ASP.NET企业网站的性能优化研究*

岳付强
(西昌学院,四川 西昌 615013)

【摘 要】ASP.NET企业网站的性能对于企业来说是极为重要的,而网站的性能优化工作是多方面的又是长期的。本文首先介绍了网站的整体性能优化思想,然后从网站架构、页面优化、业务逻辑、数据访问、缓存技术、IIS配置等方面探讨了网站性能优化的方法和技巧。

【关键词】ASP.NET;企业网站;性能优化;缓存
【中图分类号】TP393.092 【文献标识码】A 【文章编号】1673-1891(2010)03-0063-05

1 引言

由于ASP.NET技术越来越成熟,越来越多的企业使用微软.NET平台和Visual Studio工具来架构基于ASP.NET的门户网站。作为企业对外的窗口,网站除了完善的功能和美观的页面外,更需要快速响应用户请求的能力。尤其是大型企业网站,每时每刻都会有成千上万的用户在访问,如果没有良好的性能,根本无法满足用户的需求。

在面对大量用户访问、高并发请求时,网站可以通过使用高性能的服务器、高性能的数据库、高效的编程语言以及加大带宽容量等手段来解决,但这意味着更大的资金投入,而且仍然不能从根本上解决网站面临的高负载和高并发等问题。如果企业目前没有大量的资金投入又想获得更好的网站性能,则需精打细算从"软"方面着手。本文就从日常开发的角度来探讨ASP.NET企业网站性能优化的方法和技巧。

2 整体性能优化思想

企业网站的性能优化,应充分发挥ASP.NET应用程序浏览器端和服务器端数据处理的特点,最大限度地减少客户端与Web服务器连接的次数和时间,把处理任务合理地分配给客户端和Web服务器,最终获得ASP.NET应用程序的高性能。

企业网站大多是一个企业级分布式应用系统,采用分布式多层架构技术,平滑地构建集群服务和负载均衡,从而提高网站的整体性能。故本文以ASP.NET三层架构模型理论为基础,从网站架构、页面优化、业务逻辑、数据访问、缓存技术、IIS配置等方面来进行探讨。

3 性能优化策略研究

3.1 网站架构策略

互联网不断扩大的规模、日益增长的用户群以及Web 2.0的兴起,要求企业网站必须具备高性能和高可扩展性,同时还要支持高度并发的访问。通常可以通过以下这些架构策略的优化来提高其性能。

(1)Web与DB服务器分离。因为Web和DB服务器都会占用大量的CPU和内存资源以及磁盘I/O。如果两者有效分离,既可以分散压力和提高负载承受能力,同时又可为以后的系统扩展奠定基础。

(2)数据大表拆分。对于比较大的数据表或者历史数据量比较大的数据表,可以根据拆分的原则按一定的逻辑进行拆分,然后通过索引表或分区表进行关联处理。这样可以提高数据查询速度和整体性能,同时避免了对现有系统数据库核心业务数据的影响。

(3)功能和展示必须分开。核心功能用脚本语言编写,前台展示使用带特殊标签的HTML编写,这样不仅能加快开发速度,而且为以后的维护和升级提供方便。对于前台模块,一般还需要将页面的头、尾单独提取出来,页面的主体部分也按模块或功能进行拆分,这样可以减轻服务器的负担。

(4)图片服务器分离。对于Web服务器来说,不管是IIS、Apache还是其它Web服务器,图片都是最消耗资源的。无论是从管理还是从性能上来说,都要尽量部署独立的图片服务器,将图片和页面进行分离。这样架构可以分散提供页面访问请求的服务器系统压力,并且可以保证系统不会因为图片问题而崩溃。在应用服务器和图片服务器上,可以进行不同的配置优化,保证更高的系统执行效率。

