual ICollection<Guestbook> Guestbooks { get; set; }
}
```

3. 留言本信息

```
public class Guestbook
{
    public int GuestbookId { get; set; }
    [Required(ErrorMessage = "留言标题不能为空")]
    [MaxLength(20, ErrorMessage = "留言标题不超过 20 个字符")]
    public string Title { get; set; }   //留言标题
    [Required(ErrorMessage = "留言内容不能为空")]
    [MinLength(1, ErrorMessage = "留言内容不少于 1 个字符")]
    public string Content { get; set; } //留言内容
    [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
    public DateTime CreatedOn { get; set; } //创建日期时间
    public bool isPass { get; set; }
    public virtual User User { get; set; }
}
```

默认模板实现

1. 账号登录、注册部分

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scal
e=1.0">
    <title>@ViewBag.Title</title>
    <link rel="stylesheet" href="~/Content/guestbook-index.css">
    <link href="~/IconFont/fa/css/all.min.css" rel="stylesheet" />
    <link href="~/IconFont/iconfont/iconfont.css" rel="stylesheet"
/>
</head>
<body>
    <img class="img" src="~/Image/bg.jpg">
    <div class="guestbook clearfix">
        <div class="main">
            <div class="back">
                <a href="/GuestBook/Index">
                    <i class="fas fa-angle-double-left"></i>
                    <span>返回</span>
                </a>
            </div>
            <div class="title">
                <span>欢迎访问留言本</span>
            </div>
            <div class="buttons">
                <div class="button login">
                    <span>@Html.ActionLink("用户登录", "Login", "Acc
ount")</span>
```

```
                </div>
                <div class="button register">
                    <span>@Html.ActionLink("用户注册", "Register", "
Account")</span>
                </div>
            </div>
            <div class="main-login">
                @RenderBody()
            </div>
            <footer>
                <p>&copy; 2020-@DateTime.Now.Year RuanXinWei 版权所
有</p>
            </footer>
        </div>
    </div>
</body>
```

2. 后台部分

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scal
e=1.0">
    <title>@ViewBag.NAME 后台</title>
    <link rel="stylesheet" href="/Content/admin-index.css">
</head>
<body>
    <div class="title">
        <div class="welcome">
            欢迎 @ViewBag.NAME !
        </div>
    </div>
    <div class="main clearfix">
        @RenderBody()
    </div>
</body>
</html>
```

五、主要结果截图

1. 留言本主页面

2. 注册页面

3. 登录页面

4. 用户页面

留言人	时间	标题	内容
xiaohong	2020/11/30 16:10:34	小红的第四条留言	低头思故乡
ruanxinwei	2020/11/28 21:17:27	阮新伟的第五条留言	似曾相似燕归来
xiaoming	2020/11/25 12:06:15	小明的第三条留言	冲冲冲!
ruanxinwei	2020/11/24 19:16:55	阮新伟的第三条留言	夕阳西下几时回
xiaohong	2020/11/24 15:10:16	小红的第三条留言	举头望明月
ruanxinwei	2020/11/22 18:16:37	阮新伟的第二条留言	去年天气旧亭台
xiaohong	2020/11/22 14:09:57	小红的第二条留言	疑是地上霜
xiaohong	2020/11/21 13:09:26	小红的第一条留言	床前明月光
ruanxinwei	2020/11/20 17:16:22	阮新伟的第一条留言	一曲新词酒一杯
hello	2020/11/20 13:23:46	鲁迅语录二	真正的勇士，敢于直面惨淡的人生，...
xiaoming	2020/11/20 11:05:52	小明的第二条留言	奥利给!

5. 管理员页面

6. 确认信息页面

小结

首先，我很庆幸选择了学习.NET 架构，因为通过学习这门课程，让我了解到了前端和后台可以通过 MVC 进行配合。就在今年完成数据库课程设计的时候，我就十分迷茫，不知道该如何将前端和后台进行配合，虽然最后使用了 PHP 进行后台操作数据库完成了课程设计，但是深刻体会到了用字符串保存 SQL，网站整体样式的设计等痛苦。

但是在我学习完.NET 架构后，学会了用 Code-first 来操作数据库，这种方式不仅能够避免在编写 SQL 的时候出现字符错误，还能够很好地进行保存，修改等一系列操作，通过类操作数据库的好处就从而体现出来了。还有在为网站设计整体风格的时候，通过添加布局页，一方面大大减少了代码的冗余量，另一方面也提高了开发的效率。

总的来说，从最开始的客户端技术编程到最后的 MVC 编程，让我慢慢从只会做一些简单的样式到能够实现一些具有实际意义的样式，让我从只会做样式到能够配合后台来实现一个实际的网站。到最后，回顾这门课的学习，虽然过程真的很苦，每天熬夜干代码，但是收获到的结果也是十分丰厚的。