# 现象

系统页面卡住，用户无法操作。

# 第1次调优

## 分析

Jvm内存快使用完。

## 解决办法

设置JVM的初始和最大内存都为6G.

# 第2次调优

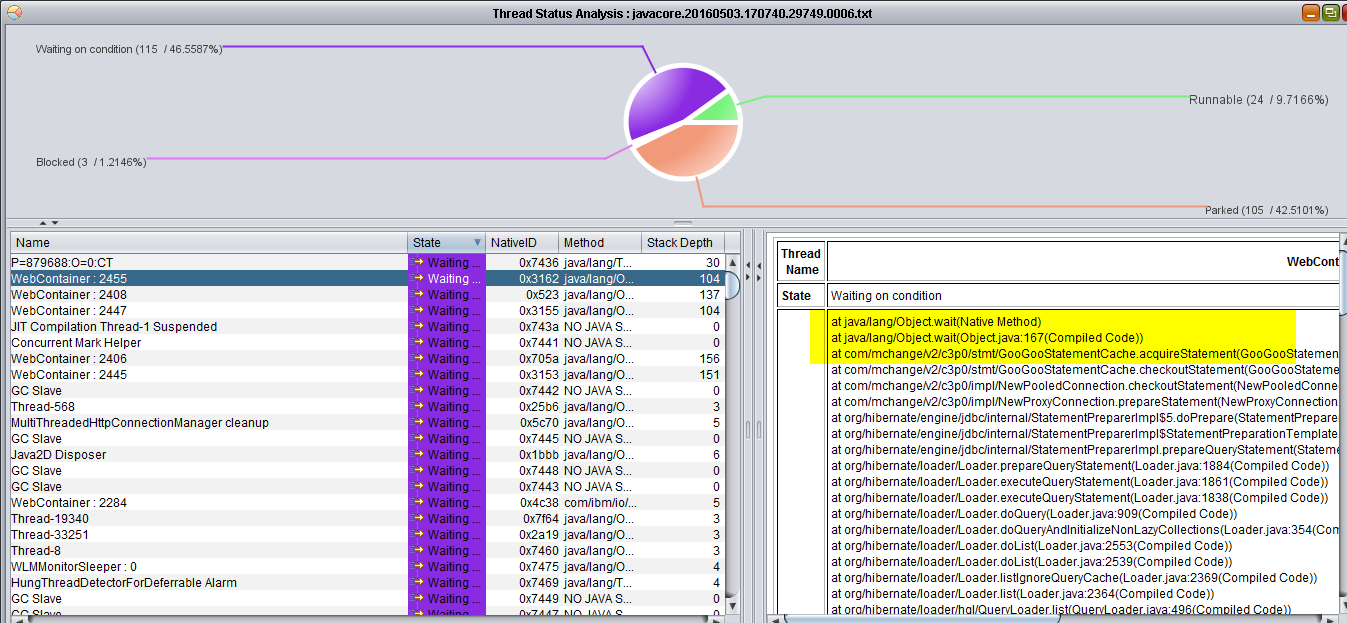
## 分析

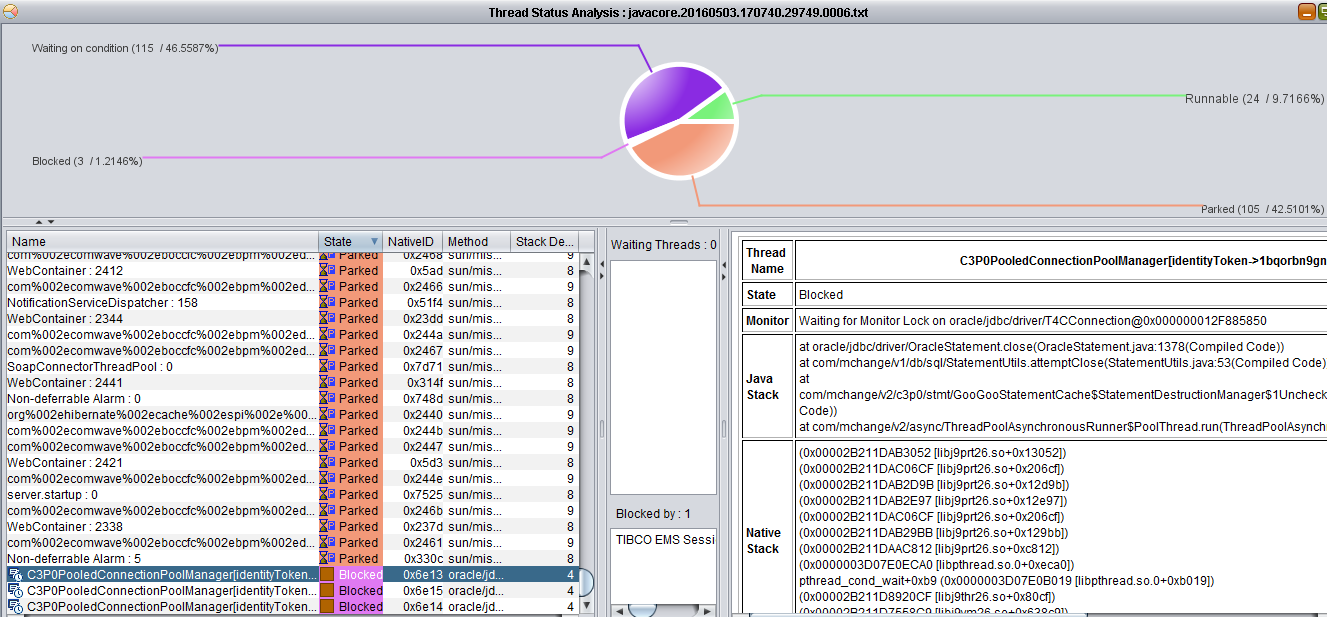
之前分析内存占用很大，主要是有部分代码缓存了很多对象，已修改部署；

但卡住的现象还是未解决，进一步分析 java core 的情况：

1. 系统线程包含很多Waiting on condition的线程，这些进程很多是等待C3P0数据源获取statement；
2. 系统线程包含3个blocked的线程，也是c3p0数据源在处理statament的时候卡住；

故可以判断是c3p0数据源配置需要优化。





## 解决办法

从生产配置看max\_statements配置过小只配置了100，建议调整为2000。

persistence.xml调整以下配置：

|  |
| --- |
| <property name=*"hibernate.c3p0.min\_size"* value=*"50"* />  <property name=*"hibernate.c3p0.max\_size"* value=*"500"* />  <property name=*"hibernate.c3p0.acquire\_increment"* value=*"5"* />  <property name=*"hibernate.c3p0.idle\_test\_period"* value=*"300"* />  **<property name=*"hibernate.c3p0.max\_statements"* value=*"2000"* />**  <property name=*"hibernate.c3p0.timeout"* value=*"120"* /> |