(5)读写分离。同时对数据库进行读、写操作是非常慢的,比较好的方法是根据读写的压力不同,分别建立两类结构完全相同的数据库服务器,

收稿日期:2010-08-30
*基金项目:四川省教育厅项目(项目编号:P09374);西昌学院科研项目(项目编号:ZZSSA0718)。
作者简介:岳付强(1979-),男,重庆忠县人,讲师,硕士,研究方向:数据库与数据挖掘以及软件过程技术与方法。

将负责写操作服务器的数据定时复制给负责读操作服务器。通过均衡服务器CPU和I/O消耗,不仅可以获得更丰富的服务器资源,还能支持暂时的过载。即使遇到突发事件或流量剧增,结果也只是系统的整体性能下降,而不是立即崩溃。

(6)封装使开发事半功倍。通过将ASP.NET网站分为表示层、逻辑层和持久层,分别进行封装,做到当某一层架构发生变化时,不会影响其它层。

(7)负载均衡。负载均衡是大型网站解决高负荷访问和大量并发请求经常采用的方法。负载均衡技术有很多专业的服务商和产品可以选择,其中最典型的是硬件四层交换和软件四层交换技术。

(8)扩容性应对流量突增。在设计架构大型企业网站的时候,必须考虑到以后可能的容量扩充。

3.2 页面优化策略

ASP.NET网页由Web窗体文件和隐藏代码文件两部分组成。前者由HTML代码构成,主要提供给用户操作界面;后者由C#等语言代码构成,实现控件的事件处理逻辑。页面表示层的优化主要从HTML页面优化和高效C#编码两方面着手。

(1)HTML页面优化。例如尽量减少HTTP请求数,通过合并JS和CSS文件、合并框架图片及相对变动较小的图片、通过CSS背景切割来完成渲染以及合理使用本地Cache来缓存JS/CSS/Image等;减小被请求文件的大小,以减小请求数据占用的网络带宽,例如删除JS/CSS/HTML文件中的空白换行和注释等、使用(X)HTML+CSS方式搭建网站结构、使用服务器端GZIP压缩JS/CSS文件大小等;通过版本化控制客户端缓存;以及其它一些页面处理技巧,例如在CSS中使用Visibility、保持同一URL的大小写一致、让标记有始有终、不要将整个页面内容塞到一个Table中、使用iframe框架嵌套另一个页面、把Javascript代码放到HTML文件末尾、尽量做文字友情链接、通过减少图片数或降低图片质量或使用恰当的图片格式来优化图片、网站后加斜杠和标明位置高度和宽度等。

(2)高效C#编码。例如用foreach代替for语句,因为foreach的平均花费时间只有for的30%;避免使用ArrayList,建议使用自定义的集合类型或泛型来代替,可以避免封箱和拆箱的发生,提高性能;存放少量数据时用HashTable代替其它如StringDictionary、NamValeCollection、HybirdCollection等字典集合类型;为字符串容器声明常量,不要直接把字符封装在双引号(“”)里面;不要使用UpperCase、Lowercase转换字符串进行比较,用String.Compare代替,因为它可以忽略大小写进行比较;使用StringBuilder代替“+”字符串连接符,可以大大节省资源,具有更佳的性能;如果只是从XML对象中读取数据,使用只读的XpathDocument代替XMLDocument,可以提高性能;避免在循环体内声明变量,而是在循环体外声明变量,在循环体内初始化变量;捕获指定的异常,不要使用通用的System.Exception;最好不要使用Exception控制程序流程,因为捕获异常是需要消耗性能的;使用using和try/finally来做资源清理;避免滥用反射,反射是比较浪费性能的操作;使用值类型的ToString方法避免装箱操作等。

3.3 业务逻辑优化选择

业务逻辑层是数据处理的最高层,用于对上下交互的数据进行逻辑处理,实现系统业务目标。它主要表现为众多的类文件,为Web表示层提供调用接口,同时又调用数据访问层。在ASP.NET中,业务逻辑的优化选择主要是通过高效的业务逻辑控制策略来现实的。

