上讲回顾

- ASP.NET应用程序生命周期
- ASP.NET网页
- Page类的内置对象
  - Response
  - Request
  - Server

测一测

在一个名为Login的Web网页中，先需要在其Page_Load事件中判断该页面是否回发，请问需要使用下列（）属性。

A. Page.IsCallback

B. Page.IsAsync

C. Page.IsPostBack

D. Login.IsPostBack

第3章 ASP.NET技术基础

内容

- ASP.NET应用程序生命周期
- ASP.NET网页
- Page类的内置对象
- Web应用的配置与配置管理工具
- **Web应用的异常处理**

Part 4

WEB应用的异常处理

3.5 Web应用的异常处理

- 为什么要进行异常处理
- try-catch异常处理块
- 页面级的Page_Error事件处理异常
- 页面级的ErrorPage属性处理异常
- 应用程序级异常处理
- 配置应用程序的异常处理

3.5.1 为什么要进行异常处理

- 对于一个Web应用程序来说，出错是在所难免的。
- 当应用程序发布后，可能由于代码本身的缺陷、网络故障或其它问题，导致用户请求得不到正确的响应，出现一些对用户而言毫无意义的错误信息，甚至泄漏了一些重要信息，让恶意用户有了攻击系统的可能。

- 当错误发生时，必须做好两件事情：

- 一是将错误信息记录日志，发邮件通知网站维护人员，方便技术人员对错误进行跟踪处理；
- 二是以友好的方式提示最终用户页面发生了错误，而不能将未处理的错误信息显示给用户。

- **ASP.NET提供了5种异常处理机制（优先级别从高到低）：**
- 通过try-catch异常处理块处理异常。
- 通过页面级的Page_Error事件处理异常。
- 通过页面级的ErrorPage属性处理异常。
- 通过应用程序级的Application_Error事件处理异常。
- 通过配置应用程序<customErrors>配置项处理异常。

3.5.2 try-catch异常处理块

- 【例3-11】演示try-catch异常处理块的使用。

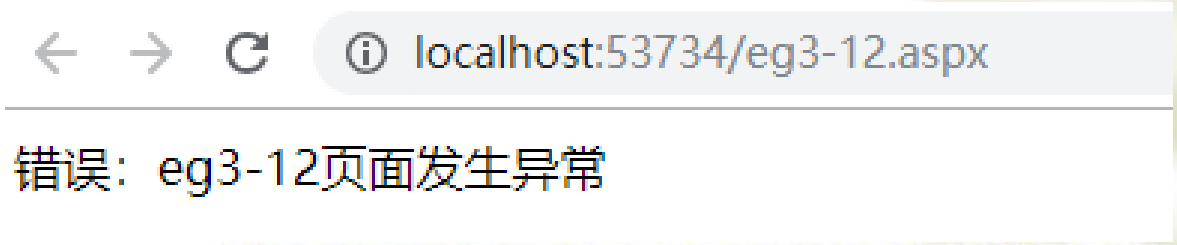
```
protected void Page_Load(object sender, EventArgs e)
{
    int x = 5;
    int y = 1;
    //int r = x / y;
    try
    {
        int r = x / y;
    }
    catch (Exception ex)
    {
        Response.Write("发生异常，原因是：" + ex.Message + "<br>");
    }
    finally
    {
        Response.Write("请联系@@@");
    }
}
```

3.5.3 页面级的Page_Error事件处理异常

- Page类有个异常处理事件（Page_Error），当页面引发了未处理的异常时触发该事件。因此，可在该事件中添加代码处理页面中发生的未处理异常。
- 【例3-12】使用页面级的Page_Error事件处理异常。

```
protected void Page_Load(object sender, EventArgs e)
{
    throw new Exception("eg3-12页面发生异常");
}

protected void Page_Error(object sender, EventArgs e)
{
    //获取未处理的异常
    Exception objErr = Server.GetLastError();
    //输出异常信息
    Response.Write("错误：" + objErr.Message);
    //清除异常，避免上一级异常处理
    Server.ClearError();
}
```

