WAS线程池与连接池调优

周佳

WAS 二线技术支持工程师

目标

- 线程池
- 数据源连接池
- 数据库连接池常见问题分析

线程池

- WAS里使用线程池来重用线程，降低在运行时创建线程的开销。
- WAS里有下面几种常用的线程池：
  - WebContainer 线程池 ：处理HTTP请求
  - ORB 线程池 ：处理EJB的RMI/IIOP请求
  - SIB/消息处理池：WAS自带的default messaging的线程池
  - Default 线程池：MDB或者未指定的 transport chain
  - WMQRA线程池：与MQ server交换消息的线程池

- 可以使用PMI的线程池相关的监测数据来观察服务器的线程使用情况

线程池的通用属性

- 要查看此管理控制台页面，单击服务器 > 服务器类型 > WebSphere 应用程序服务器 > server_name > 线程池，然后选择需要配置的线程池。
- 最小线程数-指定池中允许的最小线程数
- 最大线程数-线程池中维护的最大线程数
- 线程不活动超时-指定在收回线程之前应该经过的不活动的毫秒数。为 0 的值表明不等待而负值（小于 0）意味着永远等待。
- 允许线程分配超过最大线程大小-线程数是否可以超过线程池配置的最大大小。如果选中，那么可创建的线程数量仅受JVM和操作系统限制。

WEBCONTAINER线程池

- 如果系统资源允许，建议将webcontainer线程池设置为 minimum = maximum

- 从CPU个数 * 10 开始作为初始值设置 并通过测试来调整
- 在WAS 6.0以前，并发的用户数量和线程数是一对一的关系
- 从WAS 6.0开始，通过使用AIO 和NIO，可以同时处理更多数量的并发用户请求
- 还要考虑到数据源连接池的大小来设置

ORB 线程池

- 如果使用EJB应用时会用到ORB线程池
- 可以从CPU个数 *5 开始作为初始值设置 并通过测试来调整
- 缺省值是 min = 10，　max = 50

线程泄漏

- 当线程被销毁的时候，有可能会泄漏ThreadLocal对象，造成潜在的Native 内存问题
- 线程号可能会变得非常大，超过线程池配置的大小
- 日志里会打印出Malloc 错误信息：
    - JVMDBG001: malloc failed to allocate …
- 无法创建线程 （常常会与OOM伴随而来 java.lang.OutOfMemoryError: Failed to create a thread）
- 解决办法：**min=max 并且不允许线程分配超过最大线程大小。**这样线程创建后不会被销毁，从而避免了销毁的线程泄露ThreadLocal对象

目标

- 线程池
- 数据源连接池
- 数据源连接池常见问题分析

数据源连接池

- 从WAS 6.1开始，实现了JCA规范，该规范提供一个标准的机制去连接 EIS(Enterprise Information System)和关系数据库
- 使用连接池的好处在于，可以降低应用频繁开启和关闭连接的开销
- 当一个连接被应用使用完毕后，应用虽然调用了connection.close() 方法，该连接并不是直接关闭而是放回到连接池里以便满足别的应用的请求
- 一个数据源对应一个连接池，分为三个子池：共享连接池，空闲池，以及非共享连接池

数据源连接池参数

- 连接池常用的参数有：
  - 最大连接数 Maximum connections
  - 最小连接数 Minimum connections
  - 连接超时 Connection timeout
  - 未使用超时 Unused timeout
  - 时效超时 Aged timeout
  - 收集时间 Reap time
  - 清除策略 Purge policy
  - 语句高速缓存 Statement Cache
- 尽量使用最新版本的JDBC 驱动。新版本里提供的性能优化会很有帮助。尤其是在升级完JDK之后，要使用与新的JDK相匹配的驱动。

最大连接数 MAXIMUM CONNECTION

- 此属性定义了该数据源连接池里能够创建的最多的数据库物理连接
- 当连接池里的连接数达到了最大连接数时，新的连接无法再被创建
- 当数据源是配置在集群，节点或单元时， 每一个包含在里面的应用服务器都会创建自己的连接池，那么假设有N个服务器，最大连接数是M，这个系统连接到后台数据库的最多的连接个数= N*M，因此需要确保后台数据库能够承担如此多的连接数
- **通常来说，Webcontainer的线程池应该比最大连接数要大，这样保证连接的充分利用。在实践中常常以2倍作为测试的起点。**
- 需要根据应用的需要和压力，以及后台数据库能负担的连接数大小，来调整这个参数。

最小连接数 MINIMUM CONNECTION

- 此属性定义了在空闲池里保存的最小的连接数量
- 当服务器启动的时候，WAS并不会在初始化的时候自动创建这么多个连接
- 只有当空闲池里空闲连接低于这个值，并且这些连接的unused timeout已经到了的时候，这些物理连接不会被关掉；当空闲池里有超过这个值的连接空闲并超过unused timeout时候，才会被抛弃。
- 如果你将Aged timeout 设置为一个大于0的数值时（缺省是0）,那么不管最小连接数是否达到，只要超过这个时间且空闲，连接就会被抛弃
- 推荐设置最小连接数为**0**。如果连接空闲了太久而没被清除，它们容易出现网络问题，而WAS是不会知晓的。当应用用到这些 废连接时就会收到StaleConnectionException.