# 第3次调优

## 分析

较多处理request对象的请求处于 waiting on condition状态，应是was处理能力不足。

## 解决办法

调整WAS的线程池配置，加到线程池大小，从原来的200改为500； 最小设置为20；



# 第4次调优

日期： 2016-05-16

## 分析

* + 1. 数据源

Log中有较多以下错误，原因是c3p0数据源使用已经被关闭的connection连接：

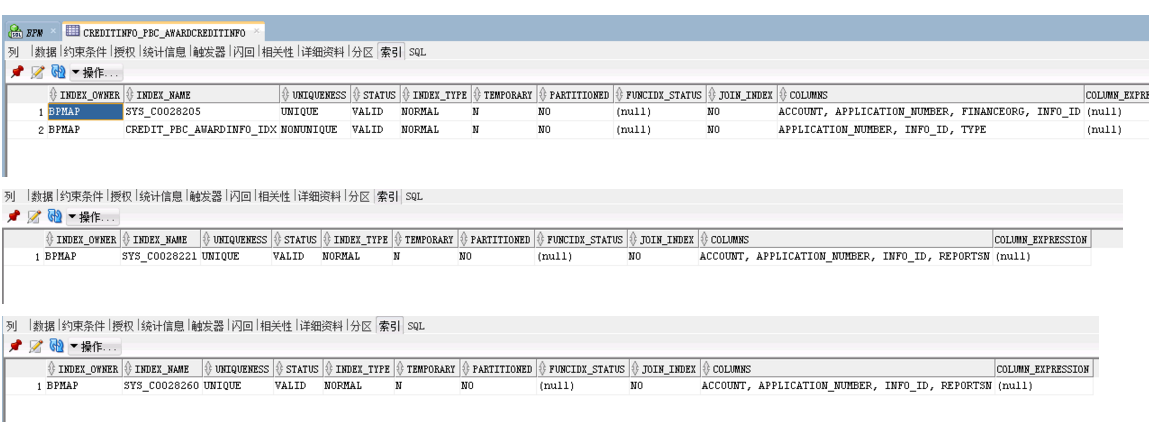
|  |
| --- |
| [5/16/16 12:48:32:521 CST] 0000005d DynamicPooled W com.mchange.v2.c3p0.management.DynamicPooledDataSourceManagerMBean getAttribute Failed to get requested attribute: numBusyConnections  java.lang.reflect.InvocationTargetException  at sun.reflect.GeneratedMethodAccessor256.invoke(Unknown Source)  at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:37)  at java.lang.reflect.Method.invoke(Method.java:611)  at com.mchange.v2.c3p0.management.DynamicPooledDataSourceManagerMBean.getAttribute(DynamicPooledDataSourceManagerMBean.java:366)  at com.sun.jmx.interceptor.DefaultMBeanServerInterceptor.getAttribute(DefaultMBeanServerInterceptor.java:678)  at com.sun.jmx.mbeanserver.JmxMBeanServer.getAttribute(JmxMBeanServer.java:651)  at com.blueware.monitor.collector.jmx.JmxService.a(JmxService.java)  at com.blueware.monitor.collector.jmx.JmxService.a(JmxService.java)  at com.blueware.monitor.collector.jmx.JmxService.beforeHarvest(JmxService.java)  at com.blueware.monitor.collector.stats.HarvestServiceImpl.a(HarvestServiceImpl.java)  at com.blueware.monitor.collector.stats.HarvestServiceImpl.a(HarvestServiceImpl.java)  at com.blueware.monitor.collector.stats.e.g(e.java)  at com.blueware.monitor.collector.stats.e.f(e.java)  at com.blueware.monitor.collector.stats.e.run(e.java)  at com.blueware.monitor.util.g.run(g.java)  at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:450)  at java.util.concurrent.FutureTask$Sync.innerRunAndReset(FutureTask.java:328)  at java.util.concurrent.FutureTask.runAndReset(FutureTask.java:161)  at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.access$101(ScheduledThreadPoolExecutor.java:109)  at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.runPeriodic(ScheduledThreadPoolExecutor.java:191)  at java.util.concurrent.ScheduledThreadPoolExecutor$ScheduledFutureTask.run(ScheduledThreadPoolExecutor.java:215)  at