【asp.net】vs2005环境下webconfig文件节点介绍

一、认识Web.config文件

Web.config 文件是一个xml文本文件，它用来储存 asp.NET Web 应用程序的配置信息（如最常用的设置asp.NET Web 应用程序的身份验证方式），它可以出现在应用程序的每一个目录中。当你通过.NET新建一个Web应用程序后，默认情况下会在根目录自动创建一个默认的Web.config文件，包括默认的配置设置，所有的子目录都继承它的配置设置。如果你想修改子目录的配置设置，你可以在该子目录下新建一个Web.config文件。它可以提供除从父目录继承的配置信息以外的配置信息，也可以重写或修改父目录中定义的设置。

(一).Web.Config是以xml文件规范存储,配置文件分为以下格式

1.配置节处理程序声明

特点：位于配置文件的顶部，包含在<configSections>标志中。

2.特定应用程序配置

特点: 位于<appSetting>中。可以定义应用程序的全局常量设置等信息.

3.配置节设置

特点: 位于<system.Web>节中，控制asp.net运行时的行为.

4.配置节组

特点: 用<sectionGroup>标记，可以自定义分组，可以放到<configSections>内部或其它<sectionGroup>标记的内部.

(二).配置节的每一节

1.<configuration>节根元素，其它节都是在它的内部.

2.<appSetting>节此节用于定义应用程序设置项。对一些不确定设置，还可以让用户根据自己实际情况自己设置

用法:

I.<appSettings>

<add key="Conntction" value="server=192.168.85.66;userid=sa;password=;database=Info;"/>

<appSettings>

定义了一个连接字符串常量，并且在实际应用时可以修改连接字符串，不用修改程式代码.

我的工具下载地址
电子书网站
网站管理
ip地址所在区域查询
qq空间
我的和讯博客
站长网，学习网站运营知识
存档
2009年05月(1)
2009年04月(5)
2009年02月(2)
2009年01月(6)
2008年12月(37)
2008年11月(35)
2008年10月(12)
2008年09月(13)
2008年08月(2)
2008年07月(2)
2008年06月(6)
2008年04月(1)
2007年12月(7)
2007年11月(15)
2007年10月(11)
2007年09月(1)

II.<appSettings>
<add key="ErrPage" value="Error.aspx"/><appSettings> 定义了一个错误重定向页面.
3.<compilation>节
格式:

```
<compilation
defaultLanguage="c#"
debug="true"
/>
```

I.default language: 定义后台代码语言,可以选择c#和vb.net两种语言.
IIdebug : 为true时，启动aspx调试；为false不启动aspx调试，因而可以提高应用程序运行时的性能。一般程序员在开发时设置为true,交给客户时设置为false.
4.<customErrors>节
格式:

```
<customErrors
mode="RemoteOnly"
defaultRedirect="error.aspx"
<error statusCode="440" redirect="err440page.aspx"/>
<error statusCode="500" redirect="err500Page.aspx"/>
/>
```

I.mode : 具有On,Off,RemoteOnly 3种状态。On表示始终显示自定义的信息; Off表示始终显示详细的asp.net错误信息; RemoteOnly表示只对不在本地Web服务器上运行的用户显示自定义信息.
II.defaultRedirect: 用于出现错误时重定向的URL地址. 是可选的
III.statusCode: 指明错误状态码，表明一种特定的出错状态.
IV. redirect:错误重定向的URL.
5.<globalization>节
格式:

```
<globalization
requestEncoding="utf-8"
responseEncoding="utf-8"
fileEncoding="utf-8"
/>
```

I.requestEncoding: 它用来检查每一个发来请求的编码.

