您的潜力，我们的动力

Microsoft
微软（中国）有限公司

ASP.NET错误处理和程序优化

讲师：邵志东

msdn

MSDN Webcasts

课前准备

- Dot Net FrameWork
- VS.NET 2002/2003
- C#/VB.NET

- Level 200

议程

- 错误类型以及处理方式
- 提高数据访问性能
- 服务器控件的使用
- 缓存的使用
- 提高性能的实用技巧

错误类型以及处理方式

1、错误的类型

- 分析程序错误
  - 语法错误：语法有问题
  - 逻辑错误：除0错误、类型不匹配、不正确输出、使用不正确的对象、处理无效的数据
- 编译错误：使用了不能被语言编译器识别，但ASP.NET能识别的关键字或语句时发生的错误
- 运行时错误
- 配置错误：Web.config文件出错

2、 错误的处理

- 使用验证控件
- 编程处理
  - 校验语句
  - try...catch...finally
  - Page_Error
  - Application_Error
- 在应用程序配置文件中，为应用程序执行的声明性错误处理

DEMO1

校验(checked)和非校验(unchecked)语句

异常类：Exception

- 所有异常对象的基类
- 属性：
  - HelpLink:到一个文件的链接，该文件包含关于错误的更详细的信息
  - InnerException:一个内部异常的引用。如果一个异常被捕获，并被传递给另外一个异常处理程序，则该属性返回到第一个异常的引用
  - Message：描述异常的消息
  - Source：一个字符串，其中包含引起错误的对象的名称
  - StackTrace：返回一个指出错误原因的栈跟踪
  - TargeSite：引起错误的方法

Page对象的Error事件

- 使用模板

```
void Page_Error(object sender,EventArgs e)
{
   Response.Write("发生错
   误:"+Server.GetLastError().ToString());
   Server.ClearError();
}
```

Appliation对象的Error事件

- 应用程序中任何页面抛出异常都会调用
- 在global.asax中
- 形式为：

```
void Application_Error(object sender,EventArgs
e)
{
    ...
}
```

DEMO2

编写到windows错误日志
把错误发送邮件到管理员

利用配置文件处理错误

- ASP.NET同以前的ASP一样，当服务器上发生了一个运行时间或编译时间错误时，就会生成一个html 错误页面。但是与ASP不同，ASP.NET格外关注的是：要确保在默认状态下，不会因为这个错误的发生而泄露“安全”信息。
- ＜system.web>

```
<customErrors defaultRedirect="url" mode="RemoteOnly">
        <error statusCode="code" redirect="url"></error>
        </customErrors>
 ＜/system.web>
```

议程

- 错误类型以及处理方式
- 提高数据访问性能
- 服务器控件的使用
- 缓存的使用
- 提高性能的实用技巧

性能引言

- 是
  - XmlDocument LoadXML(string strFileID)　//加载XML
  - bool CheckIDExisit(string strFileID,string strID) //判断节点是否存在
- 还是
  - bool CheckIDExisitByXml (string strXml,string strID)　//判断节点是否存在
  - 或 bool CheckIDExisitByXml (XmlDocument objXml,string strID)　//判断节点是否存在

逻辑设计

- 建议: 采用 3层 逻辑模型
  - Pages (.aspx) and User Controls (.ascx) UI
  - Business and Data Access classes in \bin dir
  - Data within a SQL Database via SPROCs