java.util.concurrent.ThreadPoolExecutor$Worker.runTask(ThreadPoolExecutor.java:908)  at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:931)  at java.lang.Thread.run(Thread.java:773)  Caused by: java.sql.SQLException: com.mchange.v2.c3p0.PoolBackedDataSource@3acae63 [ connectionPoolDataSource -> com.mchange.v2.c3p0.WrapperConnectionPoolDataSource@6fa3e8ea [ acquireIncrement -> 5, acquireRetryAttempts -> 30, acquireRetryDelay -> 1000, autoCommitOnClose -> false, automaticTestTable -> null, breakAfterAcquireFailure -> false, checkoutTimeout -> 0, connectionCustomizerClassName -> null, connectionTesterClassName -> com.mchange.v2.c3p0.impl.DefaultConnectionTester, debugUnreturnedConnectionStackTraces -> false, factoryClassLocation -> null, forceIgnoreUnresolvedTransactions -> false, identityToken -> 1bqorbo9g19yj9d71w4paqq|-187ef8d7, idleConnectionTestPeriod -> 300, initialPoolSize -> 50, maxAdministrativeTaskTime -> 0, maxConnectionAge -> 0, **maxIdleTime -> 120, maxIdleTimeExcessConnections -> 0, maxPoolSize -> 500, maxStatements -> 2000, maxStatementsPerConnection -> 0, minPoolSize -> 50**, nestedDataSource -> com.mchange.v2.c3p0.DriverManagerDataSource@3dbbcabd [ description -> null, driverClass -> null, factoryClassLocation -> null, identityToken -> 1bqorbo9g19yj9d71w4paqq|-646c9f5d, jdbcUrl -> jdbc:oracle:thin:@bpmdb:1521:ORCL, properties -> {user=\*\*\*\*\*\*, password=\*\*\*\*\*\*} ], **preferredTestQuery -> null, propertyCycle -> 0, statementCacheNumDeferredCloseThreads -> 0, testConnectionOnCheckin -> false, testConnectionOnCheckout -> false**, unreturnedConnectionTimeout -> 0, usesTraditionalReflectiveProxies -> false; userOverrides: {} ], dataSourceName -> null, factoryClassLocation -> null, identityToken -> 1bqorbo9g19yj9d71w4paqq|60cdb265, numHelperThreads -> 3 ] has been closed() -- you can no longer use it.  at com.mchange.v2.c3p0.impl.AbstractPoolBackedDataSource.assertCpds(AbstractPoolBackedDataSource.java:507)  at com.mchange.v2.c3p0.impl.AbstractPoolBackedDataSource.getPoolManager(AbstractPoolBackedDataSource.java:519)  at com.mchange.v2.c3p0.impl.AbstractPoolBackedDataSource.getNumBusyConnections(AbstractPoolBackedDataSource.java:170)  ... 24 more |

