您的潜力，我们的动力

Microsoft
微软（中国）有限公司

开发高性能的ASP.NET应用

王兴明
wmango@hotmail.com
微软认证金牌讲师（MCT）
微软最有价值专家（MVP）

msdn

MSDN Webcasts

议程

- 性能概述
- ASP.NET 性能最佳实践
  - 逻辑/物理设计实践
  - 语言/代码建议
  - COM 交互建议
  - 数据建议
  - 状态管理
  - 使用缓存

性能是一项功能

- 设计时就考虑性能
- 不要在事后再加入性能!
- 在项目开发的整个过程中反复测试
- 两种量化Web性能的方法:
  1) 机器吞吐率 (requests/sec)
  2) 响应时间 (time to first/last bytes)

测试Web 性能

- 通过给服务器加负载来测试
  - 使用多台客户端机器加载
  - 免费Microsoft Web Application Stress 工具
  - 测试多种场景:
    - 模拟对站点的端对端场景遍历
    - 测试单页性能
- 需计算的数值:
  - 不同负载下的Request/sec (100, 250, 500, 750, 1000等等同时访问的客户)
  - 辨明在可接受TTFB/TTLB 响应时间内的最大客户负载

关键的性能计数器

- 处理器, CPU % 使用率
  - 低数值 = 受阻或者锁竞争
- ASP.NET, 入列请求数
  - 线性增长意味着服务器已满负荷
- ASP.NET 应用, 每秒请求数
  - 动态吞吐量 (应当一致)
- ASP.NET 应用, 总错误数
  - 指示功能错误 (应当是0)
- ASP.NET 应用, 工作进程重启
  - 指示严重功能错误

议程

- 性能概述
- ASP.NET 性能最佳实践
  - 逻辑/物理设计实践
  - 语言/代码建议
  - COM 交互建议
  - 数据建议
  - 状态管理
  - 使用缓存

最佳实践

- “干净”的代码
  - 而不是拼凑在一起的方案
  - 更容易优化
  - 更容易维护
- 遵循好的设计实践:
  - 逻辑/物理设计
  - 语言选择
  - COM 交互
  - 数据访问
  - 输出缓存

议程

- 性能概述
- ASP.NET 性能最佳实践
  - 逻辑/物理设计实践
  - 语言/代码建议
  - COM 交互建议
  - 数据建议
  - 状态管理
  - 使用缓存

逻辑实践

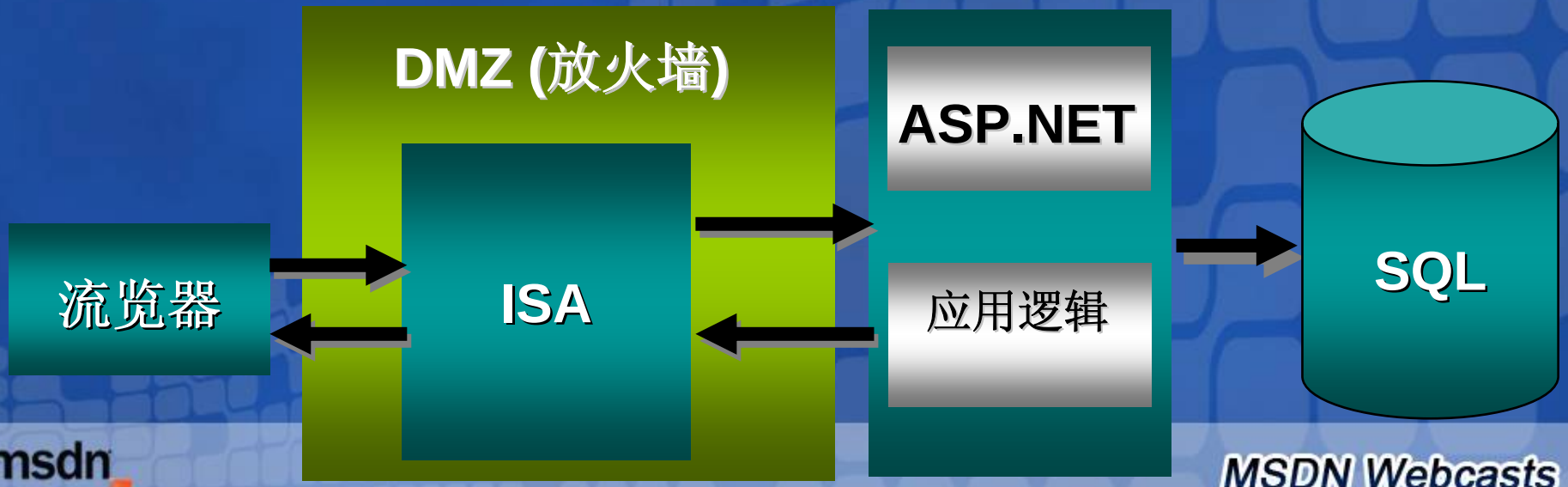
- 推荐: 使用逻辑三层模型
  - 页面 (.aspx) 和用户控件 (.ascx)
  - 可重用类型 (组件) 放在 \bin 目录下
  - 数据放在 SQL 数据库中
- 推荐: 为 Web Farm设计
  - 不要假定 访问者永远会回到同一服务器
  - 注意静态变量和应用状态
  - Web Farm会话状态

物理部署

- 推荐: 用同一个进程
- 应避免 (*可能时*):
  - 对 XML Web 服务的同步调用
  - 通过 DCOM的远程调用
- 使用XML Web 服务:
  - 因特网上应用之间的通讯
  - 不要用于应用内通讯