II.responseEncoding: 用于检查发回的响应内容编码.
III.fileEncoding: 用于检查aspx,asax等文件解析的默认编码.
6.<sessionState>节
格式:

```
<sessionState
mode="InProc"
stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes"
cookieless="false"
timeout="20"
/>
```

I.mode: 分为off,Inproc,StateServer,SqlServer几种状态
mode = InProc 存储在进程中特点：具有最佳的性能，速度最快,但不能跨多台服务器存储共享.mode = "StateServer" 存储在状态服务器中特点: 当需要跨服务器维护用户会话信息时，使用此方法。但是信息存储在状态服务器上，一旦状态服务器出现故障，信息将丢失. mode="SqlServer" 存储在sql server中特点:工作负载会变大，但信息不会丢失.
II. stateConnectionString :指定asp.net应用程序存储远程会话状态的服务器名，默认为本机
III.sqlConnectionString:当用会话状态数据库时，在这里设置连接字符串
IV. Cookieless:设置为true时，表示不使用cookie会话状态来标识客户；否则，相反.
V. TimeOut:用来定义会话状态存储的时间，超过期限，将自动终止会话.
7.<authentication>节
格式:

```
<authentication mode="Forms">
<forms name=".ASPXUSERDEMO" loginUrl="Login.aspx" protection="All" timeout="30"/>
</authentication>
<authorization>
<deny users="?"/>
</authorization>
```

I.Windows: 使用IIS验证方式
II.Forms: 使用基于窗体的验证方式
III.Passport: 采用Passport cookie验证模式
IV.None: 不采用任何验证方式

里面内嵌Forms节点的属性涵义:
I.Name: 指定完成身份验证的Http cookie的名称.
II.LoginUrl: 如果未通过验证或超时后重定向的页面URL，一般为登录页面，让用户重新登录
III.Protection: 指定 cookie数据的保护方式.
可设置为: All None Encryption Validation四种保护方式
a. All表示加密数据，并进行有效性验证两种方式
b. None表示不保护Cookie.
c. Encryption表示对Cookie内容进行加密
d. validation表示对Cookie内容进行有效性验证
IV. TimeOut: 指定Cookie的失效时间. 超时后要重新登录.
在运行时对Web.config文件的修改不需要重启服务就可以生效（注：<processModel> 节例外）。当然Web.config文件是可以扩展的。你可以自定义新配置参数并编写配置节处理程序以对它们进行处理。
web.config配置文件（默认的配置设置）以下所有的代码都应该位于
<configuration>
<system.web>
和
</system.web>
</configuration>
之间，出于学习的目的下面的示例都省略了这段xml标记。
1、<authentication> 节
作用：配置 asp.NET 身份验证支持（为Windows、Forms、PassPort、None四种）。该元素只能在计算机、站点或应用程序级别声明。< authentication> 元素必需与<authorization> 节配合使用。
示例:
以下示例为基于窗体（Forms）的身份验证配置站点，当没有登陆的用户访问需要身份验证的网页，网页自动跳转到登陆网页。
<authentication mode="Forms" >
<forms loginUrl="logon.aspx" name=".FormsAuthCookie"/>
</authentication>
其中元素loginUrl表示登陆网页的名称，name表示Cookie名称。
2、<authorization> 节
作用：控制对 URL 资源的客户端访问（如允许匿名用户访问）。此元素可以在任何级别（计算机、站点、应用程序、子目录或页）上声明。必需与<authentication> 节配合使用。

示例：以下示例禁止匿名用户的访问

```
<authorization>
  <deny users="?"/>
</authorization>
```

注：你可以使用user.identity.name来获取已经过验证的当前的用户名；可以使用web.Security.FormsAuthentication.RedirectFromLoginPage方法将已验证的用户重定向到用户刚才请求的页面.具体的

3、<compilation>节

作用：配置 asp.NET 使用的所有编译设置。默认的debug属性为“True”.在程序编译完成交付使用之后应将其设为False（Web.config文件中有详细说明，此处省略示例）

4、<customErrors>

作用：为 asp.NET 应用程序提供有关自定义错误信息的信息。它不适用于 xml Web services 中发生的错误。

示例：当发生错误时，将网页跳转到自定义的错误页面。

```
<customErrors defaultRedirect="ErrorPage.aspx" mode="RemoteOnly">
</customErrors>
```

其中元素defaultRedirect表示自定义的错误网页的名称。mode元素表示：对不在本地 Web 服务器上运行的用户显示自定义(友好的)信息。

5、<httpRuntime>节

作用：配置 asp.NET HTTP 运行库设置。该节可以在计算机、站点、应用程序和子目录级别声明。

示例：控制用户上传文件最大为4M，最长时间为60秒，最多请求数为100

```
<httpRuntime maxRequestLength="4096" executionTimeout="60" appRequestQueueLimit="100"/>
```