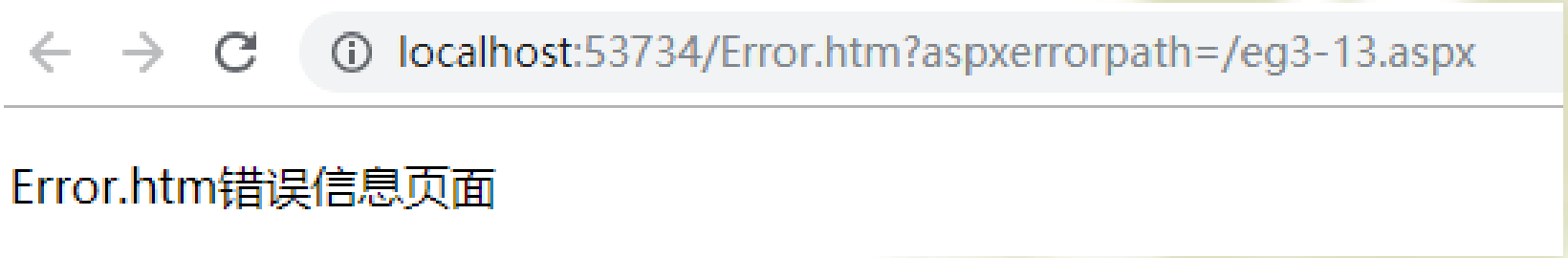


3.5.4 页面级的ErrorPage属性处理异常

- 通过设置页面的ErrorPage属性，可以让页面发生错误的时候重定向至友好的错误描述页面。例如，this.ErrorPage = "~/Error.htm"。
- 注意，要让ErrorPage属性发挥作用，web.config文件中的<customErrors>配置项中的mode属性必须设为"On"。
  - 添加到<system.web>标签中

【例3-13】使用页面级的ErrorPage属性处理异常。

```
protected void Page_Load(object sender, EventArgs e)
{
    //如果发生异常，跳转到Error.htm页面
    this.ErrorPage = "Error.htm";
    throw new Exception("eg3-13页面发生异常");
}
```



- 注意：如果Page_Error和ErrorPage都存在，当该页抛出异常时，页面执行顺序是怎样的呢？

- 页面会先执行Page_Error事件处理方法，如果Page_Error()事件中调用Server.ClearError()方法清除异常信息，则不会跳转到ErrorPage属性指定页面；如果没有调用Server.ClearError()，异常信息会继续向上抛，页面会跳转到ErrorPage指定页面。

3.5.4 应用程序级异常处理

- 与Page_Error事件相类似，可以使用Global.asax文件中的Application_Error事件捕获发生在应用程序中的所有未处理的异常。
- 由于在整个应用程序范围内发生异常，并且都没有使用前面的方法处理这些异常，则会触发Application_Error事件处理这些应用程序级别的错误。

【例3-14】演示如何使用Application_Error事件捕获发生在应用程序中所有未处理的异常，并将捕获的异常信息写入Windows事件日志。

eg3_14.aspx

```
protected void Page_Load(object sender, EventArgs e)
{
    // 故意抛出异常
    throw new Exception("eg3_14页面发生异常");
}
```

Global.asax

```
void Application_Error(object sender, EventArgs e)
{
    // 在出现未处理的错误时运行的代码
    Exception objErr = Server.GetLastError();

    // 将异常信息写入日志
    // 发送邮件通知管理员

    //清除异常
    Server.ClearError();

}
```

3.5.5 配置应用程序的异常处理

- 如果既没有设置页面级异常处理，也没有设置应用程序级异常处理，那么还可以通过在配置文件**web.config**中设置配置来处理整个应用程序中未处理的异常。

- 具体方法是修改应用程序根目录下的web.config文件，在system.web下面对customErrors元素进行以下更改：

```
<customErrors mode="RemoteOnly"
defaultRedirect="Error.htm">
    <error statusCode="403" redirect="NoAccess.htm" />
    <error statusCode="404" redirect="FileNotFound.htm" />
</customErrors>
```

- 上述代码中，mode用于设置错误页面的显示模式，有如下3个可选项。
  - RemoteOnly：如果应用程序发生未处理的异常，则对远程用户显示一个通用的错误页面，对本地用户将显示详细的错误页面。
  - Off：如果应用程序发生未处理的异常，无论请求是本地还是远程，对所有用户都显示详细的错误信息。
  - On：如果应用程序发生未处理的异常，无论请求是本地还是远程，对所有用户都显示通用的错误信息。

- 如果Application_Error和<customerErrors>同时存在，也存在执行顺序的问题。
- 因为Application_Error事件优先级高于<customErrors>配置项，所以发生应用程序级错误时，优先执行Application_Error事件中的代码，如果Application_Error事件中调用了Server.ClearError()函数，则<customerErrors>配置节中的defaultRedirect不起作用，因为异常已经被清除；如果Application_Error事件中没用调用Server.ClearError()函数，则会重新定位到defaultRedict指定的URL页面，为用户显示友好出错信息。

Web应用的异常处理机制

- **ASP.NET提供了5种异常处理机制（优先级别从高到低）：**
- 通过try-catch异常处理块处理异常。
- 通过页面级的Page_Error事件处理异常。
- 通过页面级的ErrorPage属性处理异常。
- 通过应用程序级的Application_Error事件处理异常。
- 通过配置应用程序<customErrors>配置项处理异常。

3.6 小结

- ASP.NET应用程序生命周期；
- ASP.NET网页；
- Page类的内置对象；
- Web应用的异常处理机制。

本章结束!