* + 1. 索引

以下查询语句第一次比较慢：

|  |
| --- |
| SELECT rpi.ACCOUNT AS BusinessId  ,replace(cai.OPENDATE, '.', '-') AS PayDate  ,cai.FINANCEORG AS PayAgency  ,cai.TYPE  ,cai.CREDITLIMITAMOUNT AS CreditAmount  ,cai.GUARANTEETYPE AS GuaranteeType  ,cai.ACCOUNTSTATUS AS STATE  ,rpi.SHARECREDITLIMITAMOUNT AS ShareAmount  ,rpi.USEDCREDITLIMITAMOUNT AS UsedAmount  ,rpi.LATEST6MONTHUSEDAVGAMOUNT AS AvgUsedAmountLast6Month  ,rpi.USEDHIGHESTAMOUNT AS MaxUseAmount  ,rpi.SCHEDULEDPAYMENTAMOUNT AS CurMonthNeedRepay  ,replace(rpi.SCHEDULEDPAYMENTDATE, '.', '-') AS BillDate  ,rpi.ACTUALPAYMENTAMOUNT AS CurMonthActualRepay  ,replace(rpi.RECENTPAYDATE, '.', '-') AS LastRepayDate  ,cod.CURROVERDUECYC AS CurOverDueNum  ,cod.CURROVERDUEAMOUNT AS CurOverdueAmount  ,cod.OVERDUEOVER180AMOUNT AS UnpaidBalanceOverdraw180Days  ,CURRENCY  FROM BPMAP.CREDITINFO\_PBC\_AWARDCREDITINFO cai  LEFT JOIN BPMAP.CREDITINFO\_PBC\_REPAYINFO rpi ON cai.REPORTSN = rpi.REPORTSN  AND cai.ACCOUNT = rpi.ACCOUNT  AND cai.INFO\_ID = rpi.INFO\_ID  LEFT JOIN BPMAP.CREDITINFO\_PBC\_CURROVERDUE cod ON cod.INFO\_ID = cai.INFO\_ID  AND cod.REPORTSN = cai.REPORTSN  AND cod.ACCOUNT = cai.ACCOUNT  WHERE cai.INFO\_ID = 'AC9A0F9118BE452CB594913A11160B94'  AND cai.APPLICATION\_NUMBER = 'BO20160516002462'  AND cai.TYPE = '贷记卡'; |