使用最佳的Data Provider

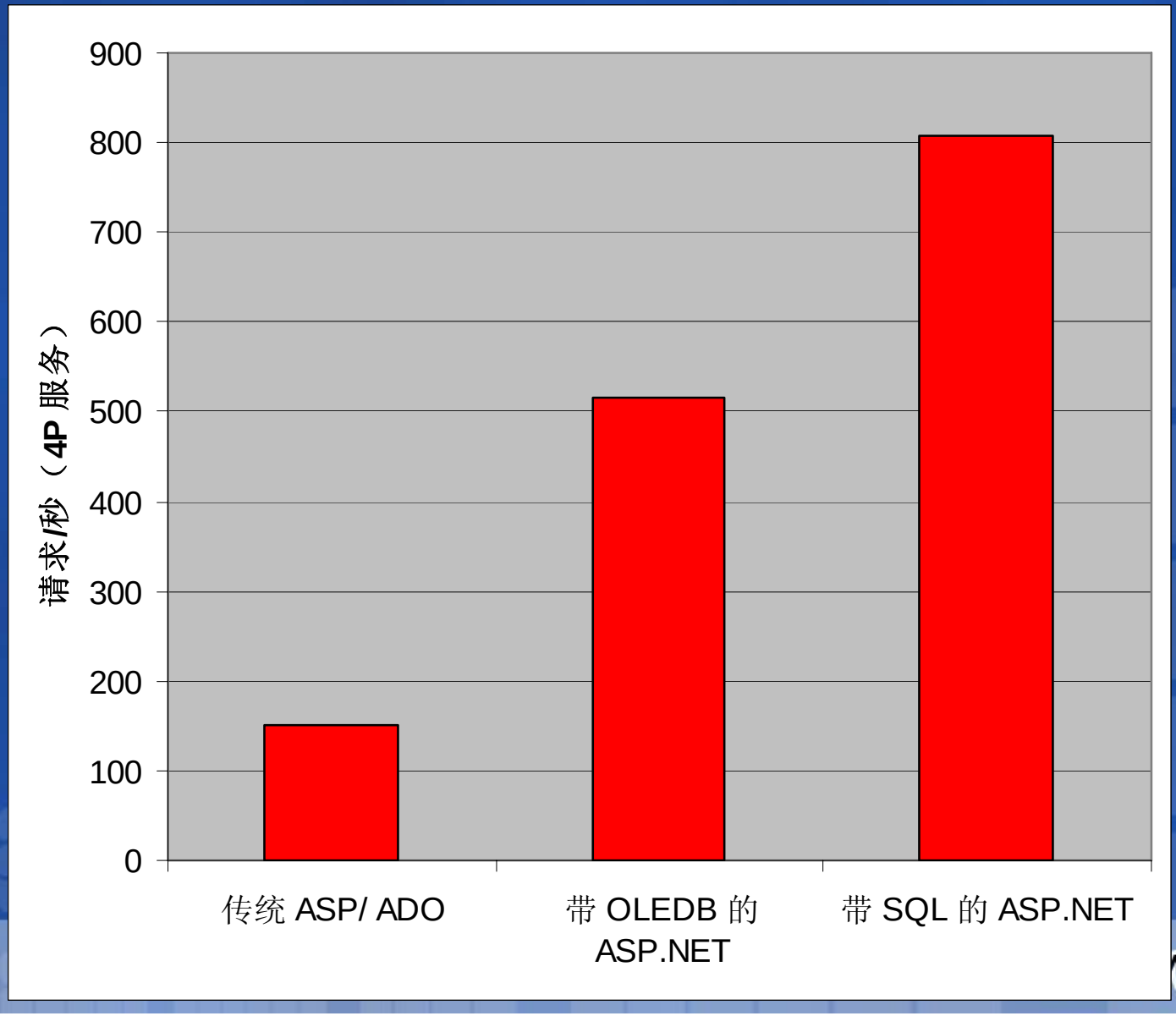
- ADO.NET 可支持多个Provider：
  - System.Data.SqlClient
  - System.Data.OracleClient
  - System.Data.OleDb
  - System.Data.Odbc
- 所有Provider的编程模型相同
  - 但是性能方面存在明显差异
- 建议：使用最佳Provider
  - 在访问 MSDE/SQL 时始终使用 SqlClient
  - 在访问 Oracle 时始终使用 OracleClient

Data Provider测试

- 方案：
  - 从 SQL Northwinds 数据库中抽取 50 行
  - <%= %> html 表格格式化技术

- 衡量的三种技术(不同的Data Provider)：
  - 传统 ASP/ADO
  - ASP.NET w/ System.Data.OleDb 提供商
  - ASP.NET w/ System.Data.SqlClient 提供商

数据性能测试



DataReaders 和 DataSets

- DataReader
  - 对查询的结果提供了单向读取的操作
  - 轻量快速 – 但在Reader为关闭之前始终处于连接状态
- DataSet
  - 非链接的数据访问方式
  - 内部使用DataReader用于获取数据
  - 在完成DataSet的获取后会自动关闭DataReader
- 如何选择?
  - 依赖于您的应用
  - 般情况下,读取大量数据,对返回数据不做大量处理用SqlDataReader.对返回数据大量处理用datset比较合