物理部署

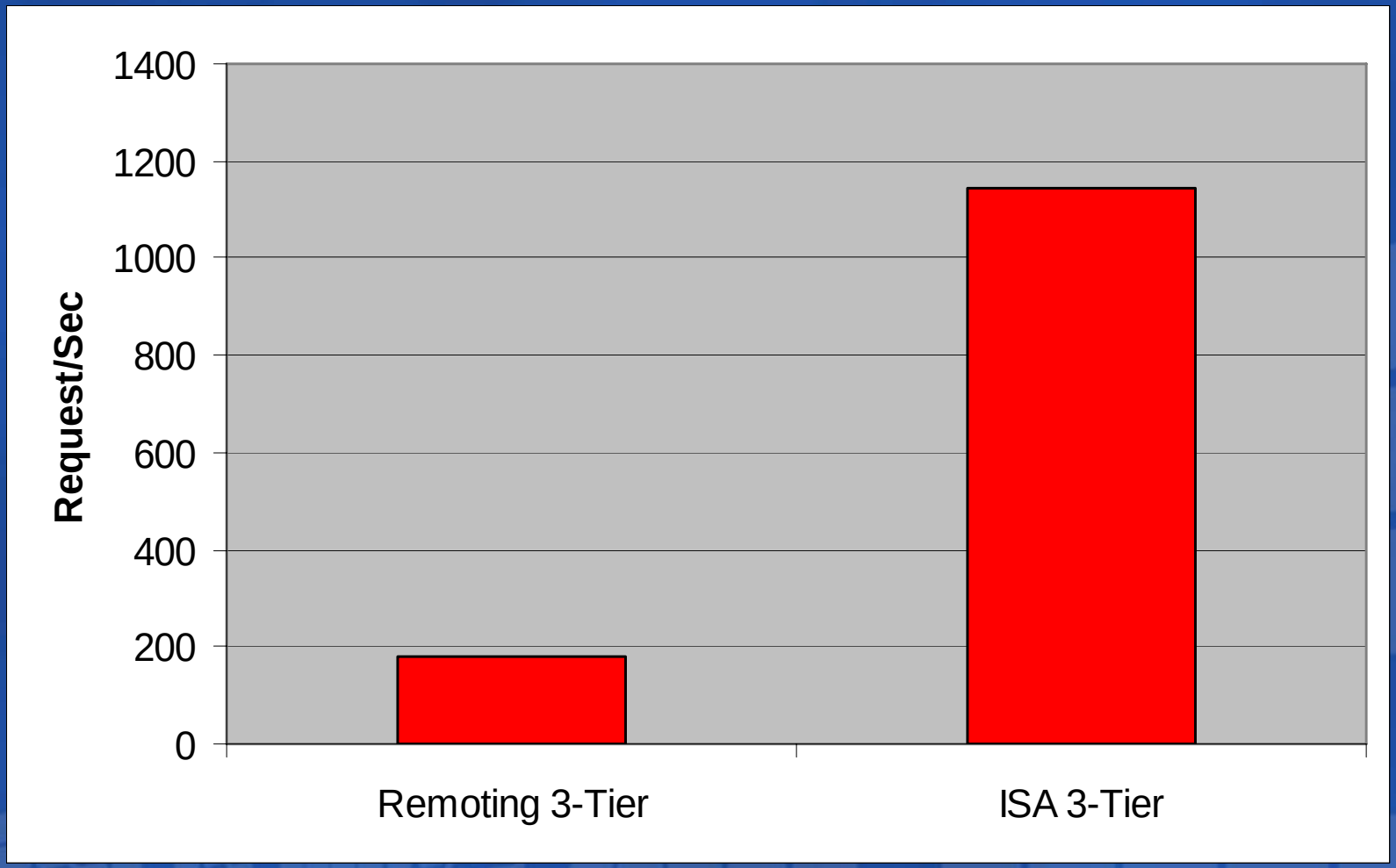
- 推荐: 使用ISA 服务器
- 当需要 DMZ 安全域时:
  - 只允许通过ISA访问
  - ISA 隧道通过DMZ 到达ASP.NET
  - 减少进程转换



DMZ 性能指标

- 测试场景:
  - 物理多机器部署
- 设计目标:
  - 不能访问应用服务器
- 测试两种技术:
  - ASP.NET 使用远程调用访问逻辑
  - ISA 前端访问包含页面和商务逻辑的ASP.NET服务器

三层部署性能



议程

- 性能概述
- ASP.NET 性能最佳实践
  - 逻辑/物理设计实践
  - 语言/代码建议
  - COM 交互建议
  - 数据建议
  - 状态管理
  - 使用缓存

代码性能

- .NET 通用语言运行环境 (CLR)
  - 运行性能大幅提高
  - 运行时JIT 编译到本机代码
  - 优化的垃圾回收器
- 简单问题: 代码分离还是混合?
  - 没有性能差别
- 简单问题: VB .NET 或者 C#?
  - 没有可觉察的差别，然而…

语言选择

- *相同*代码提供相同性能

...然而，你可以编写不同代码:

```
‘ VB 晚绑定例子
Dim Count
For Count = 0 to 7
     Response.Write(“Count: “ & Count)
Next Count
```

```
‘ VB 早绑定例子
Dim Count as Integer
For Count = 0 to 7
     Response.Write(“Count: “ & Count)
Next Count
```