CREDITINFO\_PBC\_AWARDCREDITINFO没有把查询关键字REPORTSN加索引：



## 解决办法：

* + 1. 数据源

Versions：

JDBC: jdbc4 support

C3P0: 0.9.2.1

persistence.xml增加以下配置：

<property name="hibernate.c3p0.testConnectionOnCheckout" value="false" />

<property name="hibernate.c3p0.testConnectionOnCheckin" value="true" />

<property name="hibernate.c3p0.idleConnectionTestPeriod" value="300" />

<property name="hibernate.c3p0.preferredTestQuery" value="SELECT 1 from dual" />

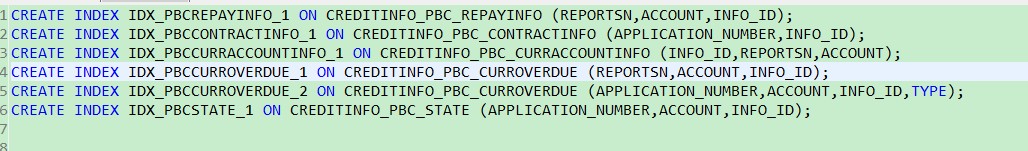
参考： <http://www.mchange.com/projects/c3p0/#configuring_connection_testing>

|  |
| --- |
| Simple advice on Connection testing If you don't know what to do, try this:   1. If you know your driver supports the JDBC 4 Connection.isValid(...) method and you are using c3p0-0.9.5 or above, don't set a preferredTestQuery. If your driver does not support this method (or if you are not sure), try **SELECT 1** for your preferredTestQuery, if you are running MySQL or Postgres. For other databases, look for [suggestions here](http://stackoverflow.com/questions/3668506/efficient-sql-test-query-or-validation-query-that-will-work-across-all-or-most). Leave automatedTestTable undefined. 2. Begin by setting testConnectionOnCheckout to true and get your application to run correctly and stably. If you are happy with your application's performance, *you can stop here!* This is the simplest, most reliable form of Connection-testing, but it does have a client-visible performance cost. 3. If you'd like to improve performance by eliminating Connection testing from clients' code path:    1. Set testConnectionOnCheckout to false    2. Set testConnectionOnCheckin to true    3. Set idleConnectionTestPeriod to 30, fire up you application and observe. This is a pretty robust setting, all Connections will tested on check-in and every 30 seconds thereafter while in the pool. Your application should experience broken or stale Connections only very rarely, and the pool should recover from a database shutdown and restart quickly. But there is some overhead associated with all that Connection testing.    4. If database restarts will be rare so quick recovery is not an issue, consider reducing the frequency of tests by idleConnectionTestPeriod to, say, 300, and see whether clients are troubled by stale or broken Connections. If not, stick with 300, or try an even bigger number. Consider setting testConnectionOnCheckin back to false to avoid unnecessary tests on checkin. Alternatively, if your application does encounter bad Connections, consider reducing idleConnectionTestPeriod and set testConnectionOnCheckin back to true. There are no correct or incorrect values for these parameters: you are trading off overhead for reliability in deciding how frequently to test. The exact numbers are not so critical. It's usually easy to find configurations that perform well. It's rarely worth spending time in pursuit of "optimal" values here.   So, when should you stick with simple and reliable (Step 2 above), and when is it worth going for better performance (Step 3)? In general, it depends on how much work clients typically do with Connections once they check them out. If clients usually make complex queries and/or perform multiple operations, adding the extra cost of one fast test per checkout will not much affect performance. But if your application typically checks out a Connection and performs one simple query with it, throwing in an additional test can really slow things down.  That's nice in theory, but often people don't really have a good sense of how much work clients perform on average. The best thing to do is usually to try Step 3, see if it helps (however you measure performance), see if it hurts (is your application troubled by broken Connections? does it recover from database restarts well enough?), and then decide. You can always go back to simple, slow, and robust. Just settestConnectionOnCheckout to true, testConnectionOnCheckin to false, and set idleConnectionTestPeriod to 0.  *But do, always, be sure that your tests themselves are performant, either because your JDBC driver supports*Connection.isValid(...)*or because you have set an efficient* preferredTestQuery *!!!* |

