构建高性能的 J2EE 分布式 Web 应用系统

Research on Building J2EE Web Application with High Performance

于　华　（山东财政学院计算机信息工程系 250014）

摘要：具有快速的响应速度是应用系统设计的标准之一，特别是对于多层分布式 Web 应用系统的开发来说，响应速度的快慢有时甚至影响到应用软件的使用价值。本文简要介绍了 J2EE 的概念和特点，并在实践的基础上，根据 J2EE 体系架构的技术特点，对设计和开发基于 J2EE 架构的分布式 Web 应用系统时应考虑的性能优化问题做了阐述、分析，并提出了一些解决问题的思路和方法。

关键词：J2EE　分布式　Web　性能优化

1　引言

J2EE 平台由一整套服务（Services）、应用程序接口（APIs）和协议构成，它对开发基于 Web 的多层应用提供了功能支持，并提出了（JSP、Java servlet、JDBC、EJB、RMI、JNDI…）等 13 种技术规范。

下面，我们将从 J2EE 应用程序体系结构出发，来讨论优化大型 Web 应用系统性能的若干策略。

2　结构优化

2.1　Java 基础优化

本质上来说，J2EE 应用程序就是有数据库操作的 Java 程序。因此 Java 编码技术的好坏也会直接影响 J2EE 应用的性能。下面是一些基本原则：

（1）对象生成和大小调整。JAVA 程序设计中一个普遍的问题就是没有好好的利用 JAVA 语言本身提供的函数，从而常常会生成大量的对象（或实例）。由于系统不仅要花时间生成对象，以后可能还需花时间对这些对象进行垃圾回收和处理。所以，生成过多的对象将会给程序的性能带来很大影响。因此，编码时要注意以下几点：

① 尽可能地使用静态变量（Static Class Variables）。如果类中的变量不会随实例而变化，就可以定义为静态变量，从而使所有的实例都共享这个变量。

② 不要对已生成的对象作过多的改变。对于一些类（如：String 类）来讲，宁愿再重新生成一个新的对象实例，而不应该修改已经生成的对象实例。

③ 生成对象时，要分配给它合理的空间和大小。JAVA 中的很多类都有它的默认的空间分配大小。避免在容量不够的时候自动增长，导致代价昂贵的复制操作，以提高性能。

④ 避免生成不太使用或生命周期短的对象或变量。对于这种情况，应该定义一个对象缓冲池。因为管理一个对象缓冲池的开销要比频繁的生成和回收对象的开销小的多。

⑤ 只在对象作用范围内进行初始化。JAVA 允许在代码的任何地方定义和初始化对象。这样，就可以只在对象作用的范围内进行初始化。从而节约系统的开销。

（2）慎用异常（Exceptions）。JAVA 语言中提供了 try/catch 来方便用户捕捉异常，进行异常的处理。但是如果使用不当，也会给 JAVA 程序的性能带来影响。因此，要注意以下两点。

① 避免对应用程序的逻辑使用 try/catch 。如果可以用 if，while 等逻辑语句来处理，那么就尽可能的不用 try/catch 语句。

② 重用异常。在必须要进行异常处理时，要尽可能的重用已经存在的异常对象。因为在异常处理中，生成一个异常对象要消耗掉大部分的时间。

（3）输入和输出（I/O）。输入和输出包括很多方面，但涉及最多的是对硬盘，网络或数据库的读写操作。对于读写操作，又分为有缓存和没有缓存的；对于

数据库的操作,有多种类型的 JDBC 驱动器可以选择。但无论怎样,都会给程序的性能带来影响。因此,需要注意如下几点:

① 使用输入输出缓存。尽可能地多使用缓存。但如果要经常对缓存进行刷新(flush),则建议不要使用缓存。

② 输出流(Output Stream)和 Unicode 字符串。当使用 Output Stream 和 Unicode 字符串时,Write 类的开销比较大。因为它要实现 Unicode 到字节(byte)的转换。因此,如果可能的话,在使用 Write 类之前就实现转换或用 OutputStream 类代替 Writer 类来使用。

