提高 ASP.NET 性能与可伸缩性的几个常用方法剖析

在 ASP.NET 中，有很多提高性能和可伸缩性的方法，本篇就为朋友们介绍 7 个，朋友们可以适当的应用在项目之中。

本篇的议题如下：

ASP.NET 管道优化

ASP.NET 处理配置的优化

ASP.NET 管道优化

我们知道，在 ASP.NET 的处理机制的设计是基于管道模型的，ASP.NET 的管道中，有很多的 HttpModule。每个要处理的请求经过 ASP.NET 管道的时候，都会被其中的 HttpModule 拦截，进行相关的处理之后，再将请求发送给下一个 HttpModule。例如，SessionStateModule 会拦截请求，并且解析请求中的 Session Cookie，然后加载合适的 Session 到 HttpContext 中（要知道，每一个 Session 都会有一个保存在 Cookie 中的 Session Key）。

同时，并且不是在管道中的所有的 HttpModule 都是必须的，例如，如果我们没有在项目中使用 Membership 和 Profiler Provider，那么没我们就没有必要使用 FormsAuthenticate 模块，再如，如果我们没有使用 Windows 身份验证，那么，我们就可以移除 WindowsAuthenticate 模块。

在默认情况下，每次 ASP.NET 运行时在启动的时候，都会根据配置文件去加载相应的 HttpModule，根据 IIS 的版本不同，要查看的配置文件也不一样。

对于 IIS 6 而言，默认的定义了加载 HttpModule 的文件就是 machine.config（位于：*$WINDOWS$\Microsoft.NET\Framework\$VERSION$\CONFIG*）。

对于 IIS 7（以及以后版本），默认的配置文件是 applicationHost.config(*位于：%SystemRoot%\system32\inetsrv)*。

我们这里以 IIS 6 为例子，machine.config 配置如下：

www.agilesharp.com 专注打造优质高效技术博客 .NET架构设计

```
<httpModules>
  <add name="OutputCache" type="System.Web.Caching.OutputCacheModule" />
  <add name="Session" type="System.Web.SessionState.SessionStateModule" />
  <add name="WindowsAuthentication"
       type="System.Web.Security.WindowsAuthenticationModule" />
  <add name="FormsAuthentication"
       type="System.Web.Security.FormsAuthenticationModule" />
  <add name="PassportAuthentication"
       type="System.Web.Security.PassportAuthenticationModule" />
  <add name="UrlAuthorization" type="System.Web.Security.UrlAuthorizationModule" />
  <add name="FileAuthorization" type="System.Web.Security.FileAuthorizationModule" />
  <add name="ErrorHandlerModule" type="System.Web.Mobile.ErrorHandlerModule,
                        System.Web.Mobile, Version=1.0.5000.0,
                        Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
</httpModules>
```

我们可以在我们站点的 web.config 移除我们不需要的 Module，如下：

```
<httpModules>
  <!-- Remove unnecessary Http Modules for faster pipeline -->
  <remove name="Session" />
  <remove name="WindowsAuthentication" />
  <remove name="PassportAuthentication" />
  <remove name="AnonymousIdentification" />
  <remove name="UrlAuthorization" />
  <remove name="FileAuthorization" />
</httpModules>
```

ASP.NET 处理配置的优化

ASP.NET 处理模型的配置定义了一些 ASP.NET 在处理请求的时候的一些特性，例如开启多少个线程进行请求的处理，每一个请求处理线程的过期时间时候多少等。

很多的时候，需要我们自己根据实际情况去修改默认的配置，特别是随着现在硬件变得越来越便宜和功能越来越强大，对配置对相应的修改，可以极大的提升站点的性能。

以 IIS 6 而言，默认的处理模型的配置文件依然在 machine.config 中，如下：

```
<system.web>
  <processModel autoConfig="true" />
```

对于 IIS 7 而言，配置在 applicationHost.config，其中，节点的名称和 IIS 6 一样。

这里，我们依然以 **IIS 6** 为例子，如下：

```
<processModel
   enable="true"
   timeout="Infinite"
   idleTimeout="Infinite"
   shutdownTimeout="00:00:05"
   requestLimit="Infinite"
   requestQueueLimit="5000"
   restartQueueLimit="10"
   memoryLimit="60"
   webGarden="false"
   cpuMask="0xffffffff"
   userName="machine"
   password="AutoGenerate"
   logLevel="Errors"
   clientConnectedCheck="00:00:05"
   comAuthenticationLevel="Connect"
   comImpersonationLevel="Impersonate"
   responseDeadlockInterval="00:03:00"
   responseRestartDeadlockInterval="00:03:00"
   autoConfig="false"
   maxWorkerThreads="100"
   maxIoThreads="100"
   minWorkerThreads="40"
   minIoThreads="30"
   serverErrorMessageFile=""
   pingFrequency="Infinite"
   pingTimeout="Infinite"
   asyncOption="20"
   maxAppDomains="2000"
/>
```

首先，看到如此众多的配置项不要惊慌，要淡定，因为其中每一个项都是非常有用的。下面我们就挑几个比较常用的，也是重点的来看看。

maxWorkerThreads:这个可以说是见名知意了，就是每一个站点的线程池可以启动的最多的处理线程的数目。在 IIS 6 中，每一个逻辑可以开启的最大线程数目是 20 个，双核的 CPU 那就是 40 个了。

如果我们的服务器的配置不错，我们可以将这个数目变大一点，例如 100 等。这里需要注意的就是，我们要根据实际的情况要配置这个数目，但是我们有一些大致的指导方针：每个线程的开启需要大约 4M 的内存，如果我们的服务器的物理内存是 2G，那么在理论上，我们是可以同时跑 500 的处理线程的。