(1)如果没有必要,尽量使用静态HTML页面。ASP.NET页面程序实现了网页信息的动态交互,运行起来非常方便,因为它们的数据交互性好,能方便地存取、更改数据库的内容,使网站能“动”起来,如论坛、留言本等。但是这类程序必须先由服务器执行处理或查询数据库后生成HTML页面,然后再“送”往客户端浏览,这就需要耗费一定的服务器资源。如果在并发较大的企业网站过多地使用这类程序,则网页显示速度肯定会变慢。其实,可以把ASP.NET动态网页通过静态化方法转换成效率最高、消耗最小的HTML页面,避免大量的数据库访问请求,提高网站的性能。

(2)用好Page_Load事件和Page.IsPostBack属性,避免不必要的回送操作。由于HTTP协议是瞬时断开的,一旦数据传输完毕,HTTP协议就会立即断开浏览器和服务器的连接。每当用户执行新的操作时,又会重新连接该页面,必然会响应Page_Load事件。此时只需要检查Page.IsPostBack属性(该属性表示该页是否正为响应客户端回发而加载,或者它是否正被首次加载和访问,如果是首次加载为false,否则为true),如果为true,表示页面之前已经被加载过,就跳过Page_Load事件,这样就大大地提升了ASP.NET程序的性能。

(3)尽量在客户端进行用户输入验证。在开发页面时,经常会在C#代码中验证用户输入的内容是否合法、是否为数字格式、是否邮件格式、是否符合

逻辑等。通过.NET丰富的类库,确实非常方便,但由于是服务器代码,每次验证都需要回送到服务器处理,这样会大大影响系统的性能。所以在不是必须的情况下,尽量采用客户端(如JS的方式)进行验证。

(4)关闭不必要的Session状态。ASP.NET使用Session来保存用户授权的相关信息,这些信息是保存在服务器端的。如果提供了Session状态,则每次调用页面时,都会首先查询Session状态,这必然会影响页面执行的性能。如果只是普通的页面,与客户授权无关,则应关闭Session状态。关闭Session状态的方法是在ASP.NET的HTML代码中添加"<%@ Page EnableSessionState=" false " %>"语句。

(5)优先使用HTML控件,而不是服务器控件。在ASP.NET中提供了Server Control控件和标准的HTML控件。HTML控件只响应客户端事件,而Server Control控件提供了runat="Server"属性,服务器控件确实比HTML控件更加方便和灵活,但这是以牺牲服务器端的资源为代价的。因此在设计页面时,应根据实际情况来选择所需控件。如果只需要响应客户端事件,那么最好选择HTML控件,这样会大大提高ASP.NET的性能。

(6)不必要时,请关闭ViewState。使用ViewState时,每个对象都必须序列化到其中,然后通过回传进行反序列化,因此使用ViewState是有代价的。尽量减少使用ViewState对象,如果可能,尽量减少放入其中的对象的数目。在ASP.NET中,如果要禁止页面所有服务器控件的ViewState对象,则需要使用"<%@ Page EnableViewState=" false " %>"页面声明语句。

(7)在Web.Config文件中禁用调试模式。在部署生成应用程序之前,一定要记住禁用调试模式。如果启用了调试模式,则应用程序的性能将会受到非常大的影响。如果要禁用调试模式,只需在Web.Config中加入"<comlilation debug=" false " >"语句即可。

3.4 数据访问性能优化

数据访问层通过ADO.NET封装后的数据库访问类来完成与数据库的交互,它同时又要为业务逻辑层服务。往往可以通过对数据访问性能的优化,来提升ASP.NET企业网站的性能。

(1)选择合适的.NET数据库提供程序。ADO.NET数据访问中最常用的是OLEDB和SQLClient两种提供程序,由于SQLClient驱动程序不必通过OLEDB层的协议转换,因此效率较OLEDB方式高出30%~40%的性能。此外,目前针对特定数据库,都有相应的.NET数据提供程序,如Oracle.NET、Mysql.NET等。使用这些特定的提供程序,可以获得更好的性能。