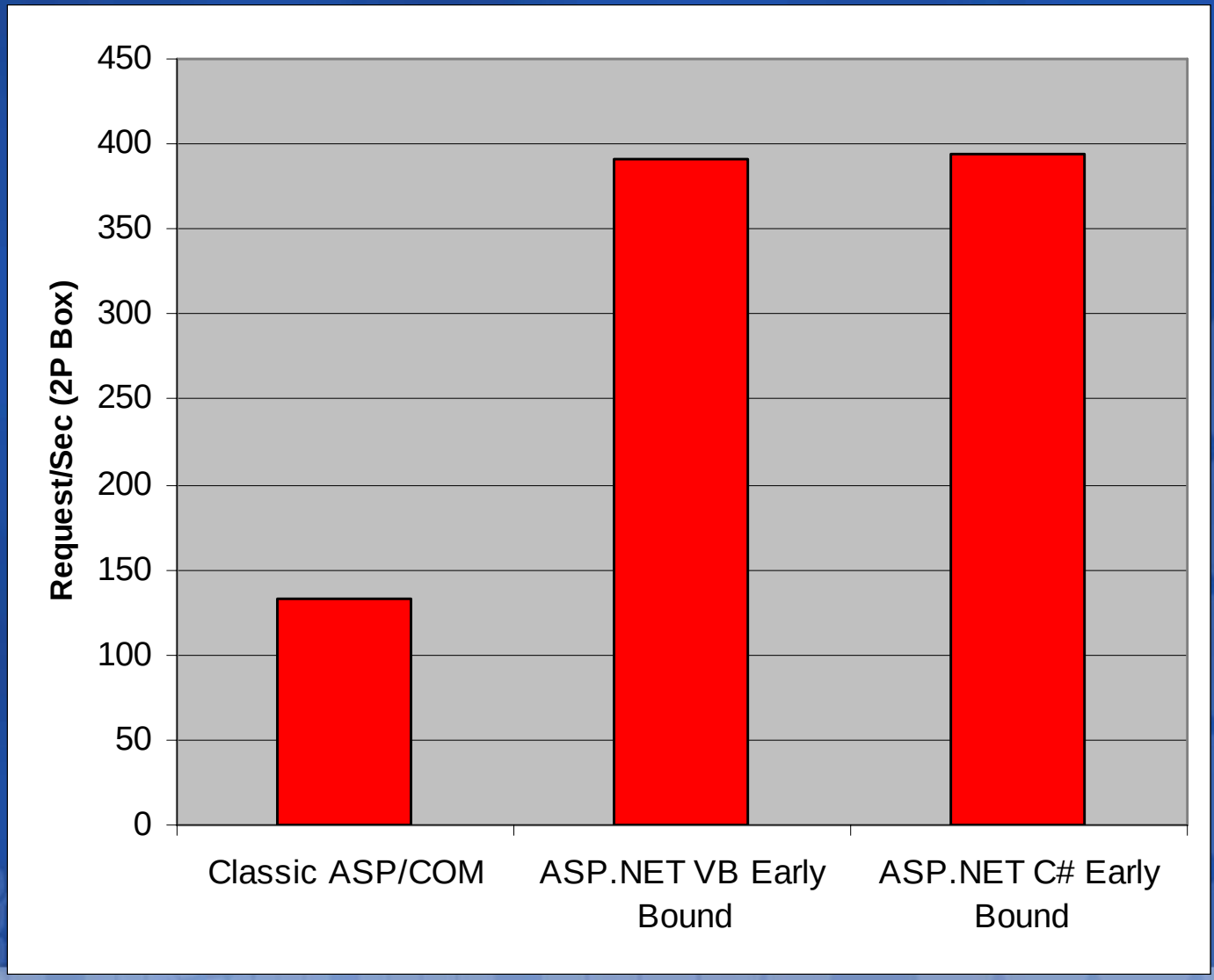
语言建议

- 建议: 避免晚绑定
- 在 VB 和 JScript中
  - 显式定义所有变量类型(Dim a as Long)
  - 避免使用Object类型的晚绑定方法
- <%@ Page Language="VB" Explicit="true" %>
  - 要求声明所有变量类型(要求DIM)
  - 依然允许晚绑定
- <%@ Page Language="VB" Strict="true" %>
  - 禁止使用任何晚绑定

语言性能指标

- 测试场景:
  - MSN 主页 (大约 3000 行代码)
  - 没有数据访问或者网络调用
  - COM或者.NET商务组件
- 测试3种技术:
  - ASP/COM 实现
  - VB.NET早绑定
  - C#.NET早绑定

MSN 性能测试



议程

- 性能概述
- ASP.NET 性能最佳实践
  - 逻辑/物理设计实践
  - 语言/代码建议
  - COM 交互建议
  - 数据建议
  - 状态管理
  - 使用缓存

COM 交互

- 建议: 尽量避免交互
- 最好将COM代码移植到.NET
  - 可能会很昂贵，尤其对于数组操作
  - “受管” 与 “不受管” 转换
- 如果你必须做交互重新设计API
  - 少一些“啰唆”
  - 多一些“块头”
- 注意单元线程组件
  - 缺省情况 ASP.NET 使用MTA线程
  - 损害STA 组件性能

注意所有VB6组件!

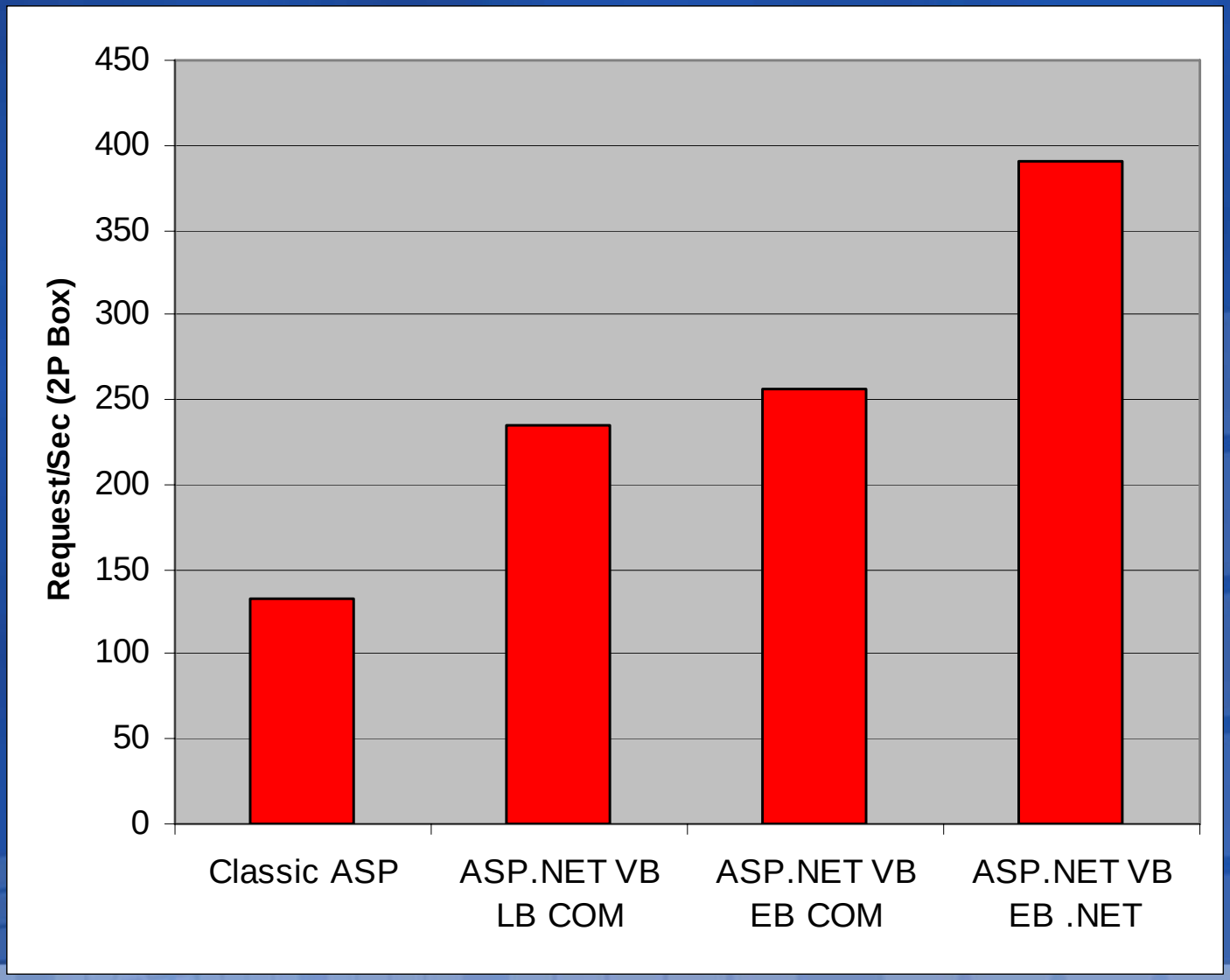
交互建议

- 标注页面让它运行在 STA 兼容模式:
  - 如果组件是单元线程模式 (VB6)
  - <%@ Page ASPCompat=“true” %>
- 影响
  - 对 ASP.NET性能有负面影响
  - 无法进行XCOPY部署
- 产生早绑定 .NET 包装
  - 使用TLBIMP.exe
  - 依然要求ASPCompat