但是很多时候，我们的服务器的内存不可能全部用在这里。因为服务器上面的内存要被用在内核模式和用户模式。所谓的内核模式，就是 Windows 内部核心的操作，例如管理线程，进程，管理 I/O 设备的驱动等。用户模式就是除了内核模式以外的操作，例如位来自用户应用程序的请求提供服务，包括 IIS，SQL Server 等。所有用户模式的应用程序通过运行在内核模式的执行层访问资源，例如，如果应用程序要进行磁盘的 I/O，那么该请求就会提交到内核模式的执行层，由它来执行请求并且将结果返回给发出请求的用户模式的进程。

这里面的东西很多，我们就不扯远了。还是言归正传。

如果我们的一个 ASP.NET 的站点应用不是消耗 CPU 的操作的应用，那么，我们可以适当的增加更多的处理线程。例如，如果我们的应用程序只要是为外界提供 Web 服务，或者进行文件的上传和下载，而这些操作对 CPU 的压力不大，这个时候，我们可以配置更多的处理线程。

这里有一点需要注意的就是：尽管，我们配置了 100 个或者更多，也不见得这 100 线程就会理解使用，而且还有一个负面的作用就是：线程越多，CPU 调度就越复杂。这里我想起了之前有朋友问我一个问题：我们已经配置了最大的处理线程是 1000，为什么我们的站点的并发处理能力没有达到 1000 呢？我当是给这个一个最简单的回复：你手里有了 1000 万元，你会一下子全部花完吗？CPU 也是这样，通过配置 maxWorkerThreads，我们告诉服务器，最大的限度可以到某个数目，但是不见得服务器就一定按照这个最大的限度来，因为有很多的东西需要服务器去考虑和计算，例如之前的内存问题，等。

另外，如果 ASP.NET 线程池中没有可以用来处理请求的线程，那么之后发给站点的请求就会加入到等待队列中。

需要大家配置之后，多多的进行测试。

另外，为大家给出下面的一个处理线程数目的表格，进行参考：

.NET Framework 版本	默认的处理线程数目
1.1	20*逻辑CPU个数-8
2.0	12*逻辑CPU个数
3.5,4.0	IIS 7经典模式:12*逻辑CPU个数
	IIS 7集成模式：100*逻辑CPU个数

maxIOThreads：默认是每个逻辑处理有 20 个线程。I/O 处理线程用来处理对 I/O 的请求，例如对文件的读和写的操作，数据库的操作，Web Service 的调用。在设置这个配置的时候，需要考虑站点中文件的上传、下载的情况，考虑的要点可以参考上面的 maxWorkderThreads，另外需要注意的就是，文件的上传、下载是一个非常消耗内存与磁盘的操作，如果站点有很多的文件要上传写入和下载，那么建议重新用另外的服务器去处理，并且将相关的文件缓存在内存中，即，搭建文件缓存服务器。

memoryLimit:配置在 IIS 6 中 w3pw.exe **进程**可以使用的最大的内存容量，这是一个百分比值。如果站点的处理进程使用内存超过了这个容量限制，那么 IIS 就会重新的启动一个进程，并且将请求给这个新的进程，原先的旧的进程就被杀死了（注：这里说的是**进程**）。如果我们的服务器上只运行一个 ASP.NET 的站点，而且没有其他消耗内存的应用，那么，我们可以尝试将这个配置改为 80（即，服务器 RAM 的 80%）。

如果我们的服务器上面有可能导致内存泄露的应用在运行，例如在应用中有 COM 组件等，那么可以把这个配置值设置的小一点，这样就可以在站点发生了内存泄露问题之后，我们的处理进程可以快速的被杀死，从而回收一些内存资源。当然，这种设置最小值的的方法只是一个临时的解决方案，最终还需我们去彻底的解决内存泄露问题。

除了可以设置 processModel 之外，我们还可以设置其他的配置，例如:system.net 节点，我们可以设置一个 IP 地址的最大的请求连接数（默认是 2），如下：

```
<system.net>
  <connectionManagement>
    <add address="*" maxconnection="100" />
  </connectionManagement>
</system.net>
```

今天就暂时到这里，余下的内容，我们稍后继续！

另外，我们还提供很多的系列文章：

系列文章链接：

IIS 负载均衡-Application Request Route 详解第一篇： ARR 介绍

IIS 负载均衡-Application Request Route 详解第二篇：创建与配置 Server Farm

IIS 负载均衡-Application Request Route 详解第三篇：使用 ARR 进行 Http 请求的负载均衡（上）

IIS 负载均衡-Application Request Route 详解第三篇：使用 ARR 进行 Http 请求的负载均衡（下）

IIS 负载均衡-Application Request Route 详解第四篇：使用 ARR 实现三层部署架构

查询优化器内核剖析第一篇

查询优化器内核剖析第二篇：产生候选执行计划&执行计划成本估算

查询优化器内核剖析第三篇：查询的执行与计划的缓存 & Hint 提示

查询优化器内核剖析第四篇：从一个实例看执行计划

查询优化器内核剖析第五篇：进一步的了解执行计划

查询优化器内核剖析第七篇：执行引擎之数据访问操作---Scan

查询优化器内核剖析第八篇：执行引擎之数据访问操作---Seek(上)

查询优化器内核剖析第八篇：执行引擎之数据访问操作---Seek(下）

www.agilesharp 至于打造高质量的技术博客--提供 .NET架构设计