相关主题 RECOMMEND ARTICLE
- 你必须知道的.NET之对接口和抽象类
- ASP.Net中利用CSS实现多界面两法
- 将ASP页面转换成HTM静态页面的方法
- ASP.NET技术获取IP与MAC地址的方法
- ASP.NET移动开发之SelectionList控件
- ASP.NET中为GridView添加删除提示框
- 理解ASP.NET与客户端缓存之HTTP协议
- ASP.NET中Session的状态保持方式浅议

MORE

推荐文章 RECOMMEND ARTICLE
- 数据广播方案的优化
- 网络游戏的位置同步
- 游戏音乐制作案例之《战火 红色警戒》音效制作揭秘
- 英雄连Online 原画
- 游戏音乐制作案例之《乱武天下》
- 游戏音乐制作案例之《诛仙》
- 《鹿鼎记》最新原画
- MIDP2.1规范的新特性

MORE

热门文章 HOT ARTICLE
- [电子书下载]游戏设计 — 原理与实践
- [电子书下载]网络游戏开发
- 游戏设计全过程
- [电子书下载]游戏设计技术
- [电子书下载]游戏设计理论
- CS游戏人物模型制作教程
- CG人物插画基本流程
- [转贴]MAX高级人头教程

MORE

您的位置： .NET

文章标题	ASP.NET中常用的优化性能方法		
来源：	[ogdev]	浏览：	[420]

1．数据库访问性能优化

数据库的连接和关闭

访问数据库资源需要创建连接、打开连接和关闭连接几个操作。这些过程需要多次与数据库交换信息以通过身份验证，比较耗费服务器资源。ASP.NET中提供了连接池(Connection Pool)改善打开和关闭数据库对性能的影响。系统将用户的数据库连接放在连接池中，需要时取出，关闭时收回连接，等待下一次的连接请求。

连接池的大小是有限的，如果在连接池达到最大限度后仍要求创建连接，必然大大影响性能。因此，在建立数据库连接后只有在真正需要操作时才打开连接，使用完毕后马上关闭，从而尽量减少数据库连接打开的时间，避免出现超出连接限制的情况。

使用存储过程

存储过程是存储在服务器上的一组预编译的SQL语句，类似于DOS系统中的批处理文件。存储过程具有对数据库立即访问的功能，信息处理极为迅速。使用存储过程可以避免对命令的多次编译，在执行一次后其执行规划就驻留在高速缓存中，以后需要时只需直接调用缓存中的二进制代码即可。

另外，存储过程在服务器端运行，独立于ASP.NET程序，便于修改，最重要的是它可以减少数据库操作语句在网络中的传输。

优化查询语句

ASP.NET中ADO连接消耗的资源相当大，SQL语句运行的时间越长，占用系统资源的时间也越长。因此，尽量使用优化过的SQL语句以减少执行时间。比如，不在查询语句中包含子查询语句，充分利用索引等。

2．字符串操作性能优化

使用值类型的ToString方法

在连接字符串时，经常使用"+"号直接将数字添加到字符串中。这种方法虽然简单，也可以得到正确结果，但是由于涉及到不同的数据类型，数字需要通过装箱操作转化为引用类型才可以添加到字符串中。但是装箱操作对性能影响较大，因为在进行这类处理时，将在托管堆中分配一个新的对象，原有的值复制到新创建的对象中。

使用值类型的ToString方法可以避免装箱操作，从而提高应用程序性能。

运用StringBuilder类

String类对象是不可改变的，对于String对象的重新赋值在本质上是重新创建了一个String对象并将新值赋予该对象，其方法ToString对性能的提高并非很显著。

在处理字符串时，最好使用StringBuilder类，其.NET 命名空间是System.Text。该类并非创建新的对象，而是通过Append，Remove，Insert等方法直接对字符串进行操作，通过ToString方法返回操作结果。

其定义及操作语句如下所示：

以下是引用片段：
```
int num;
System.Text.StringBuilder str = new System.Text.StringBuilder(); //创建字符串
str.Append(num.ToString()); //添加数值num
Response.Write(str.ToString); //显示操作结果
```

3．优化 Web 服务器计算机和特定应用程序的配置文件以符合您的特定需要

默认情况下，ASP.NET 配置被设置成启用最广泛的功能并尽量适应最常见的方案。因此，应用程序开发人员可以根据应用程序所使用的功能，优化和更改其中的某些配置，以提高应用程序的性能。下面的列表是您应该考虑的一些选项。