性能指标

- 测试场景:
  - MSN 主页 (大约 3000 行代码)
  - 没有数据访问或者网络调用
  - COM或者.NET商务组件
- 测试4种技术:
  - ASP/COM 实现
  - VB.NET晚绑定
  - VB.NET早绑定
  - VB.NET/纯.NET组件

MSN 性能测试



议程

- 性能概述
- ASP.NET 性能最佳实践
  - 逻辑/物理设计实践
  - 语言/代码建议
  - COM 交互建议
  - 数据建议
  - 状态管理
  - 使用缓存

数据访问

- 编写良好的代码对性能至关重要
  - 不要受阻在数据库服务器上
  - 简单事项，比如关闭数据库连接
- 数据代码检查表：
  - 分析并优化数据表索引
  - 利用ADO.NET的输出参数
- 推荐：使用存储过程
  - 可以消除数据库的往返访问
  - 通过使用数据库事务而避免分布式事务的耗费

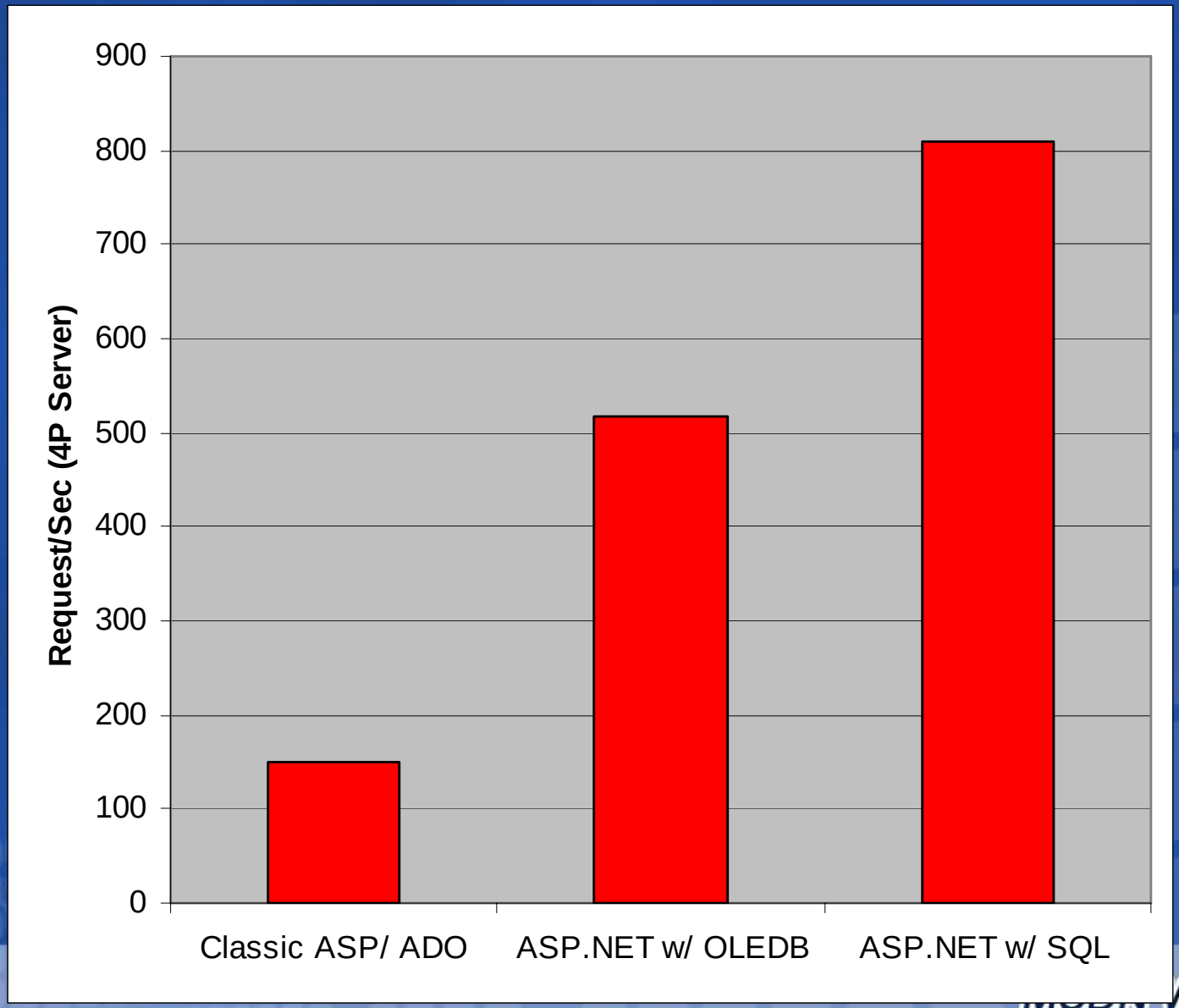
.NET 数据访问

- .NET中多种数据访问方法
  - SQL vs. OLEDB vs. ODBC数据提供者
  - DataReaders vs. DataSet
- 数据提供者选择:
  - 使用SQL服务器时使用Data.SqlClient中的类型
- 推荐:
  - DataReader: 用于只进的只读访问
  - DataSet: 用于缓存或者Web服务

数据提供者测试

- 场景:
  - 访问SQL Northwinds 数据库中50行数据
  - 用<%= %> html表输出数据
  - 不同的数据提供者 (ADO 以及ADO.NET)
- 测试3种技术:
  - ASP/ADO
  - ASP.NET 与 System.Data.OleDb 提供者
  - ASP.NET 与 System.Data.SqlClient 提供者