DataReader？DataSet

通常情况下 DataReader比DataSet快 16%！！

ExecuteNonQuery和ExecuteScalar

- ExecuteNonQuery
  - 对数据的更新不需要返回结果集
  - 由于不返回结果集可省掉网络数据传输。它仅仅返回受影响的行数。如果只需更新数据用ExecuteNonQuery性能的开销比较小。
- ExecuteScalar
  - 它只返回结果集中第一行的第一列。使用 ExecuteScalar 方法从数据库中检索单个值（例如id号）。
  - 与使用 ExecuteReader 方法， 返回的数据执行生成单个值所需的操作相比，此操作需要的代码较少
- 如何选择?
  - 只需更新数据用ExecuteNonQuery.单个值的查询使用ExecuteScalar

数据的绑定DataBinder

- 一般的绑定方法<%# DataBinder.Eval(Container.DataItem, “字段名”) %>用DataBinder.eval 绑定不必关心数据来源(Dataread或dataset)。不必关心数据的类型eval会把这个数据对象转换为一个字符串。在底层绑定做了很多工作，使用了反射性能。正因为使用方便了，但却影响了数据性能。
- 直接转换成DataRowView的话，将会给性能带来很大提升：
- <@% ((DataRowView)Container.DataItem)["字段名"] %>

连接池

- ADO.NET 拥有内置的连接池
  - 自动缓存/重新使用连接
  - 不必为此编写任何代码
- 代码建议：
  - “在后期打开代码中的连接，然后在早期将其关闭”
  - 切勿长时间保持连接状态
  - 完成后应立即显示地关闭数据库连接，以将其返回至池中

连接池

- 优化提示：
  - 不同的连接字符串可以生成多个不同的连接池
  - 在 Web.Config 中存储单个连接字符串
  - 使用 ConfigurationSettings.AppSettings，以在运行时采用编程形式对其进行访问
- 始终应明确关闭数据连接，避免连接泄漏
  - 否则连接将在下一次垃圾收集之前保持打开状态
  - 泄露连接会显著降低性能

使用存储过程

- 建议将 SPROC 用于数据存取
  - 通过 DBA 进行更轻松的性能调试
  - 通过使用数据库事务处理避免出现分布事务成本
  - 有助于防止 SQL 注入攻击
  - 有助于消除应用与数据库反复调用的成本
- 有趣的提示：
  - 可以通过企业管理器来关闭动态 SQL 支持，以强制使用 SPROC

DEMO3

使用连接池优化程序

议程

- 错误类型以及处理方式
- 提高数据访问性能
- 服务器控件的使用
- 缓存的使用
- 提高性能的实用技巧

服务器控件

- 提供了清晰的编程模型（重用，简洁，宜用）
  - 创建 ASP.NET 页面所倡导的模式
- 对性能优化而言有两点需要注意:
  - ViewState
  - 控件数量

ViewState 管理

- ASP.NET controls 能够维护页面Control元素的状态
  - 状态以“viewstate” hidden field进行传递
- 负面影响:
  - 增加网络负荷(both on render and postback)
  - 额外的服务器性能消耗 （serialize values to/from viewstate）
- Viewstate灵活性:
  - 页面级（Can disable viewstate entirely for a page）
  - 控件级（Can disable viewstate usage on a per control basis）
- 建议:
  - 认真审核该功能的使用
  - 若不使用PostBack功能，请在页面级屏蔽ViewState
  - PostBack时每次都重新生成控件，请对控件级的ViewState屏蔽
  - 使用 <%@ Page Trace=“true” %>跟踪ViewState的大小

有关ViewState管理提示

- 如果您希望更明确的限制 viewstate 的使用，可将 ASP.NET 配置为默认情况下处于关闭状态
- Machine.config：

```
<configuration>
    <system.web>
        <pages enableViewState=“false”/>
    </system.web>
</configuration>
```

- 之后需要 viewstate 的页将在页面指令中手动对其进行设置：
  - <%@ Page EnableViewState=“true” %>

生成的控件数量

- 页面上的每个服务器控件的生成都存在固定的成本
  - 每个控件的成本通常可以忽略不计
- 复合控件有时可以屏蔽使用的控件数量，尽管会出现以下情况
  - 聚集成本有时可以累加
  - 打开 ASP.NET Trace即可查看实际计数