6、 <pages>

作用：标识特定于页的配置设置（如是否启用会话状态、视图状态，是否检测用户的输入等）。<pages>可以在计算机、站点、应用程序和子目录级别声明。

示例：不检测用户在浏览器输入的内容中是否存在潜在的危险数据（注：该项默认是检测，如果你使用了不检测，一要对用户的输入进行编码或验证)，在从客户端回发页时将检查加密的视图状态，以验证视图状态是否已在客户端被篡改。(注：该项默认是不验证）

```
<pages buffer="true" enableViewStateMac="true" validateRequest="false"/>
```

7、<sessionState>

作用：为当前应用程序配置会话状态设置（如设置是否启用会话状态，会话状态保存位置）。

示例：

```
<sessionState mode="InProc" cookieless="true" timeout="20"/>
```

</sessionState>
注：
mode="InProc"表示：在本地储存会话状态（你也可以选择储存在远程服务器或SAL服务器中或不启用会话状态）
cookieless="true"表示：如果用户浏览器不支持Cookie时启用会话状态(默认为False）
timeout="20"表示：会话可以处于空闲状态的分钟数
8、<trace>
作用：配置 asp.NET 跟踪服务，主要用来程序测试判断哪里出错。
示例：以下为Web.config中的默认配置：
<trace enabled="false" requestLimit="10" pageOutput="false" traceMode="SortByTime" localOnly="true" />
注：
enabled="false"表示不启用跟踪;
requestLimit="10"表示指定在服务器上存储的跟踪请求的数目
pageOutput="false"表示只能通过跟踪实用工具访问跟踪输出;
traceMode="SortByTime"表示以处理跟踪的顺序来显示跟踪信息
localOnly="true" 表示跟踪查看器 (trace.axd) 只用于宿主 Web 服务器
自定义Web.config文件配置
自定义Web.config文件配置节过程分为两步。
1.在配置文件顶部 <configSections> 和 </configSections>标记之间声明配置节的名称和处理该节中配置数据的 .NET Framework 类的名称。
2.是在 <configSections> 区域之后为声明的节做实际的配置设置。
示例：创建一个节存储数据库连接字符串
<configuration>
<configSections>
<section name="appSettings" type="System.Configuration.NameValueFileSectionHandler, System, Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"/>
</configSections>
<appSettings>
<add key="scon" value="server=a;database=northwind;uid=sa;pwd=123"/>
</appSettings>
<system.web>
.....

</system.web>
</configuration>
访问Web.config文件你可以通过使用ConfigurationSettings.AppSettings 静态字符串集合来访问 Web.config 文件示例：获取上面例子中建立的连接字符串。例如：
protected static string Isdebug = ConfigurationSettings.AppSettings["debug"]
二、web.config中的session配置详解
打开某个应用程序的配置文件Web.config后，我们会发现以下这段：
< sessionState
mode="InProc"
stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes"
cookieless="false"
timeout="20"
/>
这一段就是配置应用程序是如何存储session信息的了。我们以下的各种操作主要是针对这一段配置展开。让我们先看看这一段配置中所包含的内容的意思。sessionState节点的语法是这样的：
< sessionState mode="Off|InProc|StateServer|SQLServer"
cookieless="true|false"
timeout="number of minutes"
stateConnectionString="tcpip=server:port"
sqlConnectionString="sql connection string"
stateNetworkTimeout="number of seconds"
/>
必须有的属性是：属性选项描述
mode 设置将session信息存储到哪里
Ø Off 设置为不使用session功能，
Ø InProc 设置为将session存储在进程内，就是asp中的存储方式，这是默认值，
Ø StateServer 设置为将session存储在独立的状态服务中，
Ø SQLServer 设置将session存储在sql server中。
可选的属性是：属性选项描述
Ø cookieless 设置客户端的session信息存储到哪里，
Ø ture 使用Cookieless模式，