仅对需要的应用程序启用身份验证。默认情况下，身份验证模式为 Windows，或集成 NTLM。大多数情况下，对于需要身份验证的应用程序，最好在 Machine.config 文件中禁用身份验证，并在 Web.config 文件中启用身份验证。

根据适当的请求和响应编码设置来配置应用程序。ASP.NET 默认编码格式为 UTF-8。如果您的应用程序为严格的 ASCII，请配置应用程序使用 ASCII 以获得稍许的性能提高。

考虑对应用程序禁用 AutoEventWireup。在 Machine.config 文件中将 AutoEventWireup 属性设置为 false，意味着页面不将方法名与事件进行匹配和将两者挂钩(例如 Page_Load)。如果页面开发人员要使用这些事件，需要在基类中重写这些方法(例如，需要为页面加载事件重写 Page.OnLoad，而不是使用 Page_Load 方法)。如果禁用 AutoEventWireup，页面将通过将事件连接留给页面作者而不是自动执行它，获得稍许的性能提升。

从请求处理管线中移除不用的模块。默认情况下，服务器计算机的 Machine.config 文件中 节点的所有功能均保留为

激活。根据应用程序所使用的功能，您可以从请求管线中移除不用的模块以获得稍许的性能提升。检查每个模块及其功能，并按您的需要自定义它。

例如，如果您在应用程序中不使用会话状态和输出缓存，则可以从 列表中移除它们，以便请求在不执行其他有意义的处理时，不必执行每个模块的进入和离开代码。

4．一定要禁用调试模式

在部署生产应用程序或进行任何性能测量之前，始终记住禁用调试模式。如果启用了调试模式，应用程序的性能可能受到非常大的影响。

5．对于广泛依赖外部资源的应用程序，请考虑在多处理器计算机上启用网络园艺

ASP.NET 进程模型帮助启用多处理器计算机上的可缩放性，将工作分发给多个进程(每个 CPU 一个)，并且每个进程都将处理器关系设置为其 CPU。此技术称为网络园艺。如果应用程序使用较慢的数据库服务器或调用具有外部依赖项的 COM 对象(这里只是提及两种可能性)，则为您的应用程序启用网络园艺是有益的。但是，在决定启用网络园艺之前，您应该测试应用程序在网络园中的执行情况。

6．只要可能，就缓存数据和页输出

ASP.NET 提供了一些简单的机制，它们会在不需要为每个页请求动态计算页输出或数据时缓存这些页输出或数据。另外，通过设计要进行缓存的页和数据请求(特别是在站点中预期将有较大通讯量的区域)，可以优化这些页的性能。与 .NET Framework 的任何 Web 窗体功能相比，适当地使用缓存可以更好的提高站点的性能，有时这种提高是超数量级的。

使用 ASP.NET 缓存机制有两点需要注意。首先，不要缓存太多项。缓存每个项均有开销，特别是在内存使用方面。不要缓存容易重新计算和很少使用的项。其次，给缓存的项分配的有效期不要太短。很快到期的项会导致缓存中不必要的周转，并且经常导致更多的代码清除和垃圾回收工作。若关心此问题，请监视与 ASP.NET Applications 性能对象关联的 Cache Total Turnover Rate 性能计数器。高周转率可能说明存在问题，特别是当项在到期前被移除时。这也称作内存压力。

7．选择适合页面或应用程序的数据查看机制

根据您选择在 Web 窗体页显示数据的方式，在便利和性能之间常常存在着重要的权衡。例如，DataGrid Web 服务器控件可能是一种显示数据的方便快捷的方法，但就性能而言它的开销常常是最大的。在某些简单的情况下，您通过生成适当的 HTML 自己呈现数据可能很有效，但是自定义和浏览器定向会很快抵销所获得的额外功效。Repeater Web 服务器控件是便利和性能的折衷。它高效、可自定义且可编程。

8．将 SqlDataReader 类用于快速只进数据游标

SqlDataReader 类提供了一种读取从 SQL Server 数据库检索的只进数据流的方法。如果当创建 ASP.NET 应用程序时出现允许您使用它的情况，则 SqlDataReader 类提供比 DataSet 类更高的性能。情况之所以这样，是因为 SqlDataReader 使用 SQL Server 的本机网络数据传输格式从数据库连接直接读取数据。另外，SqlDataReader 类实现 IEnumerable 接口，该接口也允许您将数据绑定到服务器控件。有关更多信息，请参见 SqlDataReader 类。有关 ASP.NET 如何访问数据的信息，请参见通过 ASP.NET 访问数据。