③ 当需要序列化时使用 transient。当序列化一个类或对象时,对于那些原子类型(atomic)或可以重建的原素要标识为 transient 类型。这样就不用每一次都进行序列化。如果这些序列化的对象要在网络上传输,这一小小的改变对性能会有很大的提高。

④ 使用高速缓存(Cache)。对于那些经常要使用而又不大变化的对象或数据,可以把它存储在高速缓存中。这样就可以提高访问的速度。这一点对于从数据库中返回的结果集尤其重要。

(4) 线程(Threading)。一个高性能的应用程序中一般都会用到线程。因为线程能充分利用系统的资源。在其他线程因为等待硬盘或网络读写时,程序能够继续处理和运行。但是对线程运用不当,也会影响程序的性能。关于线程的操作,要注意如下几个方面。

① 防止过多的同步。不必要的同步常常会造成程序性能的下降。因此,如果程序是单线程,则一定不要使用同步。

② 同步方法而不要同步整个代码段。对某个方法或函数进行同步比对整个代码段进行同步的性能要好。

2.2 servlet 和 JSP 的性能优化

(1) 在 HttpServlet init() 方法中缓存数据。通过缓存来提高性能的好处是众所周知的事情。servlet 的 init() 方法在 servlet 的生命周期中仅调用一次。服务器会在创建 servlet 实例之后和 servlet 处理任何请求之前调用它,为了提高性能,可以在 init() 中缓存静态数据或完成要在初始化期间完成的代价昂贵的操作。例如,可以用它来缓存连接的数据源。在 J2EE 中,数据源是通过 JNDI(Java Naming Directory Interface)来查找获得的。如果每次使用 SQL 访问数据库时,都用 JNDI 去查找数据源,会严重影响系统的性能。Servlet 的 init() 方法可以用于获取 DataSource 并缓存它以便之后的重用。

(2) 禁用 servlet 和 Jsp 的自动装载功能。当每次修改了 Servlet/JSP 之后,就必须重新启动服务器。由于自动装载功能减少开发时间,因此在开发阶段该功能是非常有用的。但是,它在运行阶段是非常昂贵的;servlet/JSP 由于不必要的装载,增加类装载器的负担而造成性能的下降。因此,在运行环境中为了得到更好的性能,应关闭 servlet/JSP 的自动装载功能。

(3) 恰当使用线程池。servlet 引擎为每个请求创建一个单独的线程,将该线程指派给 service() 方法,然后在 service() 方法执行完后删除该线程。默认情况下,servlet 引擎可能为每个请求创建一个新的线程。由于创建和删除线程的开销很昂贵,于是这种默认行为降低了系统的性能。我们可以使用线程池来提高性能。根据预期的并发用户数量,配置一个线程池,设置好线程池里的线程数量的最小和最大值以及增长的最小和最大值。起初,servlet 引擎创建一个线程数与配置中的最小线程数量相等的线程池。然后 servlet 引擎把池中的一个线程指派给一个请求而不是每次都创建新的线程,完成操作之后,servlet 引擎把线程放回到线程池中。

(4) 为 servlet(及 JSP)对象设置合适的范围

servlet(及 JSP)支持四种不同的数据保存范围(或内存区域)。

Page:保存在这里的数据只在单个页面范围有效。

Request:保存在这里的数据在单个请求的范围内有效(在返回应答给调用者之前,数据在 servlet、JSP 页面之间传递)。

Session:保存在这里的数据会话期间一直有效。(数据的有效范围跨越多个请求直至请求超时或数据被显式地清除)。

Application:保存在这里的数据是全局数据,对应用的所有 servlet 和 JSP 都有效,除非数据被显式地清除,或者 servlet 容器重新启动。

在 servlet/JSP 编程中数据存储位置的选择非常重要,应该根据应用的需求选择正确的范围,否则它将影响应用的性能。四种范围之中,尤以 Session 范围对内

存占用的影响最大。

2.3 EJB 优化

(1) 尽量通过会话 Bean 访问实体 Bean。直接访问实体 Bean 对性能不利。因为当客户程序远程访问实体 Bean 时,每一个 get 方法都是一个远程调用。而访问实体 Bean 的会话 Bean 是本地的,能够把所有数据组织成一个结构,然后返回它的值。