议程

- 错误类型以及处理方式
- 提高数据访问性能
- 服务器控件的使用
- 使用缓存进行程序优化
- 提高性能的实用技巧

缓存技术

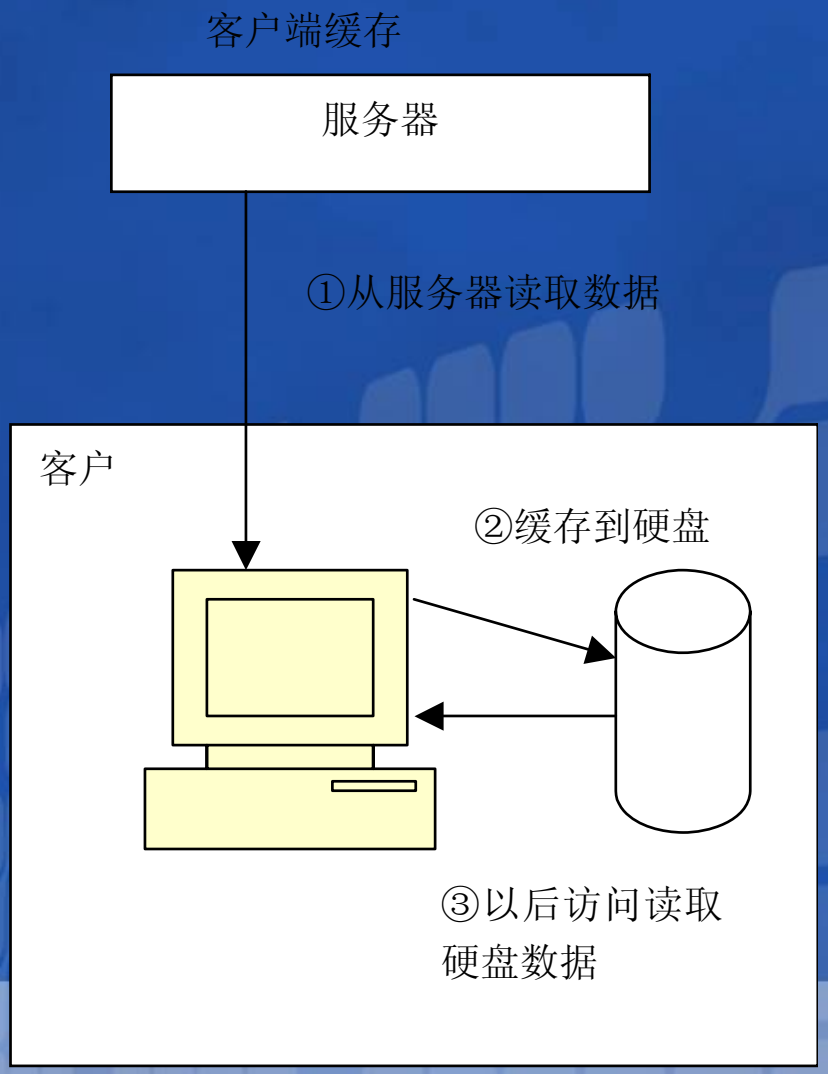
1、什么是缓存技术？

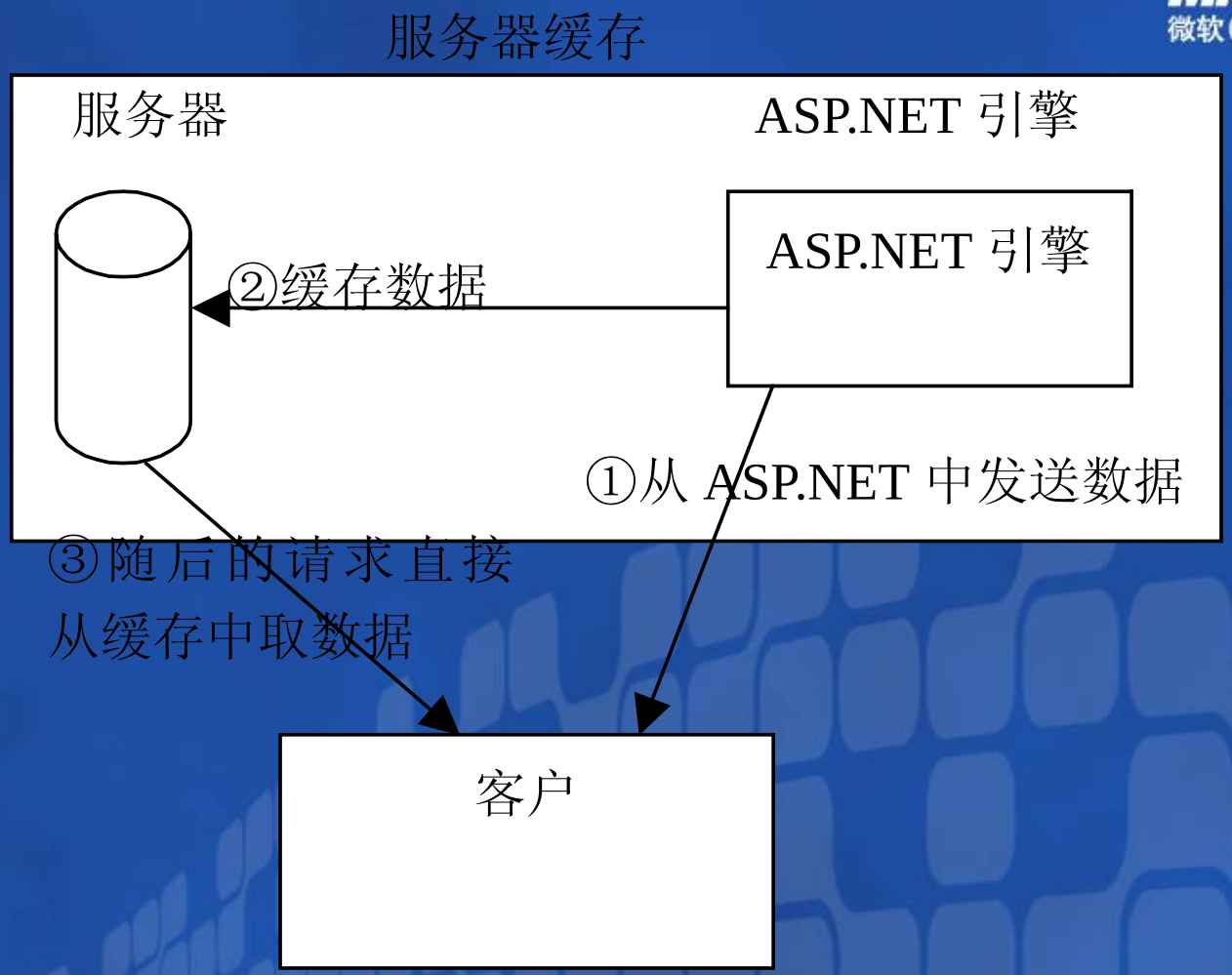
缓存是计算机快速地再次获得数据地方式。

2、缓存原理

将经常访问地数据存储到计算机可以更快、更容易地读取地位置。

3、ASP.NET中缓存的位置





4、什么时候用缓存？

使用缓存的情况	不应该使用缓存的情况
缓存那些经常被访问、并且变化不大的数据	不要缓存个人信息，以防止别人盗用
缓存整个应用程序都要使用的设置或对象（但这些设置和对象必须在其生存期内不变化）	不要缓存包含时间的页面
	不要缓存用户随时都会修改的对象，如购物车

5、如何使用缓存？

- ASP.NET有两种用于WEB应用的缓冲技术：输出缓冲和数据缓冲。
  - 输出缓冲指：把一次请求所产生的动态输出保存于内存中。
  - 数据缓冲指：按照一定的策略把事先不确定的对象保存于内存中。
- 输出缓存的使用
  - 使用@OutputCache指令
  - 例如(添加在页头)

```
<%@ OutputCache Duration="10" VaryByParam="None" %>
```

DEMO4

输出缓存练习

数据缓存