9．将 SQL Server 存储过程用于数据访问

在 .NET Framework 提供的所有数据访问方法中，基于 SQL Server 的数据访问是生成高性能、可缩放 Web 应用程序的推荐选择。使用托管 SQL Server 提供程序时，可通过使用编译的存储过程而不是特殊查询获得额外的性能提高。

10．避免单线程单元 (STA) COM 组件

默认情况下，ASP.NET 不允许任何 STA COM 组件在页面内运行。若要运行它们，必须在 .aspx 文件内将 ASPCompat=true 属性包含在 @ Page 指令中。这样就将执行用的线程池切换到 STA 线程池，而且使 HttpContext 和其他内置对象可用于 COM 对象。前者也是一种性能优化，因为它避免了将多线程单元 (MTA) 封送到 STA 线程的任何调用。

使用 STA COM 组件可能大大损害性能，应尽量避免。若必须使用 STA COM 组件，如在任何 interop 方案中，则应在执行期间进行大量调用并在每次调用期间发送尽可能多的信息。另外，小心不要在构造页面期间创建任何 STA COM 组件。例如下面的代码中，在页面构造时将实例化由某个线程创建的 MySTAComponent，而该线程并不是将运行页面的 STA 线程。这可能对性能有不利影响，因为要构造页面就必须完成 MTA 和 STA 线程之间的封送处理。

以下是引用片段：

首选机制是推迟对象的创建，直到以后在 STA 线程下执行上述代码，如下面的例子所示。

以下是引用片段：

推荐的做法是在需要时或者在 Page_Load 方法中构造任何 COM 组件和外部资源。

永远不要将任何 STA COM 组件存储在可以由构造它的线程以外的其他线程访问的共享资源里。这类资源包括像缓存和会话状态这样的资源。即使 STA 线程调用 STA COM 组件，也只有构造此 STA COM 组件的线程能够实际为该调用服务，而这要求封送处理对创建者线程的调用。此封送处理可能产生重大的性能损失和可伸缩性问题。在这种情况下，请研究一下使 COM 组件成为 MTA COM 组件的可能性，或者更好的办法是迁移代码以使对象成为托管对象。

11．将调用密集型的 COM 组件迁移到托管代码

.NET Framework 提供了一个简单的方法与传统的 COM 组件进行交互。其优点是可以在保留现有投资的同时利用新的平台。但是在某些情况下，保留旧组件的性能开销使得将组件迁移到托管代码是值得的。每一情况都是不一样的，决定是否需要迁移组件的最好方法是对 Web 站点运行性能测量。建议您研究一下如何将需要大量调用以进行交互的任何 COM 组件迁移

到托管代码。

许多情况下不可能将旧式组件迁移到托管代码，特别是在最初迁移 Web 应用程序时。在这种情况下，最大的性能障碍之一是将数据从非托管环境封送到托管环境。因此，在交互操作中，请在任何一端执行尽可能多的任务，然后进行一个大调用而不是一系列小调用。例如，公共语言运行库中的所有字符串都是 Unicode 的，所以应在调用托管代码之前将组件中的所有字符串转换成 Unicode 格式。

另外，一处理完任何 COM 对象或本机资源就释放它们。这样，其他请求就能够使用它们，并且最大限度地减少了因稍后请求垃圾回收器释放它们所引起的性能问题。

12. 在 Visual Basic .NET 或 JScript 代码中使用早期绑定

以往，开发人员喜欢使用 Visual Basic、VBScript 和 JScript 的原因之一就是它们所谓“无类型”的性质。变量不需要显式类型声明，并能够简单地通过使用来创建它们。当从一个类型到另一个类型进行分配时，转换将自动执行。不过，这种便利会大大损害应用程序的性能。

Visual Basic 现在通过使用 Option Strict 编译器指令来支持类型安全编程。为了向后兼容，默认情况下，ASP.NET 不启用该选项。但是，为了得到最佳性能，强烈建议在页中启用该选项。若要启用 Option Strict，请将 Strict 属性包括在 @ Page 指令中，或者，对于用户控件，请将该属性包括在 @ Control 指令中。下面的示例演示了如何设置该属性，并进行了四个变量调用以显示使用该属性是如何导致编译器错误的。