(2)正确使用连接池,并及时关闭数据库连接。访问数据库需要反复地创建、打开和关闭连接,比较耗费服务器资源。ASP.NET中提供了连接池技术,可以用来改善打开和关闭数据库性能。不论是否采用了连接池技术,在每一次连接中都应该尽可能晚地打开连接,尽可能早地关闭连接,并且在关闭连接时一定要关闭连接期间建立的所有临时对象,并结束所有用户定义的事务,这样才能保证连接的正常关闭,避免系统资源的浪费。

(3)善用数据库的存储过程。存储过程是存储在服务器上的一组预编译的SQL语句,与SQL语句不同,它是一次性编译成可执行计划缓存在数据库中,避免了重复的解析过程,节约了时间。同时,存储过程存储在服务器中,减少了执行该过程所需的网络传输带宽和执行时间。另外,存储过程在服务器端运行,独立于ASP.NET程序,便于以后更新。在程序安全性方面,使用存储过程还可以避免SQL注入攻击。

(4)优化SQL命令。数据库每次解析和执行SQL语句都需要执行很多步骤。如果提交的SQL语句的质量不高甚至有逻辑错误,就会造成无谓的开销和时间浪费。为了避免这种情况,在使用SQL语句时可以采用下面这些优化策略,以提高查询效率。例如字段按需提取,避免使用"select *"的语句形式;尽量用exists代替select count(*),(NOT)exists代替(NOT)IN;尽量不要使用OR;避免使用NOT IN,可以使用LEFT OUTER JOIN代替;注意Where子句中的语句顺序,尽量与索引顺序保持一致;在编写SQL语句时,尽量缩小查询范围并限制返回的列数;尽量不要使用在服务器端排序的Order By语句,而是在客户端的Dataset中进行排序操作等。

(5)合理使用与数据库相关的控件和方法。例如DataReader适合只读操作,而Dataset适合读写操作;ExecuteNonQuery方法完成数据的更新操作而不需要返回结果集,ExecuteScalar只返回结果集中第一行的第一列数据;在使用数据绑定DataBinder时,直接使用DataRowView;在使用DataReader对象读取查询结果时,应该优先通过列的序号而不是列的名称来读取,可以避免列名到序号额外的转换开销;在使用Command对象时ExectuteReader方法会创建DataReader对象,需要很大的系统开销,所以当

从数据库中检索单个值，如执行count()、max()、avg()等聚合函数时，应该优先使用ExecuteScalar方法；如果需要反复执行一条SQL语句时，可以利用Command对象的Prepare()方法来提高效率等。

(6)此外，还可以通过其它一些编程技巧来提高数据库访问性能。例如跟踪监视数据库当前连接池状态、分页的数据访问、合理使用tempdb数据库、使用视图代替跨库操作、为表建立适当的索引以及尽量避免大事务操作和使用游标等。

3.5 系统缓存技术

缓存是用来提高计算机系统性能的一项技术，它将经常访问的数据或高成本数据保留在内存中。缓存通常分为客户端缓存和服务器缓存，前者驻留在客户端中由浏览器智能管理，而后者放在服务器端，又分为静态文件缓存和动态缓存。静态文件缓存放在IIS的Kernel内存中，由Https.SYS直接管理，性能非常高，而动态缓存管理起来却非常困难。在ASP.NET中，常见的动态缓存机制主要有传统缓存、页面输出缓存、页面局部缓存和编程实现的数据缓存等几种方式。

(1)传统缓存方式。它是将需要重复利用的数据放在Application或Session中保存。

(2)页面输出缓存。它将ASP.NET页面内容保存在服务器内存中。当用户请求该页面时，系统从内存中输出相关数据，直到缓存数据过期。在这个过程中，缓存内容直接发送给用户，不必再次经过页面处理生命周期。通常情况下，页面输出缓存对于那些不需要经常更新但是需要大量处理才能编译完成的页面特别有用。在ASP.NET中，只需在ASPX页面顶部加入“<%@ OutputCache Duration= " 60" VaryByParam= " none " %>”语句即可。