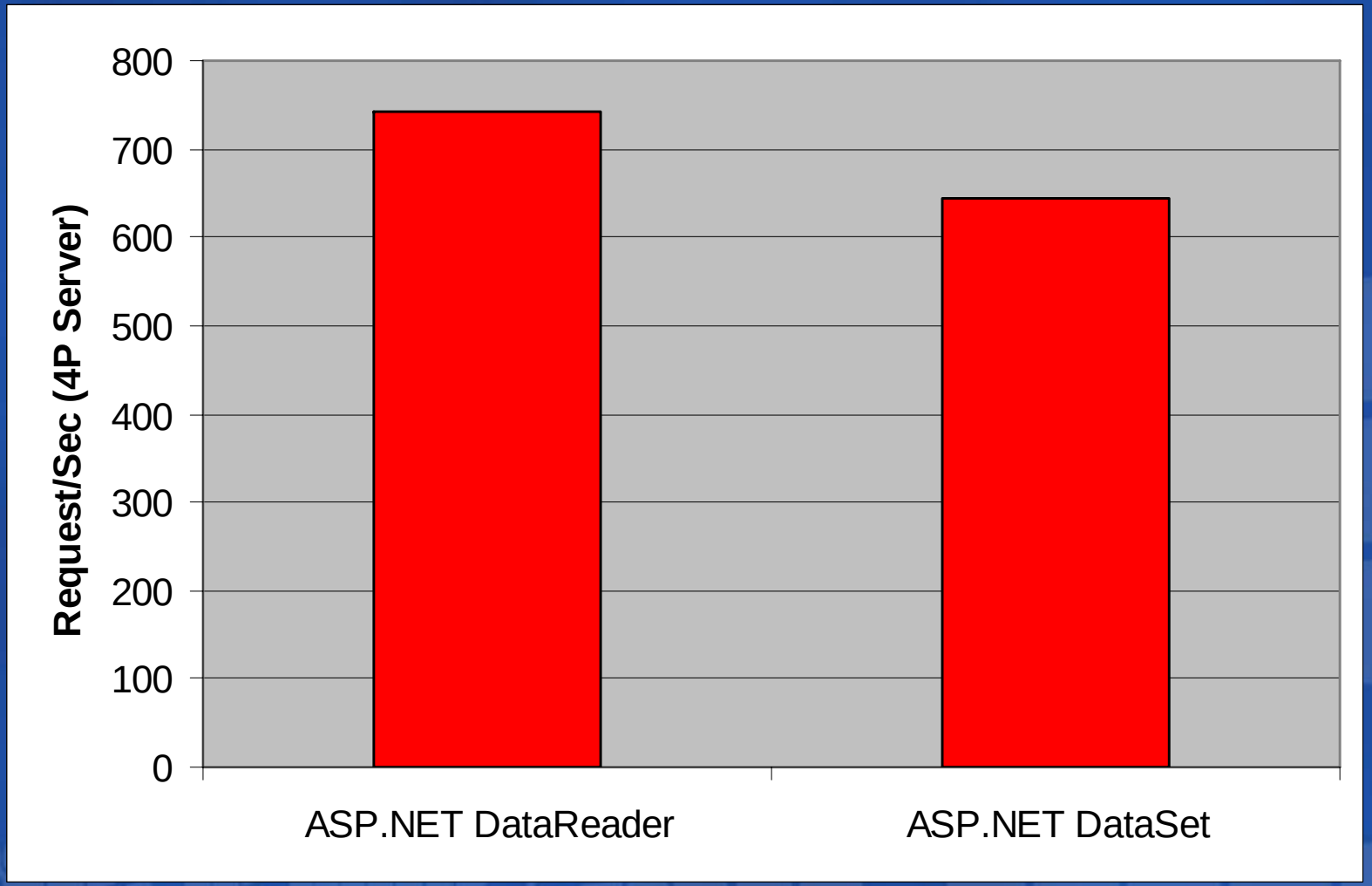
数据性能测试



DataSet/DataReader 性能测试

- 场景:
  - 从Pubs数据库的Authors表中输出23行
  - 用<%= %> html 表输出数据
  - System.Data.SqlClient 数据提供者
- 测试2种技术:
  - ASP.NET 和 Sql Provider 以及 DataSet
  - ASP.NET 和 Sql Provider 以及 DataReader

DataSet/DataReader 性能测试



议程

- 性能概述
- ASP.NET 性能最佳实践
  - 逻辑/物理设计实践
  - 语言/代码建议
  - COM 交互建议
  - 数据建议
  - 状态管理
  - 使用缓存

状态管理

- 客户端状态管理技术
  - Client cookies
  - Query strings
  - Hidden controls
- 服务器端状态管理技术
  - Application
  - Session
  - Database

Store Simple State on the Client Where Possible

Applicatoin State

- 使用静态属性而不是对象来操作应用状态

```
private static string[] _states[];
public static string[] States
{
    get {return _states;}
}
public static void PopulateStates()
{
        //ensure this is thread safe
        if(_states == null)
        {
                lock(_lock)
                {
                    //populate the states...
                }
        }
}
```

- 尽量在Application中只保存静态的只读数据
- 避免在Applicatoin中保存STA的COM对象

Session State

- 选择合适的方式存放Session
  - Inproc/StateServer/database
- 使用简单数据类型
- 避免在Session中保存STA的COM对象
- 如果不需要Session，则禁用它
  - <%@ Page EnableSessionState="false" .. .%>
- 如果可能，可使用readonly特性
  - <%@ Page EnableSessionState="ReadOnly" .%>

View State

- 如果不需ViewState，则禁用它
  - EnableViewState=false;
- 最小化ViewState中的数据
- 查看ViewState的大小
  - 查看HTML输出源码
  - 使用Trace

议程

- 性能概述
- ASP.NET 性能最佳实践
  - 逻辑/物理设计实践
  - 语言/代码建议
  - COM 交互建议
  - 数据建议
  - 状态管理
  - 使用缓存

ASP.NET 缓存

- 页面缓存
- 部分页面缓存
- Cache对象
- 数据缓存
- WebSevice缓存

页面输出缓存

- 页面缓存
  - 第一次请求总是被动态处理
  - 之后的请求由缓存服务
- 支持两种 APIs
  - 高层, `<%@ OutputCache … %>`
  - 低层, `Response.Cache`
- 可以极大提高性能
  - 大多数情况下提高2-3倍