以下是引用片段：

JScript .NET 也支持无类型编程，但它不提供强制早期绑定的编译器指令。若发生下面任何一种情况，则变量是晚期绑定的：

被显式声明为 Object。

是无类型声明的类的字段。

是无显式类型声明的专用函数或方法成员，并且无法从其使用推断出类型。

最后一个差别比较复杂，因为如果 JScript .NET 编译器可以根据变量的使用情况推断出类型，它就会进行优化。在下面的示例中，变量 A 是早期绑定的，但变量 B 是晚期绑定的。

以下是引用片段：

```
var A;
var B;
A = "Hello";
B = "World";
B = 0;
```

为了获得最佳的性能，当声明 JScript .NET 变量时，请为其分配一个类型。例如，var A : String。

13. 使请求管线内的所有模块尽可能高效

请求管线内的所有模块在每次请求中都有机会被运行。因此，当请求进入和离开模块时快速地触发代码至关重要，特别是在不使用模块功能的代码路径里。分别在使用及不使用模块和配置文件时执行吞吐量测试，对确定这些方法的执行速度非常有用。

14. 使用 HttpServerUtility.Transfer 方法在同一应用程序的页面间重定向

采用 Server.Transfer 语法，在页面中使用该方法可避免不必要的客户端重定向。

15. 必要时调整应用程序每个辅助进程的线程数

ASP.NET 的请求结构试图在执行请求的线程数和可用资源之间达到一种平衡。已知一个使用足够 CPU 功率的应用程序，该结构将根据可用于请求的 CPU 功率，来决定允许同时执行的请求数。这项技术称作线程门控。但是在某些条件下，线程门控算法不是很有效。通过使用与 ASP.NET Applications 性能对象关联的 Pipeline Instance Count 性能计数器，可以在 PerfMon 中监视线程门控。

当页面调用外部资源，如数据库访问或 XML Web services 请求时，页面请求通常停止并释放 CPU。如果某个请求正在等待被处理，并且线程池中有一个线程是自由的，那么这个正在等待的请求将开始被处理。遗憾的是，有时这可能导致 Web 服务器上存在大量同时处理的请求和许多正在等待的线程，而它们对服务器性能有不利影响。通常，如果门控因子是外部资源的响应时间，则让过多请求等待资源，对 Web 服务器的吞吐量并无帮助。

为缓和这种情况，可以通过更改 Machine.config 配置文件节点的 maxWorkerThreads 和 maxIOThreads 属性，手动设置进程中的线程数限制。

注意 辅助线程是用来处理 ASP.NET 请求的，而 IO 线程则是用于为来自文件、数据库或 XML Web services 的数据提供服务的。

分配给这些属性的值是进程中每个 CPU 每类线程的最大数目。对于双处理器计算机，最大数是设置值的两倍。对于四处理器计算机，最大值是设置值的四倍。无论如何，对于有四个或八个 CPU 的计算机，最好更改默认值。对于有一个或两个处理器的计算机，默认值就可以，但对于有更多处理器的计算机的性能，进程中有一百或两百个线程则弊大于利。

注意 进程中有太多线程往往会降低服务器的速度，因为额外的上下文交换导致操作系统将 CPU 周期花在维护线程而不是处理请求上。

16. 适当地使用公共语言运行库的垃圾回收器和自动内存管理

小心不要给每个请求分配过多内存，因为这样垃圾回收器将必须更频繁地进行更多的工作。另外，不要让不必要的指针指向对象，因为它们将使对象保持活动状态，并且应尽量避免含 Finalize 方法的对象，因为它们在后面会导致更多的工作。特别是在 Finalize 调用中永远不要释放资源，因为资源在被垃圾回收器回收之前可能一直消耗着内存。最后这个问题经常会对 Web 服务器环境的性能造成毁灭性的打击，因为在等待 Finalize 运行时，很容易耗尽某个特定的资源。

17. 如果有大型 Web 应用程序，可考虑执行预批编译

每当发生对目录的第一次请求时都会执行批编译。如果目录中的页面没有被分析并编译，此功能会成批分析并编译目录

中的所有页面，以便更好地利用磁盘和内存。如果这需要很长时间，则将快速分析并编译单个页面，以便请求能被处理。此功能带给 ASP.NET 性能上的好处，因为它将许多页面编译为单个程序集。从已加载的程序集访问一页比每页加载新的程序集要快。

批编译的缺点在于：如果服务器接收到许多对尚未编译的页面的请求，那么当 Web 服务器分析并编译它们时，性能可能较差。为解决这个问题，可以执行预批编译。为此，只需在应用程序激活之前向它请求一个页面，无论哪页均可。然后，当用户首次访问您的站点时，页面及其程序集将已被编译。