- ASP.NET提供了一个相当出色的缓存引擎机制，它允许页面保存和索引HTTP请求所要求的各种各样的对象。ASP.NET的缓存对各个应用来说是私有的，是存储各种对象的存储器。缓存的生存周期取决于应用的生存周期，也就是说，当应用重新启动时，缓存实际上也已重建。

- 数据缓冲
  - 使用（类似于Session变量的使用）

    Cache[“userName”] = “MeMe”;

    Response.Write(Cache(“userName”));
  - 注意不能通过下标访问缓存中的变量，如

    Response.Write(Cache[0]);是错误的。
  - 缓存的删除

    Cache.Remove(“userName”);

- 使用缓存依存关系
  - 缓存变量的添加
    - Cache.Add()
    - Cache.Insert()

    它们功能相同，但Insert更加灵活一些
  - Insert（key,value,dependencies,absoluteExpiration, slidingExpiration,priority,priorityDecay,onRemoveCallBack）

缓存替换策略

1. “腐烂搜索”（Scavenging）
   - 当内存变得比教紧张时，缓存机制会找出最不常用和最不重要的对象，把它从内存中移出，以减轻系统压力。
2. “到期控制”（Expiration）
   - 编程者可以指定缓存对象的生存周期，这种指定的时间可以是绝对的也可以是相对的。
3. “文件和键值依赖”
   - 从外部文件或者是其他缓存键值是否改变，来决定本身键值是否有效。