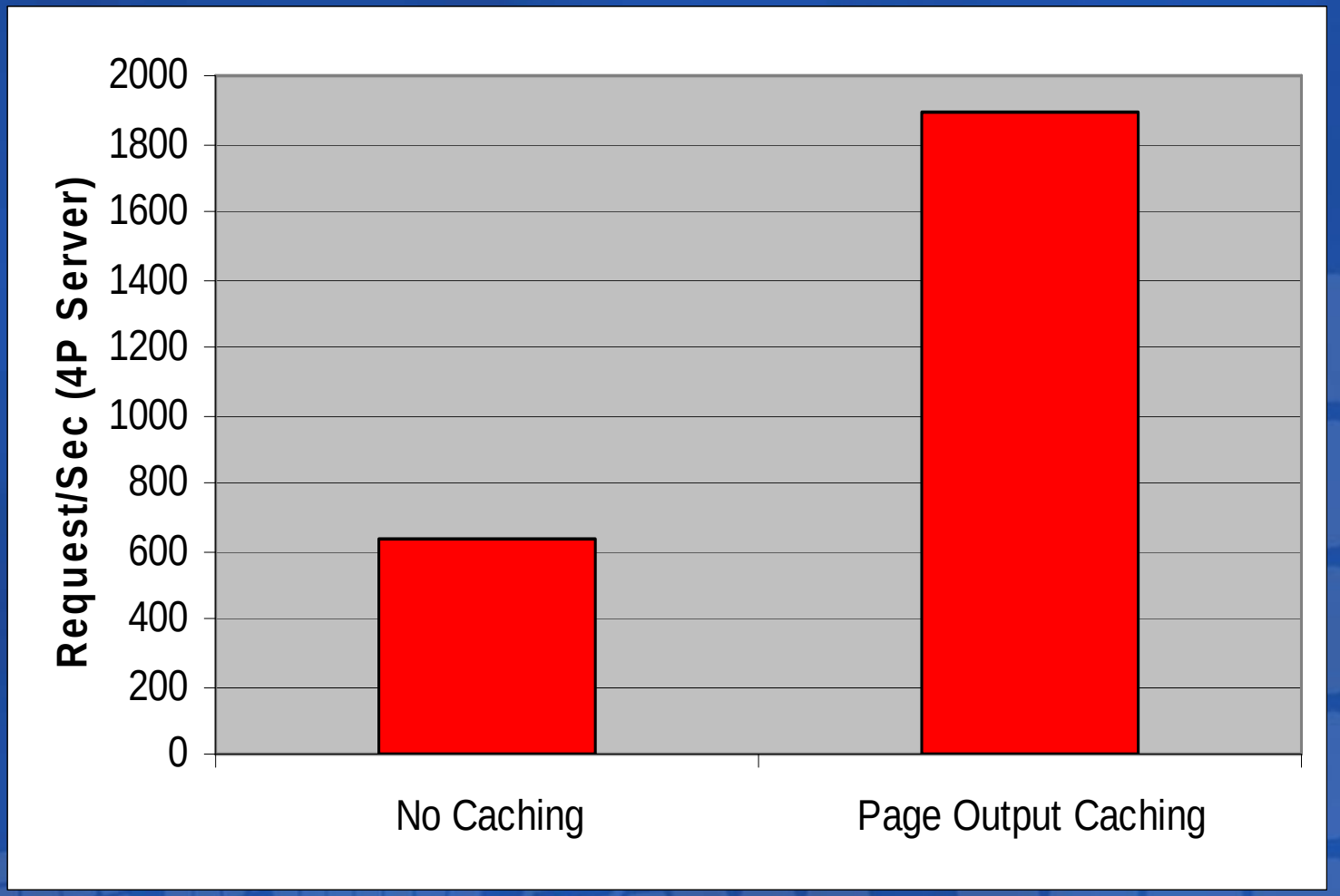
输出缓存指令

- Duration
  - 数据在缓存中有效时间
- VaryByParam
  - GET/POST 值, ‘*’ (所有), 以及‘none’
- VaryByHeader
  - 根据 HTTP 头部而变化
- VaryByCustom
  - 根据 ‘browser’ (类型和主版本值)
- 用VaryByCustom扩展
  - 重载 VaryByCustomString

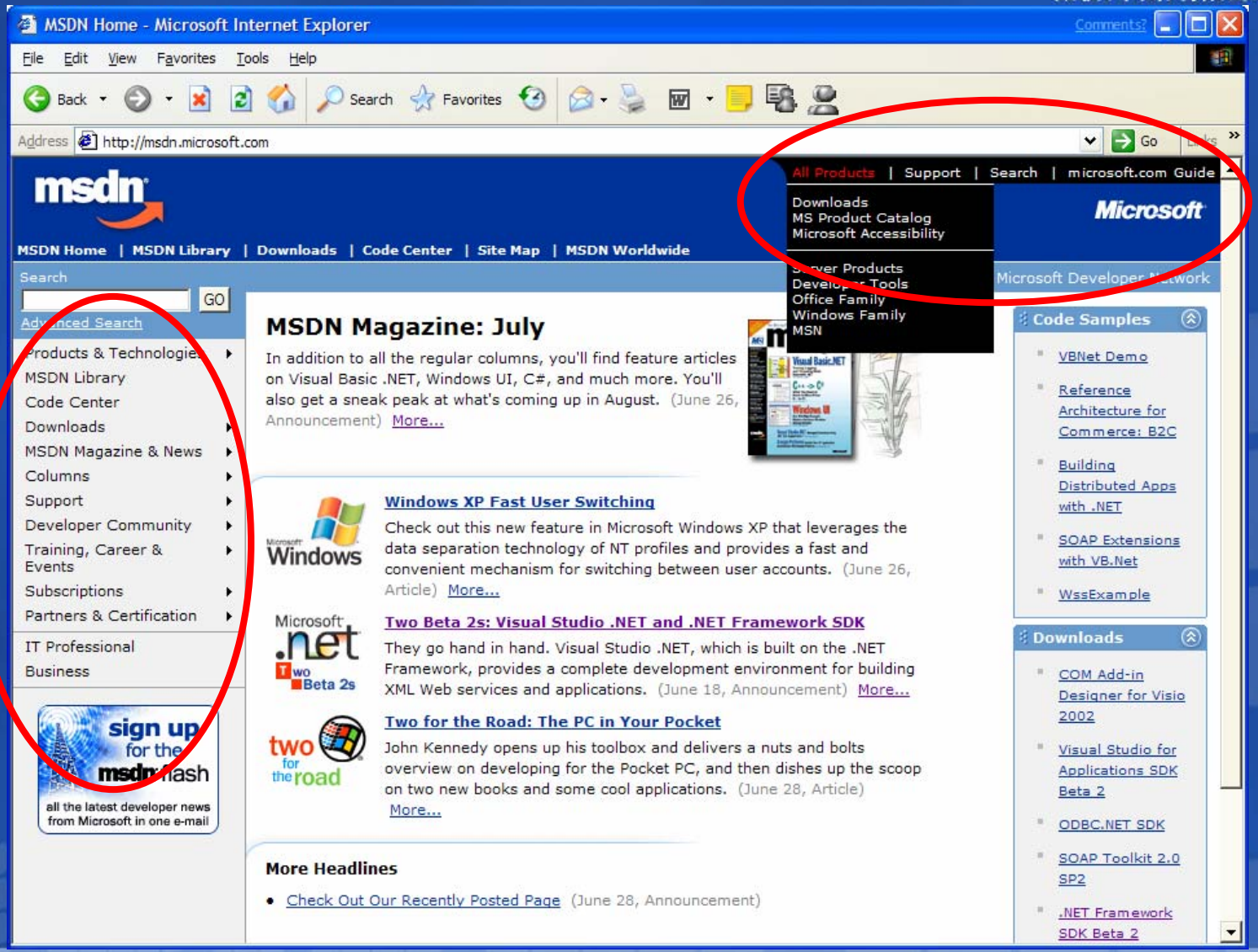
缓存性能测试

- 场景:
  - 用DataReader作标准的DataGrid数据绑定
- 测试2种技术:
  - 不用缓存
  - 使用缓存

缓存测试结果



部分页面缓存



部分页面缓存

- 页面缓存并不总是可行
  - 动态内容，比如个人化内容
  - 交互内容，比如广告
  - 依然可从输出缓存受益
- 部分页面缓存
  - 制定缓存页面中部分内容
  - 对于第一次请求完全执行
  - 使用用户控件