没有简单的机制可以知道批编译何时发生。需一直等到 CPU 空闲或者没有更多的编译器进程(例如 csc.exe(C# 编译器)或 vbc.exe(Visual Basic 编译器))启动。

还应尽量避免更改应用程序的 \bin 目录中的程序集。更改页面会导致重新分析和编译该页，而替换 \bin 目录中的程序集则会导致完全重新批编译该目录。

在包含许多页面的大规模站点上，更好的办法可能是根据计划替换页面或程序集的频繁程度来设计不同的目录结构。不常更改的页面可以存储在同一目录中并在特定的时间进行预批编译。经常更改的页面应在它们自己的目录中(每个目录最多几百页)以便快速编译。

Web 应用程序可以包含许多子目录。批编译发生在目录级，而不是应用程序级。

18. 不要依赖代码中的异常

因为异常大大地降低性能，所以您不应该将它们用作控制正常程序流程的方式。如果有可能检测到代码中可能导致异常的状态，请执行这种操作。不要在处理该状态之前捕获异常本身。常见的方案包括：检查 null，分配给将分析为数字值的 String 一个值，或在应用数学运算前检查特定值。下面的示例演示可能导致异常的代码以及测试是否存在某种状态的代码。两者产生相同的结果。

以下是引用片段：

```
try
{
result = 100 / num;
}
catch (Exception e)
{
result = 0;
}
// ...to this.
if (num != 0)
result = 100 / num;
else
result = 0;
```

19. 使用 HttpResponse.Write 方法进行字符串串联

该方法提供非常有效的缓冲和连接服务。但是，如果您正在执行广泛的连接，请使用多个 Response.Write 调用。下面示例中显示的技术比用对 Response.Write 方法的单个调用连接字符串更快。

以下是引用片段：

```
Response.Write("a");
Response.Write(myString);
Response.Write("b");
Response.Write(myObj.ToString());
Response.Write("c");
Response.Write(myString2);
Response.Write("d");
```

20. 除非有特殊的原因要关闭缓冲，否则使其保持打开

禁用 Web 窗体页的缓冲会导致大量的性能开销。

21. 只在必要时保存服务器控件视图状态

自动视图状态管理是服务器控件的功能，该功能使服务器控件可以在往返过程上重新填充它们的属性值(您不需要编写任何代码)。但是，因为服务器控件的视图状态在隐藏的窗体字段中往返于服务器，所以该功能确实会对性能产生影响。您应该知道在哪些情况下视图状态会有所帮助，在哪些情况下它影响页的性能。例如，如果您将服务器控件绑定到每个往返过程上的数据，则将用从数据绑定操作获得的新值替换保存的视图状态。在这种情况下，禁用视图状态可以节省处理时间。

默认情况下，为所有服务器控件启用视图状态。若要禁用视图状态，请将控件的EnableViewState 属性设置为 false，如下面的 DataGrid 服务器控件示例所示。

以下是引用片段：

您还可以使用 @ Page 指令禁用整个页的视图状态。当您不从页回发到服务器时，这将十分有用：

以下是引用片段：

注意 @ Control 指令中也支持 EnableViewState 属性，该指令允许您控制是否为用户控件启用视图状态。

若要分析页上服务器控件使用的视图状态的数量，请(通过将 trace="true" 属性包括在 @ Page 指令中)启用该页的跟踪并查看 Control Hierarchy 表的 Viewstate 列。有关跟踪和如何启用它的信息，请参见 ASP.NET 跟踪。

22. 避免到服务器的不必要的往返过程

虽然您很可能希望尽量多地使用 Web 窗体页框架的那些节省时间和代码的功能，但在某些情况下却不宜使用 ASP.NET 服务器控件和回发事件处理。

通常，只有在检索或存储数据时，您才需要启动到服务器的往返过程。多数数据操作可在这些往返过程间的客户端上进行。例如，从 HTML 窗体验证用户输入经常可在数据提交到服务器之前在客户端进行。通常，如果不需要将信息传递到服务

器以将其存储在数据库中，那么您不应该编写导致往返过程的代码。

如果您开发自定义服务器控件，请考虑让它们为支持 ECMAScript 的浏览器呈现客户端代码。通过以这种方式使用服务器控件，您可以显著地减少信息被不必要的发送到 Web 服务器的次数。