用会话 Bean 封装对实体 Bean 的访问能够改进事务管理,因为会话 Bean 只有在到达事务边界时才会提交。每一个对 get 方法的直接调用产生一个事务,容器将在每一个实体 Bean 的事务之后执行一个"装入-读取"操作。如果实体 Bean 的唯一用途就是提取和更新数据,改成在会话 Bean 之内利用 JDBC 访问数据库可以得到更好的性能。

(2) 尽量使用 CMP(Container managed persistence,容器管理持久性)。CMP 方式不仅减少了编码的工作量,而且在 Container(容器)以及 container 产生的数据库访问代码中包括了许多优化的可能。因为 Container 可以访问内存缓冲中的 bean,所以它可以监视缓冲中的任何变化。在事务没有提交之前,如果缓存的数据没有变化就不用写到数据库中。这样就可以避免许多不必要的数据库写操作。

(3) 在企业 Bean 中把 ejbStore 中的数据库访问减小到最少。当使用 CMP 时,由于 ejbStore 不受 bean 的控制,它将所有的优化任务留给容器去处理,因此程序设计人员不必考虑优化的事情。但当使用 BMP(Bean managed persistence,Bean 管理持久性)时,由于容器不负责提供优化 BMP 的处理,因此开发设计人员必须自己进行优化处理。处理时,可以在配置 bean 的时候,为缓冲区保留一个数据改变标志 dirty(脏)标记。所有改变数据库中底层数据的操作,都设置标记,而在 ejbStore() 中,首先检测 dirty 的值,如果 dirty 的值没有改变,表明目前数据库中的数据与缓存的一致,就不必进行数据库操作了,反之,就要把缓存数据写入数据库。这样可以极大地提高系统的性能。

3 数据库访问优化

对于基于数据库的 Web 应用来说,性能的好坏在很大程度上也取决于对数据库的访问速度。为了提高数据库的访问速度,除了在数据库的物理设计、关系规范化等方面进行优化外,还应在应用系统的开发和设计上进行优化。对于 J2ee 数据库应用来说,可从以下角度对其进行优化。

3.1 恰当地使用数据库连接池技术

在数据库处理中,资源开销最大的是建立数据库连接操作。若每一个用户访问时,重新建立连接,不仅用户要长时间等待,而且系统有可能会由于资源消耗过大而停止响应。为了解决这一问题,在 JDBC2.0 中提出了 JDBC 连接池技术。JDBC 连接池的原理是:系统在内存中维持一个连接缓冲池,当某个线程需要访问数据库时,它向数据库连接池请求一个连接,然后用连接池返回的连接执行数据库操作(例如执行 select 或 update、delete 命令),操作结束后再把数据库连接对象返回给连接池,以便其他组件使用该连接。因此,可以大大地提高系统的响应速度,从而提高整个系统的性能。

但是,由于 J2ee 应用支持多个并发用户,连接池的规模也会极大地影响应用的性能。由于每一个应用都有不同的数据库访问请求,因此如何调节连接池中连接的数量必须根据应用的具体情况而定。有一条原则必须要记住:很多情况下,JDBC 连接池的规模往往是对应用的整体性能影响最大的因素之一。

3.2 使用参数标记作为存储过程的参数

调用存储过程时,使用参数标记做为参数,尽量不要用字符做参数。JDBC 驱动调用存储过程时要么象执行其他 SQL 查询一样执行该过程,要么通过 RPC 直接调用来优化执行过程。如果象 SQL 查询那样执行存储过程,数据库服务器先解析该语句,验证参数类型,然后把参数转换成正确的数据类型,显然这种调用方式不是最高效的。SQL 语句总是做为一个字符串送到数据库服务器上,例如,

```
CallableStatement cstmt = conn.prepareCall ( "
{call getCustName (12345)}" );
ResultSet rs = cstmt.executeQuery ();
```

在这种情况下,即使程序员设想给 getCustName 唯一的参数是整型,事实上参数传进数据库的仍旧是字符串。数据库服务器解析该语句,分离出单个参数值 12345,然后在把过程当作 SQL 语言执行之前,将字符串"12345"转换成整型值。