连接超时 CONNECTION TIMEOUT

- 当应用请求连接，但是空闲池里没有空闲连接时，连接请求会等待一段时间看看有没有可用的连接，这段时间就是由连接超时（Connection timeout）
- 如果超过了这个时间仍没有可用的连接，那么该请求就会抛出createOrWaitForConnection 的错误：
  - J2CA0045E: Connection not available while invoking method createOrWaitForConnection for resource jdbc/xxxxxx
- 这个属性并不会影响现有的正在使用的连接
- 如果您将这个值设为0，那么连接请求就会无止境的等待下去直到有可用的连接。这样做会导致线程挂起，因此不建议这么设置
- 由于每个系统和应用都有自己的特点，因此没有一个适用于所有客户所有场景的推荐值。**缺省这个属性是180秒，这样设置对于大多数用户都是可以接受的**。如果您想降低请求等待的时间，可以适当的降低这个属性。

未使用超时 UNUSED TIMEOUT

- 该属性控制在空闲池里超过一定时间未被重新使用的空闲连接何时被关闭
- 如果一个空闲连接未使用超时的值已经达到，那么连接池的维护线程会先去查看该连接池的最小连接数
  - 如果空闲池里的空闲连接数量超过了最小连接数，那么维护线程会把这个超时的连接关闭
  - 如果空闲连接数小于或是等于最小连接数，那么该连接还会继续留在空闲池里，即使超过了未使用超时
  - 举例：假如最小连接数设为5，空闲连接数<5的时候，即使空闲超过了未使用超时，不会清除这些连接；空闲连接数>5的时候且有连接空闲超过了这个时间，那么下次连接池维护线程运行的时候会把这些超时的连接清除，但仍需保证连接池里有5个连接。

未使用超时 UNUSED TIMEOUT

- **这个属性也没有一个适合各种情况的推荐值**
- 它只会影响在空闲池里的空闲连接，并不会影响正在被应用使用的连接
- 如果设成0，那么空闲池里的连接将永远不会超时。那么这样会导致在某些情况下等待了很久的空闲连接变得不可用或者
- 很多情况下，如果系统中存在防火墙或者其它设备，它们有可能有设置在一定时间后切断连接，此时可以尝试将未使用超时设置为该设备上设置的超时值的一半，以降低出现StaleConnectionException的概率，但是并不能完全避免该错误。

时效超时 AGED TIMEOUT

- 该属性用于指定丢弃物理连接的时间间隔
- 由于该属性是作用于物理连接的，所以它可能会影响到正在使用的共享、非共享连接，以及正在空闲池里等待重用的空闲连接上
  - 当一个在空闲池里的连接已经创建超过这个值时，它会在下一次维护线程运行的时候关闭这个物理连接。在这种情况下会不考虑最小连接数的影响。例如，如果最小连接数设为10，现在只有5个空闲连接，这个超时的物理连接仍旧会被关闭。
  - 如果一个连接正在参与事务的时候超过了这个时间，那么该连接会被标记为 ‘即将关闭’ 。等到事务结束的时候，该连接就会被马上关闭。
- **这个属性的推荐值是0**，以便于让连接尽可能的被重用。
- 只有在WAS与数据库之间存在防火墙或者其它设备终止连接时才建议设置为一个大于0的值

时效超时 AGED TIMEOUT

- 当有些情况下必须将最小连接数设定的比0大的时候，尤其是需要设到10或者20那么高的时候，很有可能出现的情况是，有些物理连接会一直呆在连接池里直到服务器停止。即便未使用超时已经达到，但是由于最小连接数的影响，它们仍旧不会被销毁。有的情况下物理连接有可能几天都不关闭。 为了避免这种情况的发生，可以适当的设置时效超时，将它设置为一个很高的值，例如几个小时，这样就可以避免经常关闭连接，又能可以避免连接一直在开启状态

收集时间 REAP TIME

- 此属性定义连接池维护线程运行的频率。维护线程负责将超过未使用超时和时效超时的连接关闭。
- 如果设的过低，维护线程就会频繁的启动，可能会造成一些性能上的影响。**缺省的180秒适用于大多数的场合。**