使用 Page.IsPostBack 避免对往返过程执行不必要的处理

如果您编写处理服务器控件回发处理的代码，有时可能需要在首次请求页时执行其他代码，而不是当用户发送包含在该页中的 HTML 窗体时执行的代码。根据该页是否是响应服务器控件事件生成的，使用 Page.IsPostBack 属性有条件地执行代码。例如，下面的代码演示如何创建数据库连接和命令，该命令在首次请求该页时将数据绑定到 DataGrid 服务器控件。

以下是引用片段：

```
void Page_Load(Object sender, EventArgs e)
{
// Set up a connection and command here.
if (!Page.IsPostBack)
{
String query = "select * from Authors where FirstName like '%JUSTIN%'";
myCommand.Fill(ds, "Authors");
myDataGrid.DataBind();
}
}
```

由于每次请求时都执行 Page_Load 事件，上述代码检查 IsPostBack 属性是否设置为 false。如果是，则执行代码。如果该属性设置为 true，则不执行代码。

注意 如果不运行这种检查，回发页的行为将不更改。Page_Load 事件的代码在执行服务器控件事件之前执行，但只有服务器控件事件的结果才可能在输出页上呈现。如果不运行该检查，仍将为 Page_Load 事件和该页上的任何服务器控件事件执行处理。

23. 当不使用会话状态时禁用它

并不是所有的应用程序或页都需要针对于具体用户的会话状态，您应该对任何不需要会话状态的应用程序或页禁用会话状态。

若要禁用页的会话状态，请将 @ Page 指令中的 EnableSessionState 属性设置为 false。例如:

以下是引用片段：

注意 如果页需要访问会话变量，但不打算创建或修改它们，则将 @ Page 指令中的 EnableSessionState 属性设置为 ReadOnly。

还可以禁用 XML Web services 方法的会话状态。有关更多信息，请参见使用 ASP.NET 和 XML Web services 客户端创建的 XML Web services。

若要禁用应用程序的会话状态，请在应用程序 Web.config 文件的 sessionstate 配置节中将 mode 属性设置为 off。例如:

以下是引用片段：

24. 仔细选择会话状态提供程序

ASP.NET 为存储应用程序的会话数据提供了三种不同的方法：进程内会话状态、作为 Windows 服务的进程外会话状态和 SQL Server 数据库中的进程外会话状态。每种方法都有自己的优点，但进程内会话状态是迄今为止速度最快的解决方案。如果只在会话状态中存储少量易失数据，则建议您使用进程内提供程序。进程外解决方案主要用于跨多个处理器或多个计算机缩放应用程序，或者用于服务器或进程重新启动时不能丢失数据的情况。有关更多信息，请参见 ASP.NET 状态管理。

25. 不使用不必要的Server Control

ASP.net中，大量的服务器端控件方便了程序开发，但也可能带来性能的损失，因为用户每操作一次服务器端控件，就产生一次与服务器端的往返过程。因此，非必要，应当少使用Server Control。

26. ASP.NET应用程序性能测试

在对ASP.NET应用程序进行性能测试之前，应确保应用程序没有错误，而且功能正确。具体的性能测试可以采用以下工具进行：

Web Application Strees Tool (WAS)是Microsoft发布的一个免费测试工具，可以从 http://webtool.rte.microsoft.com/上下载。它可以模拟成百上千个用户同时对web应用程序进行访问请求，在服务器上形成流量负载，从而达到测试的目的，可以生成平均TTFB、平均TTLB等性能汇总报告。

Application Center Test (ACT) 是一个测试工具，附带于Visual Studio.NET的企业版中，是Microsoft正式支持的web应用程序测试工具。它能够直观地生成图表结果，功能比WAS多，但不具备多个客户机同时测试的能力。

服务器操作系统"管理工具"中的"性能"计数器，可以对服务器进行监测以了解应用程序性能。

结论：对于网站开发人员来说，在编写ASP.NET应用程序时注意性能问题，养成良好的习惯，提高应用程序性能，至少可以推迟必需的硬件升级，降低网站的成本。

- 尊重网上道德，遵守《全国人大常委会关于维护互联网安全的决定》及中华人民共和国其他各项有关法律法规
- 尊重网上道德，遵守中华人民共和国的各项有关法律法规
- 承担一切因您的行为而直接或间接导致的民事或刑事法律责任
- 中国网游研发中心新闻留言板管理人员有权保留或删除其管辖留言中的任意内容
- 您在中国网游研发中心留言板发表的作品，中国网游研发中心有权在网站内转载或引用
- 参与本留言即表明您已经阅读并接受上述条款

昵　　称：

GO

联系EMAIL：