通过 RPC 在数据库服务器中调用存储过程,就能

避免使用 SQL 字符串带来的开销。且 JDBC 能以 RPC 方式直接在数据库中调用存储过程来优化执行,所以,执行时间也大大地缩短了。

3.3 使用批处理

更新大量的数据通常是准备一个 INSERT 语句并多次执行该语句,结果产生大量的网络往返。为了减少 JDBC 调用次数和提高性能,可以使用 PreparedStatement 对象的 addBatch( ) 方法一次将多个查询送到数据库里。

尽管使用批处理需要更多的数据库 CPU 运算开销,但性能可由减少的网络往返获得。为了提升 JDBC 驱动的性能,就要减少 JDBC 驱动和数据库服务器之间的网络通信量。

3.4 选择合适的游标

选择合适的游标能提高应用的灵活性。向前游标对连续读表中所有的行提供了优秀的性能。就检索表数据而言,没有一个检索数据的方法要比向前游标更快。然而,当应用必须处理非连续方式的行时,就不能使用它。

对需要数据库高层次的并发控制和需要结果集向前和向后滚动能力的应用而言,JDBC 驱动使用的无感知游标是最为理想的选择。对无感知游标的第一次请求是获取所有的行(或者当 JDBC 使用"懒惰"方式读时,可以读取部分行)并将它们存储在客户端。那么,第一次请求将会非常慢,特别是当长数据被检索到的时候。后续的请求不再需要网络交通(或当驱动采用懒惰方式时,只有有限的网络交通)并处理得很快。由于第一次请求处理缓慢,无感知游标不应该用于一行数据的单个请求。当要返回长数据时,内存很容易被耗尽,所以开发人员也应该避免使用无感知游标。

3.5 Statement 对象和 PreparedStatement 对象的选择

在执行 SQL 命令时,我们有二种选择:可以使用 PreparedStatement 对象,也可以使用 Statement 对象。在使用 PreparedStatement 对象执行 SQL 命令时,命令被数据库进行解析和编译,然后被放到命令缓冲区。然后,每当执行同一个 PreparedStatement 对象时,它就会被再解析一次,但不会被再次编译。在缓冲区中可以发现预编译的命令,并且可以重新使用。当使用 Statement 对象时,则每次执行一个 SQL 命令时,都会对它进行解析和编译。JDBC 根据不同的用途来优化性能,所以我们需要根据用途在 PreparedStatement 对象和 Statement 对象之间做出选择。

如果只执行一个单独的 SQL 语句,可以选择 Statement 对象;此外,使用 Statement 对象也可使得编写动态 SQL 命令更加简单,因为我们可以将字符串连接在一起,建立一个有效的 SQL 命令。

3.6 SQL 语句的优化

若一个 SQL 语句要执行很长的一段时间,对整个资源也将一直占用,会降低程序的执行效率,使系统没有足够的资源服务于其他用户。因此,应尽量使用优化过的语句。SQL 使用时应注意以下几点:

(1) 充分利用索引,避免相关子查询。嵌套层次越多,效率越低。

(2) 避免对大型表行数据的顺序存取,这有可能对查询效率造成致命的影响。

(3) 使用临时表可加速查询。把表的一个子集进行排序并创建临时表,有时能加速查询。

(4) 避免返回成百上千的数据记录。如果返回成百上千的数据记录的话,系统将会耗费大量的资源和时间来保存。

4 结束语

如今,分布式 Web 应用正在成为商业应用的主流,一个 Web 应用的成败很大程度上取决于其响应速度的快慢。以上提到的是为了提高 Web 应用系统的性能而在开发中使用的策略和方法,在实际应用开发中恰当使用这些方法和策略,可以达到提升 J2EE Web 应用系统总体性能的目的。

参考文献

1 IBM Websphere Application Server White Paper: WebSphere Application Server Development Best Practices for Performance and Scalability. 2000.

2 Sucharitakul A. http://developer.java.sun.com/developer/technical Articles/ebeans/sevenrules.

3 Subrahmanyam Allamaraju. J2EE 服务器端高级编程[M],机械工业出版社, 2001. 1038210781.