部分页面缓存

- 页面引用用户控件
  - 用户控件定义缓存区
  - 页面动态执行
- 运作类似于页面缓存
  - 使用 <%@ OutputCache … %> 指令
  - 使用于用户控件
- 使用 VaryByControl 功能
  - 有点类似于VaryByParam
  - 用户控件中的控件唯一辩识

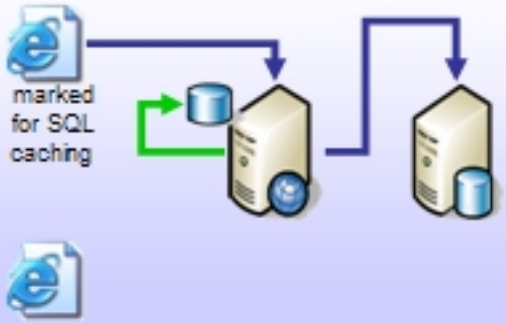
数据缓存

- 如果想缓存数据库查询的结果而不是静态的页面输出，怎么办？
- 当数据库中数据发生变化后，缓存的数据立即失效，重新从数据库中取最新的数据。

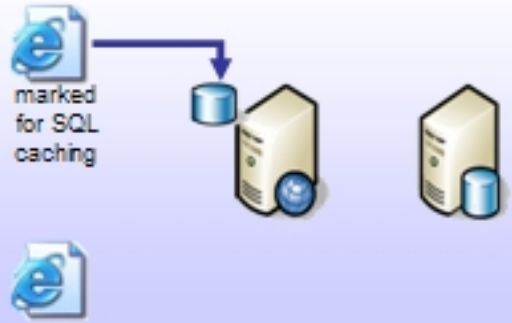
数据缓存工作过程（SQL7.0,2000）

您的潜力，我们的动力
Microsoft
微软（中国）有限公司

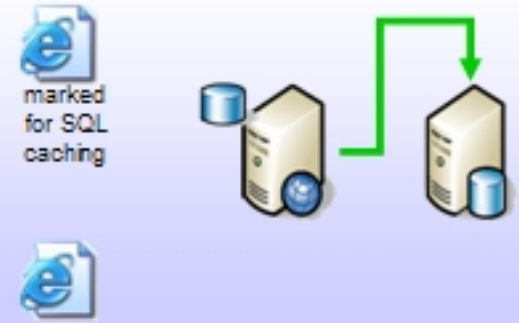
1 You have a web page that is enabled with SQL 7/2000 dependency caching. The first time it is executed it will be compiled and it will access the database, but it is then cached.



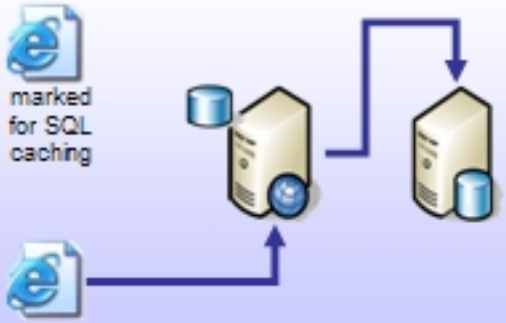
2 The second and subsequent instances that the page is executed, it is served from the ASP.NET cache until a change occurs to the table it is dependent on.



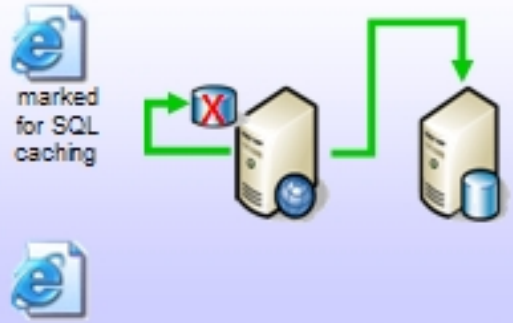
3 ASP.NET periodically monitors the SQL table to determine whether or not changes are occurring. This is an automatic process that is enabled when you setup caching.



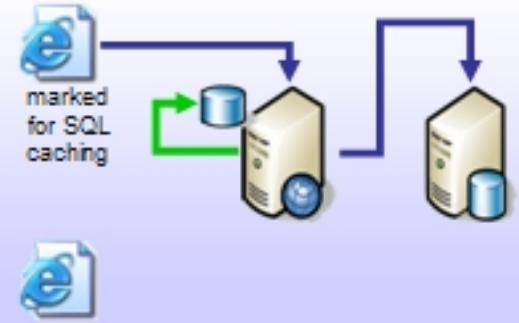
4 The database table being monitored is modified by the execution of a different web page. The database update could have been from any type of application in your organization.



5 ASP.NET is continuously monitoring if changes have occurred within your table. A change is found and the dependent cache is automatically invalidated.