Ø false 使用Cookie模式，这是默认值，
Ø timeout 设置经过多少分钟后服务器自动放弃session信息，默认为20分钟。
stateConnectionString 设置将session信息存储在状态服务中时使用的服务器名称和端口号，例如："tcpip=127.0.0.1:42424”。当mode的值是StateServer是，这个属性是必需的。
sqlConnectionString 设置与sql server连接时的连接字符串。例如"data source= localhost;Integrated Security=SSPI;Initial Catalog=northwind"。当mode的值是 SQLServer时，这个属性是必需的。
stateNetworkTimeout 设置当使用StateServer模式存储session状态时，经过多少秒空闲后，断开Web服务器与存储状态信息的服务器的tcp/IP连接的。默认值是10秒钟。
asp.NET中客户端session状态的存储
在我们上面的session模型简介中，大家可以发现session状态应该存储在两个地方，分别是客户端和服务器端。客户端只负责保存相应网站的SessionID，而其他的session信息则保存在服务器端。在asp中，客户端的SessionID实际是以Cookie的形式存储的。如果用户在浏览器的设置中选择了禁用Cookie，那末他也就无法享受session的便利之处了，甚至造成不能访问某些网站。为了解决以上问题，在 asp.NET中客户端的session信息存储方式分为：Cookie和Cookieless两种。
asp.NET中，默认状态下，在客户端还是使用Cookie存储session信息的。如果我们想在客户端使用Cookieless的方式存储session信息的方法如下：
找到当前Web应用程序的根目录，打开Web.Config文件，找到如下段落：
< sessionState
mode="InProc"
stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes"
cookieless="false"
timeout="20"
/>
这段话中的cookieless="false"改为：cookieless="true"，这样，客户端的session信息就不再使用 Cookie存储了，而是将其通过URL存储。关闭当前的IE，打开一个新IE，重新访问刚才的Web应用程序，就会看到类似下面的样子：
其中，http://localhost/MyTestApplication/(ulqsek45heu3ic2a5zgdl245) /default.aspx中黑体标出的就是客户端的session ID。注意，这段信息是由IIS自动加上的，不会影响以前正常的连接。
asp.NET中服务器端session状态的存储准备工作：
为了您能更好的体验到实验现象，您可以建立一个叫做SessionState.aspx的页面，然后把以下这些代码添加到< body>< /body>中。

```
< scriptrunat="server">
Sub Session_Add(sender As Object, e As EventArgs)
session("MySession") = text1.Value
span1.InnerHtml = "Session data updated! < P>Your session contains: < font color=red>" & session("
MySession"). ToString() & "< /font>"
End Sub
Sub CheckSession(sender As Object, eAs EventArgs)
If (Session("MySession")Is Nothing) Then
span1.InnerHtml = "NOTHING, session DATA LOST!"
Else
span1.InnerHtml = "Your session contains: < font color= red>" & session("MySession").ToString() & "
<   /font>"
End If
End Sub
< /script>
< formrunat="server"id="Form2">
< inputid="text1"type="text"runat="server"name="text1">
< inputtype="submit"runat="server"OnServerClick="Session_Add"
value="Add to session State " id="Submit1"name="Submit1">
< inputtype="submit"runat="server"OnServerClick="CheckSession"
value=" View session State " id="Submit2"name="Submit2">
< /form>
< hrsize="1">
< fontsize="6">< spanid="span1"runat="server" />< /font>
```

这个SessionState.aspx的页面可以用来测试在当前的服务器上是否丢失了session信息。

将服务器session信息存储在进程中

让我们来回到Web.config文件的刚才那段段落中:

```
< sessionState
mode="InProc"
stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes"
cookieless="false"
```