清除策略 Purge policy

- 这个属性定义了在出现致命的连接错误，或者遇到StaleConnectionException异常时，是清除掉连接池里的所有连接，还是只清掉出了问题的这个连接
- 推荐的设定是**EntirePool**。这样设置，对于出现失效连接时非常有用。例如， 如果数据库宕机，那么连接池里的所有连接都会失效。当应用想要使用这些连接的时候，SCE会被抛出。在这种情况下，将所有连接一起清除比一次次清除出现故障的连接要有效率的多。

语句高速缓存 STATEMENT CACHE

- 这个属性定义了每一个物理连接可以高速缓存的语句数。在关闭语句时，WAS会缓存起来，以便提高效率
  - PreparedStatement 和 CallableStatement
- 缓存数设太小，会经常丢弃旧的缓存，这样没有达到缓存的效果；如果设的太高，那么有可能会造成资源紧张，尤其是内存。
  - **缺省值是10。没有建议值。**一开始可以用应用代码里不同的PreparedStatement 和 CallableStatement数量相加作为初始值，但是如果这个值比较大，会容易造成OOM。
- 可以通过PMI的PreparedStmtCacheDiscardCount来监测语句高速缓存的设置是否合适
- 如果有一些PreparedStatement 并不会被频繁使用，建议将他们改成普通的Statement（继承java.sql.Statement 接口），因为普通的Statement对象并不会被缓存。

目标

- 线程池
- 数据源连接池
- 数据源连接池常见问题分析

ConnectionWaitTimeoutException

- 当连接池满了，没有空闲连接时会出现
- 可以在日志里搜索关键字J2CA0045E
- 检查这个连接池：
  - 是否最大连接数设的太小，不够应用使用
  - 是否将connection wait timeout 设的太低
  - 代码是否正确的关闭连接
  - 是否正确的使用了共享连接
  - 是否后台数据库响应太慢

连接泄漏

- 当应用代码没有调用connection.close() 方法强制关闭连接时，有可能导致连接泄漏，**所以开发人员必须要遵从get/use/close的设计模式，把用到的资源，包括连接connection，session，resultset，preparedstatement等都在使用完后关闭**
- 也可能在servlet应用里出现。即使代码里关闭了连接，但是如果servlet会运行相对较长的时间，那么这个连接也会一直被占用无法被其他线程使用，造成无空闲连接的局面

共享连接

- 共享连接 – 一个应用的不同部分在申请连接的时候，可以同时获得同一个共享连接的句柄。这样可以更进一步的降低应用使用的连接数。

共享连接

- 只有当在同一个共享范围里申请共享连接，才会返回同一个连接给应用。最常见的共享范围是事务。当同一个事务里不同处代码请求连接时才会可能获得同一个物理连接的句柄。
  - 全局事务 global transaction(user transaction) 是由J2EE应用发起的用户事务，或者EJB容器发起的容器管理的事务。全局事务由外部的事务管理器来管理。
    - com.ibm.ws.Transaction.JTA.TransactionImpl@103a048#tid=61010
  - 本地事务Local Transaction Containment (LTC)：如果不是一个全局事务，那么就是一个本地事务。本地事务是由该资源来管理。
    - com.ibm.ws.LocalTransaction.LocalTranCoordImpl@3ba43ba4

共享连接

- 在WAS里，缺省使用的是共享连接
- 全局事务里，连接可以被同时使用，真正的共享
- 本地事务里，连接其实是被线性使用的 (get/use/close, get/use/close, etc)
- 本地事务特点：如果应用关闭了连接，该连接并不会立刻回到空闲池，而是留在共享连接池以便该本地事务继续使用，直到LTC结束。

典型 LTC 用例： servlet

- LTC A begin
  - Servlet A begin
    - get Connection A //shareable
    - use Connection A
    - close Connection A
    - // Connection A remains in shared pool, associated with LTC A
    - include or forward Servlet B
    - LTC A suspend
    - LTC B begin
      - Servlet B begin
        - get Connection B //shareable
        - use Connection B
        - close Connection B
        - // Connection B remains in shared pool, associated with LTC B
      - Servlet B end
    - LTC B end //Connection B released to free pool
    - LTC A resume
    - // possible Connection A re-use (get/use/close)
  - Servlet A end
- LTC A end //Connection A released to free pool

非共享连接

- 非共享连接并不会被应用的其他部分共享
- 使用非共享连接的部分代码完全掌握连接的生命周期
- 必须在代码里关闭非共享连接，避免连接泄漏

共享连接 VS 非共享连接

- 共享连接在下面的使用案例里会比较合适：
  - 应用可以很快的处理完一个HTTP请求
  - 在一个请求里会经常需要打开和关闭数据库连接
  - 有些EJB容器的事务需要使用共享连接
- 某些情境下，非共享连接可能会是更好的选择：
  - 处理一个HTTP请求需要很长的时间
  - 应用并不经常打开和关闭连接
  - 应用不怎么使用全局事务