DEMO5

数据缓存

议程

- 错误类型以及处理方式
- 提高数据访问性能
- 服务器控件的使用
- 使用缓存进行程序优化
- 提高性能的实用技巧

提高性能的实用技巧

- 不要使用不必要的Session，和ASP中一样，在不必要的时候不要使用Session
- 不使用不必要的Server Control
- 不使用不必要的ViewState
- 不要用Exception控制程序流程
- 禁用VB和Jscript动态数据类型
- 使用存储过程完成数据访问
- 只读数据访问不要使用DataSet
- 关闭ASP.NET的Debug模式
- 使用ASP.Net Output Cache缓冲数据

提高性能的实用技巧

- 尽量用SQL返回DataGrid需要绑定的DataSet,尽量不要对DataSet进行二次加工，特别不要对DataSet进行大量删除，实践证明这很慢。不如复制部分数据。
- 尽量把查询数据的数据库操作次数压缩到最少，尽量1-2次数据库操作就可完成；
- 注意优化数据库查询操作
- 不要在页面加载时默认选择全部数据，尽管可以方便后续操作，但用户会以为“还没有操作就这么慢”
- 建议尽量用比较高效的SQL代替后续复杂的DataSet二次加工

提高性能的实用技巧

- 仅在需要的时候打开数据库连接
- 一旦数据库操作完毕，一定关闭连接
- 在关闭连接时记得删除临时对象
- 在关闭连接前，确保关闭任何用户定义事务
- 显示非交互性数据，使用SQLDataReader可以获得最佳性能
- 注意共享那些经过复杂处理或漫长查询才得到的数据
- 在页面跳转时记得终止当前页面的处理
- 有大量连接的字符串操作不要使用+，改用StringBuilder

小结

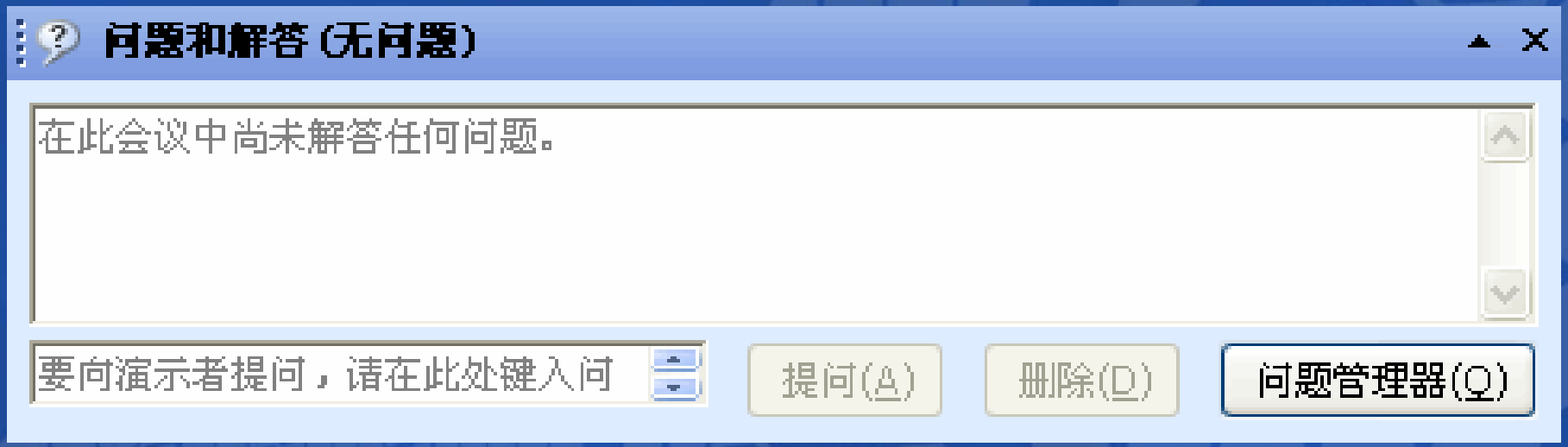
- 错误类型以及处理方式
- 提高数据访问性能
- 服务器控件的使用
- 使用缓存进行程序优化
- 提高性能的实用技巧

更多信息.....

- MSDN网站： http://msdn.microsoft.com
- 程序员大本营:**www.csdn.net**

Question & Answer

如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。




您的潜力，我们的动力
Microsoft
微软（中国）有限公司
Microsoft
msdn
MSDN Webcasts