timeout="20"
/>
当mode的值是InProc时，说明服务器正在使用这种模式。
这种方式和以前asp中的模式一样，就是服务器将session信息存储在IIS进程中。当IIS关闭、重起后，这些信息都会丢失。但是这种模式也有自己最大好处，就是性能最高。应为所有的session信息都存储在了IIS的进程中，所以IIS能够很快的访问到这些信息，这种模式的性能比进程外存储session信息或是在sql server中存储session信息都要快上很多。这种模式也是asp.NET的默认方式。
好了，现在让我们做个试验。打开刚才的SessionState.aspx页面，随便输入一些字符，使其存储在session中。然后，让我们让IIS重起。注意，并不是使当前的站点停止再开始，而是在IIS中本机的机器名的节点上点击鼠标右键，选择重新启动IIS。(想当初使用NT4时，重新启动IIS必须要重新启动计算机才行，微软真是@#$%^&)返回到SessionState.aspx页面中，检查刚才的session信息，发现信息已经丢失了。
将服务器session信息存储在进程外
首先，让我们来打开管理工具->服务，找到名为：asp.NET State Service的服务，启动它。实际上，这个服务就是启动一个要保存session信息的进程。启动这个服务后，你可以从Windows任务管理器->进程中看到一个名为 aspnet_state.exe的进程，这个就是我们保存session信息的进程。
然后，回到Web.config文件中上述的段落中，将mode的值改为StateServer。保存文件后的重新打开一个IE，打开 SessionState.aspx页面，保存一些信息到session中。这时，让我们重起IIS，再回到SessionState.aspx页面中查看刚才的session信息，发现没有丢失。
实际上，这种将session信息存储在进程外的方式不光指可以将信息存储在本机的进程外，还可以将session信息存储在其他的服务器的进程中。这时，不光需要将mode的值改为StateServer，还需要在stateConnectionString中配置相应的参数。例如你的计算你是192.168.0.1，你想把session存储在ip为192.168.0.2的计算机的进程中，就需要设置成这样： stateConnectionString="tcpip=192.168.0.2:42424"。
当然，不要忘记在192.168.0.2的计算机中装上.NET Framework，并且启动asp.NET State Services服务。
将服务器session信息存储在sql server中
首先，还是让我们来做一些准备工作。启动sql server和sql server代理服务。在sql server中执行一个叫做 InstallSqlState.sql的脚本文件。这个脚本文件将在sql server中创建一个用来专门存储session信息的数据库，及一个维护session信息数据库的sql server代理作业。我们可以在以下路径中找到那个文件：
[system drive]\winnt\Microsoft.NET\Framework\[version]\

然后打开查询分析器，连接到sql server服务器，打开刚才的那个文件并且执行。稍等片刻，数据库及作业就建立好了。这时，你可以打开企业管理器，看到新增了一个叫ASPState的数据库。但是这个数据库中只是些存储过程，没有用户表。实际上session信息是存储在了tempdb 数据库的ASPStateTempSessions表中的，另外一个ASPStateTempApplications表存储了asp中 application对象信息。这两个表也是刚才的那个脚本建立的。另外查看管理->SQL server代理->作业，发现也多了一个叫做ASPState_Job_DeleteExpiredSessions的作业，这个作业实际上就是每分钟去ASPStateTempSessions 表中删除过期的session信息的。

接着，我们返回到Web.config文件，修改mode的值改为SQLServer。注意，还要同时修改sqlConnectionString的值，格式为：

sqlConnectionString="data source=localhost; Integrated Security=SSPI;"

其中data source是指sql server服务器的ip地址，如果sql server与IIS是一台机子，写127.0.0.1 就行了。Integrated Security=SSPI的意思是使用Windows集成身份验证，这样，访问数据库将以asp.NET的身份进行，通过如此配置，能够获得比使用userid=sa;password=口令的sql server验证方式更好的安全性。当然，如果sql server运行于另一台计算机上，你可能会需要通过Active Directory域的方式来维护两边验证的一致性。

同样，让我们做个试验。向SessionState.aspx中添加session信息，这时发现session信息已经存在 sql server中了，即使你重起计算机，刚才的session信息也不会丢失。现在，你已经完全看见了session信息到底是什么样子的了，而且又是存储在sql server中的，能干什么就看你的发挥了。

总结

三、asp.net 关于form认证的一般设置

asp.net 关于form认证的一般设置：

1: 在web.config中，加入form认证；

```
  <authentication mode="Forms">
        <forms name="auth" loginUrl="index.aspx" timeout="30"></forms>
</authentication>
<authorization>
      <deny users="?" />
 </authorization>
```

2: 如果有注册页面时还应该允许匿名用户调用注册页面进行注册;