* + 1. **索引**

新增索引：

CREATE INDEX IDX\_PBC\_AWC\_RPTSN ON CREDITINFO\_PBC\_AWARDCREDITINFO(REPORTSN);



# 第5次调优

日期： 2016-05-16

## 分析

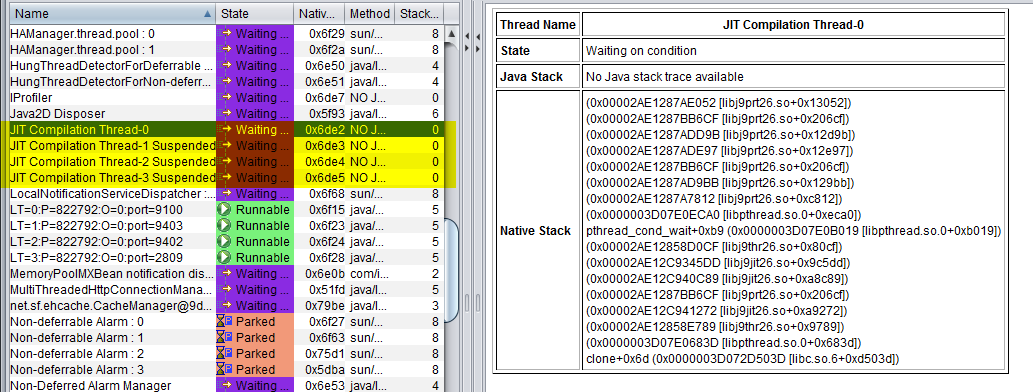
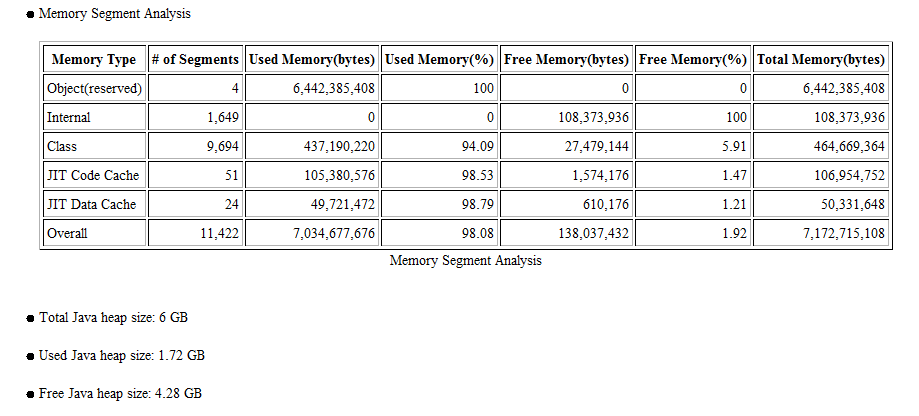
当JVM对代码进行编译后，被编译的代码以汇编指令的形式存在于代码缓存中(Code Cache)，显然这个缓存区域也是有大小限制的，当此区域被填满了之后，编译器就不能够再编译其他[Java](http://lib.csdn.net/base/17)字节码了。

所以当此区域设置的太小时，会对程序性能造成影响，因为编译器不会对Java字节码进行编译来得到运行速度更快的汇编指令/二进制代码了。

这个区域的默认空间经常不够，所以在必要的场景下需要增加它的空间。然而，并没有一个非常好的方法来给出一个应用到底在Code Cache区域的空间为多少时才能够达到最好的性能。你所能做的就是不断地进行尝试，来得到一个最好的结果。

**Code Cache的最大空间可以通过：-XX:ReservedCodeCacheSize=N来进行设置**。在默认情况下，N就是上表中的默认空间大小。关于Code Cache的空间管理，和JVM中对于其他内存空间的管理方法类似，也提供了一个设置初始值的方法：-XX:InitialCodeCacheSize=N。初始值和选择编译器的类型以及处理器的[架构](http://lib.csdn.net/base/16)相关，但是**一般需要设置的只是最大空间**。

从下图可以了解到JIT内存使用率也接近100%，出现JIT代码编译处于wait



## 解决办法：

添加启动参数 **-XX:ReservedCodeCacheSize=256m**