连接泄漏

```
Connection Leak Logic Information:
  MCWrapper id 385d385d  Managed connection WSRdbManagedConnectionImpl@56d556d5  State:STATE_TRAN_WRAPPER_INUSE Thread Id: 00000153 Thread Name: WebContain
  er : 104 Handle count 1
     Start time inuse Fri Nov 25 09:19:22 CST 2011 Time inuse 29 (seconds)
     Last allocation time Fri Nov 25 09:19:22 CST 2011
       getConnection stack trace information:
          com.ibm.ejs.j2c.ConnectionManager.allocateConnection(ConnectionManager.java:895)
          com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:668)
          com.ibm.ws.rsadapter.jdbc.WSJdbcDataSource.getConnection(WSJdbcDataSource.java:635)
          org.springframework.jdbc.datasource.DataSourceUtils.doGetConnection(DataSourceUtils.java:113)
          org.springframework.jdbc.datasource.DataSourceUtils.getConnection(DataSourceUtils.java:79)
          [redacted]getConnection(DbService.java:63)
          [redacted]executeQuery(DbService.java:108)
          [redacted]getQueryLevel3MovaUseInfoHTML(rptKpiStatus.java:12164)
          [redacted].doQueryLevel3MovaUseInfo(rptKpiStatus.java:11872)
          sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
          sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:60)
          sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:37)
          java.lang.reflect.Method.invoke(Method.java:611)
          [redacted].dispatch(CoreForm.java:405)
          [redacted].execute(CoreForm.java:311)
          [redacted]execute(DispatchOperationStep.java:68)
          [redacted].doExecute(Unknown Source)
          [redacted]execute(Unknown Source)
          [redacted]executeRequest(Unknown Source)
          [redacted]processRequest(Unknown Source)
          [redacted]processRequest(HtmlRequestHandler.java:60)
          [redacted]service(Unknown Source)
          javax.servlet.http.HttpServlet.service(HttpServlet.java:831)
          com.ibm.ws.webcontainer.servlet.ServletWrapper.service(ServletWrapper.java:1657)
          com.ibm.ws.webcontainer.servlet.ServletWrapper.service(ServletWrapper.java:1597)
```

解决方法

- 改变缺省的使用方式，让应用去获得非共享连接
  - Use custom property ‘defaultConnectionTypeOverride’ or ‘globalConnectionTypeOverride’
    - https://www.ibm.com/support/knowledgecenter/SS7JFU_7.0.0/com.ibm.websphere.express.doc/info/exp/ae/tdat_conpoolman.html
  - 在全局事务里使用共享连接，提高共享的效率 – 可以在代码里实现 UserTransaction 接口来开启全局事务

StaleConnectionException

- 表明当前应用使用的连接已经不可用，通常伴随着数据库返回的SQL错误信息和代码
- 可以通过将min conn 设置为 0 ，并设置合适的unused timeout 来避免
- 还可以通过使用连接池的验证连接的功能来规避这个问题
- 查找是否有可能的网络问题，还有防火墙，代理服务器或者路由器的设置，因为它们可能在中间断开物理连接而WAS无从知晓

死锁

- 一个应用线程可能会获取超过一个连接，当数据库连接池无法同时满足已获得连接的线程对下一个连接的需求时会出现死锁
- 问题诊断– 收集 javacores/ Javadumps 和 j2c trace

问题诊断

- Trace specification (可以在运行时开启):
  - 标准使用的trace: *=info:WAS.j2c=all:RRA=all:Transaction=all
  - 如何开启trace：登陆管理控制台，点击troubleshooting -> logs and trace -> 选择服务器，点击change log detail levels，在文本框输入上述trace字符串。注意这里有两个tab，configuration tab里设置的trace需要重启生效，在runtime tab里面设置的trace 保存后即生效。

监控连接池

- 使用PMI里与数据源相关的统计数据来记录，并用TPV来观看结果和报表
- 使用wsadmin 脚本来获得当前数据源的连接池使用状况

使用wsadmin 脚本

- 使用 showPoolContents 可以给出和trace中一样的连接池信息
- Sample Jython commands:
  - wsadmin>ds = AdminControl.queryNames("*:name=<DISPLAY NAME OF DATASOURCE>,process=<SERVER NAME>,node=<NODE NAME>,j2eeType=JDBCDataSource,*")
  - wsadmin>AdminControl.invoke(ds, "showPoolContents")
  - 例子可参照下面文档的PART 2
  - http://www-01.ibm.com/support/docview.wss?uid=swg21385033

使用wsadmin 脚本

问题?