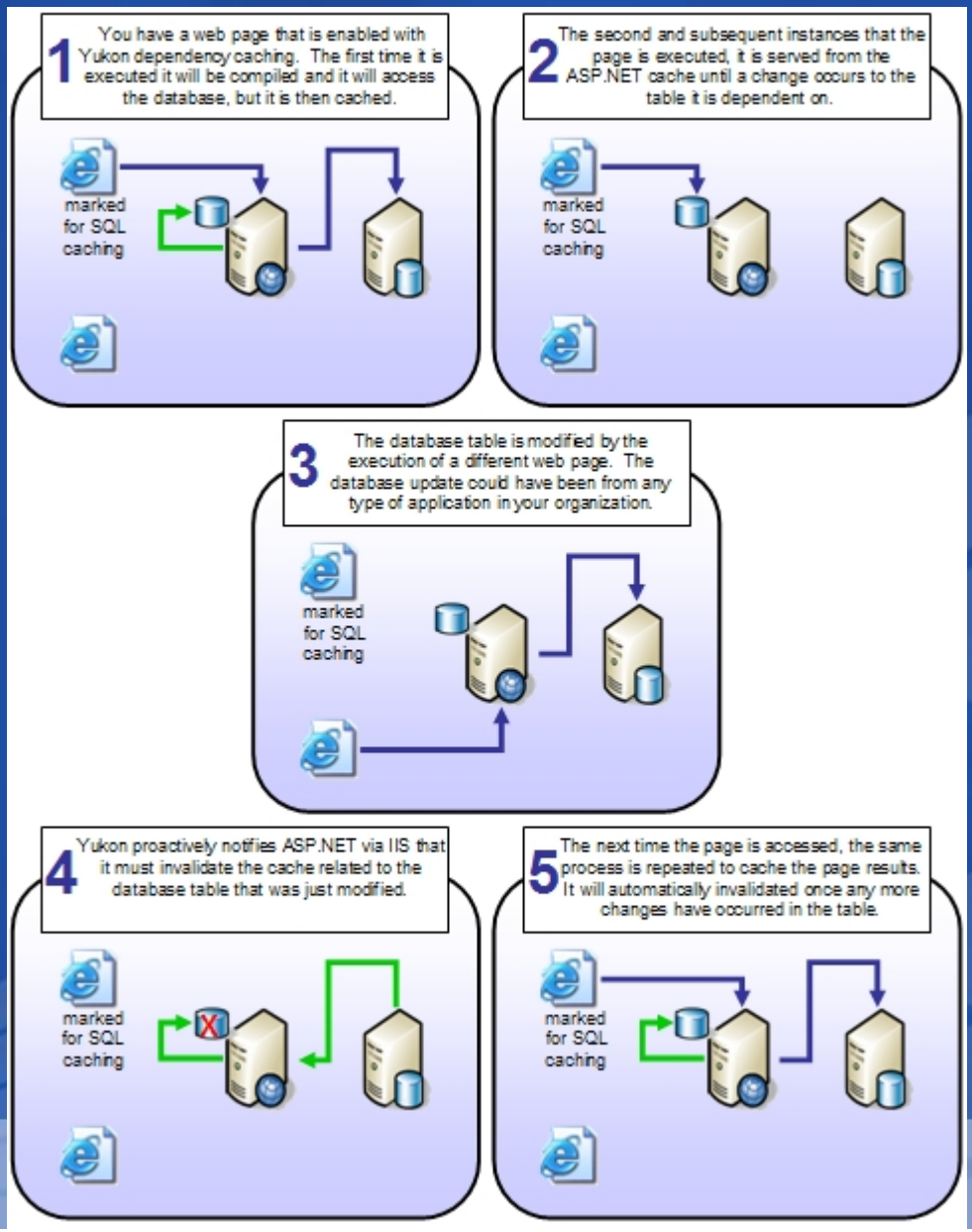
6 The next time the page is accessed, the same process is repeated to cache the page results. It will automatically invalidated once any more changes have occurred in the table.



数据缓存工作过程（SQL7.0,2000）

- Enable notifications for the database
  - aspnet_regsql -S "your-machine-name\sqlexpress" -E -d pubs -ed
- Enable notifications for the table(s) you want to have dependencies on
  - aspnet_regsql -S "your-machine-name\sqlexpress" -E -d pubs -t authors -et
- Web.config
  - <caching>
    <sqlCacheDependency enabled="true" pollTime="5000">
    <databases>
    <add name="pubs" connectionStringName="AppConnectionString1" />
    </databases>
    </sqlCacheDependency>
    </caching>
- Page.aspx
  - <%@ OutputCache Duration="3600" VaryByParam="none" SqlDependency="pubs:authors"%>

数据缓存工作过程（SQL2005）



数据缓存工作过程（SQL2005）

- 声明方式
  - On a page:
    - <%@ OutputCache Duration="999999" SqlDependency="CommandNotification" VaryByParam="none" %>
  - On a datasource control:
    - <asp:SqlDataSource EnableCaching="true" SqlCacheDependency="CommandNotification" CacheDuration="1000" ... />
- 编程方式

```
Sqlconnection conn=new Sqlconnection(connString);
Sqlcommand cmd=new SqlCommand("select * from authors",conn);
SqlCacheDependency sqlDependency=new SqlCacheDependency(cmd);
this.GridView1.Datasource=cmd.ExecudeReader();
this.GridView1.DataBind();
conn.Close();

Responce.AddCacheDependency(sqlDependency);
Response.Cache.SetValidUntilExpires(True);
Response.Cache.SetExpires(DateTime.Now.AddMinutes(60));
```

总结

1. 使用逻辑3层部署
2. 为 Web Farm而设计
3. 使用同一进程
4. 使用ISA服务器
5. 避免晚绑定
6. 尽可能避免使用COM交互
7. 对VB6组件使用ASPCompat模式
8. 使用存储过程
9. 对于一般数据访问使用DataReader
10. 恰当地处理状态
11. 使用缓存设计

获取更多MSDN资源

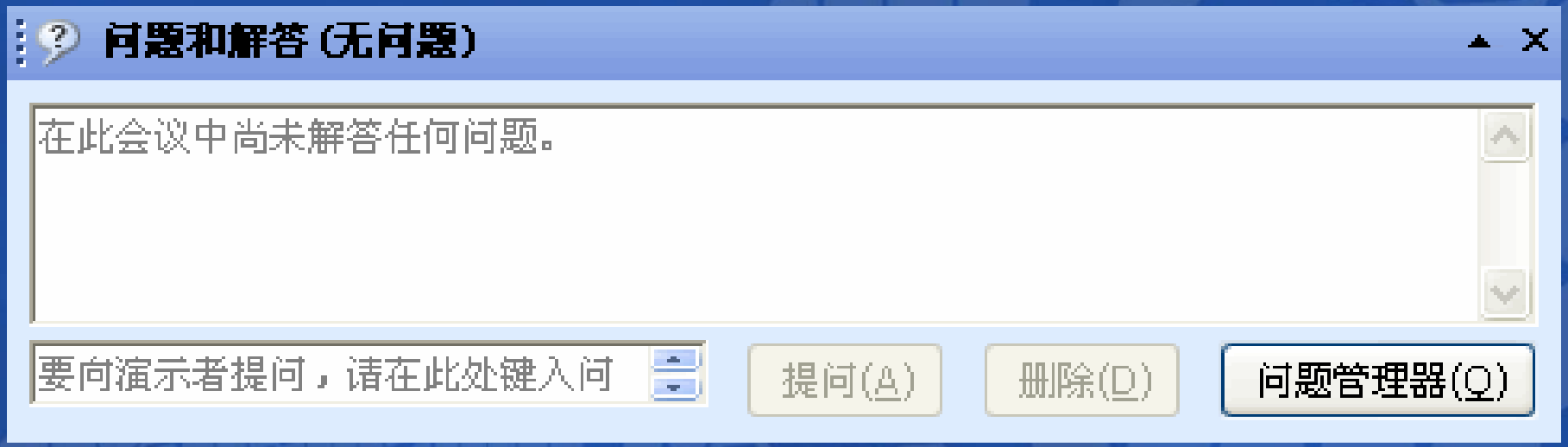
您的潜力，我们的动力

Microsoft
微软（中国）有限公司

- **MSDN中文网站**
  http://www.microsoft.com/china/msdn
- **MSDN中文网络广播**
  http://www.msdnwebcast.com.cn
- **MSDN Flash**
  http://www.microsoft.com/china/newsletter/case/msdn.aspx
- **MSDN开发中心**
  http://www.microsoft.com/china/msdn/DeveloperCenter/default.mspx

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