(3)页面局部缓存。它将页面部分内容保存在内存中以便响应用户请求，而页面其它部分内容则为动态内容。页面局部缓存主要有控件缓存和替换后缓存两种方式。在ASP.NET中，分别由UserControl控件和Substitution控件来具体实现。

(4)数据缓存。它以编程方式来缓存任何对象。在ASP.NET中，由System.Web.Cacheing命名空间中的Cache类来实现。

此外，系统缓存还可以利用文件缓存依赖、数据库缓存依赖、分布式缓存系统以及.NET架构下的分布式缓存项目来实现。缓存技术可以大大地提高应用程序的性能，对ASP.NET提供的缓存机制的理解和运用是任何ASP.NET网站开发人员都应该掌握的技巧。

3.6 IIS配置策略

Internet信息服务器(IIS)管理器是ASP.NET网站常用的Web服务器，其安装非常简单。但是IIS的管理却没有那么简单，因为使用一段时间后，管理员往往会遇到服务器性能不良的情况，这就需要对IIS服务器进行优化。一般情况下，可以通过监控IIS服务器并发数、采用GZIP压缩网站页面和有效配置网站应用程序池等方法来对IIS进行管理与优化，提高Web服务器的性能。

4 结束语

本文主要从网站架构、页面优化、业务逻辑、数据访问、缓存技术和IIS配置等几个方面，分析了开发ASP.NET企业网站所必须注意的优化细节，以便更好地提高企业网站的性能和承受能力。虽然每个细节看起来都不是很起眼，但所有细节累积起来将会使网站的性能大幅提升，尤其是大规模的企业网站。当然，ASP.NET企业网站的性能优化是一个多方面的、长期的和细致的工作，更需要在实践中去探索。

注释及参考文献：

[1]王茹.基于ASP.NET应用程序的性能优化[J].黑龙江科技信息，2010(10)：72-73.
[2]桂友武，桂友超.基于ASP.NET企业网站的性能优化探讨[J].企业技术开发，2009(4)：121-122.
[3]张龙.基于ASP.NET网站的系统架构和性能优化[J].硅谷，2008(24)：69.
[4]张艺博.ASP.NET性能优化设计[J].光盘技术，2008(12)：47-48.
[5]万荣泽.基于ASP.NET管理信息系统性能优化的研究与实践[J].广西轻工业，2008(12)：91-92.
[6]俞华锋.基于ASP.NET的电子商务网站的性能优化研究[J].科技创新导报，2007(31)：200-201.
[7]厉毅.优化ASP.NET应用程序性能的研究与探讨[J].福建电脑，2005(9)：67-68.
[8]李天平..NET深入体验与实战精要[M].北京：电子工业出版社，2009：525-599.

Research on the Performance Optimization of ASP.NET Enterprise Website

YUE Fu-qiang

(Xichang College, Xichang, Sichuan 615013)

Abstract: The performance optimization of ASP.NET enterprise website is extremely important to an enterprise, but the work of the website performance optimization is various and will take a long-term. This paper introduces the ideas of the website overall performance optimization, and then discusses the methods and techniques of the website performance optimization from the website architecture, page optimization, business logic, data access, cache, IIS website configurations and so on.

Key words: ASP.NET; Enterprise website; Performance optimization; Cache

(上接62页)

Association Rule Discovering Algorithm Based on Genetic Algorithm

ZHU Yan-ting

(Department of Computer Science, Guangxi Modern Polytechnic College, Hechi, Guangxi 547000)

Abstract: According to the requirement of association rule data mining, an genetic algorithm based on association rule data mining algorithm was developed. In addition to three basic operators, selection, crossover and mutation, the new operators-pick was included. The designing idea and algorithms were presented in detail. We design the experiment to test the performance of the algorithms. It is proved that the efficiency of the algorithms is excellent.

Key words: Data mining; Association rule; Genetic algorithm