以下代码应该在<configuration><system.web>之间,而不应该包含到<system.web>..</system.web>之间;

----------表示允许匿名用户对 userReg.aspx页面进行访问.

```
 <location path="userReg.aspx">
 <system.web>
    <authorization>
       <allow users="?" />
    </authorization>
 </system.web>
</location>
```

3 在登录成功后要创建身份验证票, 表明已经通过认证的合法用户;

```
if(登陆成功)
System.Web.Security.FormsAuthentication.SetAuthCookie(用户名称, false);
```

四、访问Web.config文件

你可以通过使用ConfigurationSettings.AppSettings 静态字符串集合来访问 Web.config 文件示例：获取上面例子中建立的连接字符串。例如：

```
protected static string Isdebug = ConfigurationSettings.AppSettings["scon"]
```

asp.Net性能优化.

(一).选择会话状态存储方式

在Webconfig文件配置:

```
<sessionState mode="???" stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes"
 cookieless="false" timeout="20"/>
```

asp.net有三种方式存储会话状态信息:

1. 存储在进程中: 属性mode = InProc

   特点： 具有最佳的性能，速度最快,但不能跨多台服务器存储共享.

2. 存储在状态服务器中: 属性mode = "StateServer"

   特点: 当需要跨服务器维护用户会话信息时，使用此方法。

   但是信息存储在状态服务器上，一旦状态服务器出现故障，信息将丢失

3. 存储在sql server中: 属性mode="SqlServer"

   特点: 工作负载会变大，但信息不会丢失.

补充一点:

I. 由于某些页面不需要会话状态，则可以将会话状态禁用:

代码如下： <%@ Page EnableSessionState="false" %>

II.如果页面需要访问会话变量但不允许修改它们，可以设置页面会话状态为只读:

代码如下： <%@ Page EnableSessionState="false" %>

使用时可以根据具体情况选择某种方式

(二).使用Page.IsPostBack

Page.IsPostBack表示是否是从客户端返回的. 初次运行时，不是从客户端返回，它的值为false,当触发页面上的事件或刷新页面时，Page.IsPostBack由于是回发的，值变为true;

一般在: Page_Load方法中用:

```
private void Page_Load(Object sender,EventArgs e)
{
   if(!Page.IsPostBack)
   {
      ....; //初始化页面的代码。这些代码第一次页面初始化时执行，当第二次回发时，
          //不会再执行。提高效率。
   }
}
```

往往很多时候不得不用IsPostBack, 因为有些控件初始化后，要保持它的状态.

例如: DropDownList,如果每次都初始化，则用户无论选择其选项，都会被初始化为默认值.

(三).避免使用服务器控件

1.一般的静态显示信息，尽量不要用服务端控件显示. 因为服务端控件需要回发服务端执行,会降低程序执行效率,一般用<DIV>显示即可.

如果用了服务端控件，将: runat="server"去掉,也会提高效率.

2.禁用服务端控件的状态视图，有些控件不需要维护其状态，可以设置其属性: EnableViewState=false;

如果整个页面控件都不需要维持状态视图，则可以设置整个页面的状态视力为false:

代码如下: <%@ Page EnableViewState="false"%>

3.在Web.Config文件中配置:

asp.NET Sessionss可以在Web.config或Machine.config中的Sessionsstate元素中配置。

下面是在 Web.config中的设置的例子:

```
<Sessionsstate timeout="10" cookieless="false" mode="Inproc" />
```

(四).避免使用DataGrid

大家都知道DataGrid功能强大。但是功能强大的同时，增加了性能上的开销。一般用其它控件: DataList

或Repeater控件能实现的，尽量不用DataGrid.

(五).字符串操作

1.避免装箱操作. 装箱操作运行效率比较低.

例如运行两个代码段:

```
string test="";
for(for int i=0;i<10000;i++)
{
   test = test + i;
}
```

和

```
      string test="";
for(for int i=0;i<10000;i++)
{
   test = test + i.ToString();
}
```

下面的代码段显然效率要高.因为i是整型的，系统要先把i进行装箱转换为string型的，再进行连接.需要时间

读者可以Copy到自己机器上测试一下.

2.使用StringBulider类

在进行字符串连接时: string str = str1 + str2 + ...;

一般超过三项连接，最好用StringBuilder来代替string类. StringBuilder可以避免重新创建string 对象造成

的性能损失.

一般用于组装sql语句时用到: StringBulider.

读者可以到自己机器上测试一下.

3.尽量少用:

```
try
{}
catch
```

{}
finally
{}
语句.此语句执行效率比较低.

(六).ADO.Net使用方面优化

1.数据库连接打开和关闭。 在需要连接时打开，当访问完数据库要立刻关闭连接.

举例说明,还是看两个代码段:

I.

```
DataSet ds = new DataSet();
SqlConnection MyConnection = new SqlConnection("server=localhost; uid=sa; pwd=; database
=NorthWind");
SqlCommand myCommand = new SqlCommand(strSql,MyConnection);
SqlDataAdapter myAdapter=new SqlDataAdapter(queryStr,connectionStr);
MyConnection.Open();     //打开连接
for(int i=0;i<1000;i++)   //for循环模拟取得数据前的商业逻辑操作
{
  Thread.Sleep(1000);
}
myAdapter.Fill(ds);
for(int i=0;i<1000;i++)   //for循环模拟取得数据后的商业逻辑操作
{
  Thread.Sleep(1000);
}
MyConnection.Close();    //关闭连接
```

II.

```
DataSet ds = new DataSet();
SqlConnection MyConnection = new SqlConnection("server=localhost; uid=sa; pwd=; database
=NorthWind");
SqlCommand myCommand = new SqlCommand(strSql,MyConnection);
SqlDataAdapter myAdapter=new SqlDataAdapter(queryStr,connectionStr);
for(int i=0;i<1000;i++)   //for循环模拟取得数据前的商业逻辑操作
{
  Thread.Sleep(1000);
```

```
}
MyConnection.Open();     //打开连接
   myAdapter.Fill(ds);
  MyConnection.Close();    //关闭连接
for(int i=0;i<1000;i++)   ////for循环模拟取得数据后的商业逻辑操作
{
   Thread.Sleep(1000);
}
```

显示II代码比I代码好的多，I中早早占着连接不放，如果用户很多的话，容易出现连接池满情况。严重时出现死机现象.

2.数据库查询

I. 直接生成sql语句。 sql server每次都要对其进行编译，在性能方面不会有很大的提高。另外也不够安全。容易被攻击.

II. 使用带参数的sql命令。这种方式sql server只对其编译一次，对于不同的参数可以重复使用编译后的命令。提高了性能.

III.使用sql server存储过程. 编译一次. 具有独立性，便于修改和维护. 一次能完成用语句发送多次的功能.减少了网络的

流量。 并不一定存储过程一定比语句效率要高，如果商业逻辑很复杂的话，有时候用语句比存储过程效率要高.

(六).缓存优化

缓存分为两种：页面缓存和API缓存.

1.使用页面缓存和片段缓存

```
<%@ OutputCache Duration="5" VaryByParam="None"%>
<%@ OutputCache Duration=60 VaryByParam="TextBox1,TextBox2" %>
```

说明: Duration是设置Cache的过期时间;

VarByParam是设置是否根据参数而变化,None时所有参数使用同一Cache,

设置TextBox1时则根据TextBox1的不同值分别缓存;当有多个参数时则要组合缓存;

2.API缓存。用于在应用程序中使用

I. 一个Cache使用的例子:

http://blog.csdn.net/chengking/archive/2005/10/03/494545.aspx

II.使用时注意Page.Cache和HttpContext.Current.Cache区别:

它们指的同一个对象，在Page里，用Page.Cache，如果在global.asax或自己的类里用:HttpCont

ext.Current.Cache 在有些事件中，由于其没有HttpContext，就用HttpRuntime.Cache.

发表于 @ 2008年12月02日 22:01:00 | 评论(0) | 举报 | 收藏

旧一篇:【ASP.NET】 程序中常用的三十三种代码 | 新一篇:【技术资料】photoshop 167种技巧

发表评论

表 情:

评论内容:

用 户 名: 匿名用户 登录 注册

验 证 码: FKTD 重新获得验证码

发表评论