深入理解JDBC的超时设置  
C:\Users\admin\AppData\Roaming\Tencent\QQEIM\Temp\%W@GJ$ACOF(TYDYECOKVDYB.pnghttp://www.importnew.com/2466.html（译文）  
C:\Users\admin\AppData\Roaming\Tencent\QQEIM\Temp\%W@GJ$ACOF(TYDYECOKVDYB.pnghttps://www.cubrid.org/blog/understanding-jdbc-internals-and-timeout-configuration（原文）

# 深入理解JDBC的超时设置

2013/01/20 | 分类： [基础技术](http://www.importnew.com/cat/basic) | [5 条评论](http://www.importnew.com/2466.html#comments) | 标签： [JDBC](http://www.importnew.com/tag/jdbc)

分享到：**84**

本文作者： [ImportNew](http://www.importnew.com/) - [刘翀](http://www.importnew.com/author/gemoji) 未经许可，禁止转载！

ImportNew注： 本文由新浪微博：[@哥墨迹](http://www.weibo.com/lc87624) 投稿至ImportNew。感谢 [@哥墨迹](http://www.weibo.com/lc87624) ! 如果你希望分享好的原创文章或者译文，欢迎[投稿到ImportNew](http://www.importnew.com/guest-post" \t "_blank)。

恰当的JDBC超时设置能够有效地减少服务失效的时间。本文将对数据库的各种超时设置及其设置方法做介绍。

**真实案例：应用服务器在遭到DDos攻击后无法响应**

在遭到DDos攻击后，整个服务都垮掉了。由于第四层交换机不堪重负，网络变得无法连接，从而导致业务系统也无法正常运转。安全组很快屏蔽了所有的DDos攻击，并恢复了网络，但业务系统却还是无法工作。 通过分析系统的thread dump发现，业务系统停在了JDBC API的调用上。20分钟后，系统仍处于WAITING状态，无法响应。30分钟后，系统抛出异常，服务恢复正常。

为什么我们明明将query timeout设置成了3秒，系统却持续了30分钟的WAITING状态？为什么30分钟后系统又恢复正常了？ 当你对理解了JDBC的超时设置后，就能找到问题的答案。

**为什么我们要了解JDBC？**   
当遇到性能问题或系统出错时，业务系统和数据库通常是我们最关心的两个部分。在公司里，这两个部分是交由两个不同的部门来负责的，因此各个部门都会集中精力地在自身领域内寻找问题，这样的话，在业务系统和数据库之间的部分就会成为一个盲区。对于Java应用而言，这个盲区就是DBCP数据库连接池和JDBC，本文将集中介绍JDBC。

**什么是JDBC？**

JDBC是Java应用中用来连接关系型数据库的标准API。Sun公司一共定义了[4种类型的JDBC](http://en.wikipedia.org/wiki/JDBC_driver" \t "_blank)，我们主要使用的是第4种，该类型的Driver完全由Java代码实现，通过使用socket与数据库进行通信。

图1 JDBC Type 4.

第4种类型的JDBC通过socket对字节流进行处理，因此也会有一些基本网络操作，类似于HttpClient这种用于网络操作的代码库。当在网络操作中遇到问题的时候，将会消耗大量的cpu资源，并且失去响应超时。如果你之前用过HttpClient，那么你一定遇到过未设置timeout造成的错误。同样，第4种类型的JDBC，若没有合理地设置socket timeout，也会有相同的错误——连接被阻塞。   
接下来，就让我们来学习一下如何正确地设置socket timeout，以及需要考虑的问题。

应用与数据库间的timeout层级

图2 Timeout Class.

上图展示了简化后应用与数据库间的timeout层级。（译者注：WAS/BLOC是作者公司的具体应用名称，无需深究）   
高级别的timeout依赖于低级别的timeout，只有当低级别的timeout无误时，高级别的timeout才能确保正常。例如，当socket timeout出现问题时，高级别的statement timeout和transaction timeout都将失效。   
我们收到的很多评论中提到：

**引用**

即使设置了statement timeout，当网络出错时，应用也无法从错误中恢复。

statement timeout无法处理网络连接失败时的超时，它能做的仅仅是限制statement的操作时间。网络连接失败时的timeout必须交由JDBC来处理。   
JDBC的socket timeout会受到操作系统socket timeout设置的影响，这就解释了为什么在之前的案例中，JDBC连接会在网络出错后阻塞30分钟，然后又奇迹般恢复，即使我们并没有对JDBC的socket timeout进行设置。

DBCP连接池位于图2的左侧，你会发现timeout层级与DBCP是相互独立的。DBCP负责的是数据库连接的创建和管理，并不干涉timeout的处理。当连接在DBCP中创建，或是DBCP发送校验query检查连接有效性的时候，socket timeout将会影响这些过程，但并不直接对应用造成影响。   
当在应用中调用DBCP的getConnection()方法时，你可以设置获取数据库连接的超时时间，但是这和JDBC的timeout毫不相关。

图3 Timeout for Each Levels.

**什么是Transaction Timeout？**

transaction timeout一般存在于框架（Spring, EJB）或应用级。transaction timeout或许是个相对陌生的概念，简单地说，transaction timeout就是“statement Timeout \* N（需要执行的statement数量） + @（垃圾回收等其他时间）”。transaction timeout用来限制执行statement的总时长。   
例如，假设执行一个statement需要0.1秒，那么执行少量statement不会有什么问题，但若是要执行100,000个statement则需要10,000秒（约7个小时）。这时，transaction timeout就派上用场了。EJB CMT (Container Managed Transaction)就是一种典型的实现，它提供了多种方法供开发者选择。但我们并不使用EJB，Spring的transaction timeout设置会更常用一些。在Spring中，你可以使用下面展示的XML或是在源码中使用@Transactional注解来进行设置。

**Xml代码**

1. **<tx:attributes>**
2. **<tx:method** name=“…” timeout=“3″**/>**
3. **</tx:attributes>**

Spring提供的transaction timeout配置非常简单，它会记录每个事务的开始时间和消耗时间，当特定的事件发生时就会对消耗时间做校验，当超出timeout值时将抛出异常。   
Spring中，数据库连接被保存在ThreadLocal里，这被称为事务同步（Transaction Synchronization），与此同时，事务的开始时间和消耗时间也被保存下来。当使用这种代理连接创建statement时，就会校验事务的消耗时间。EJB CMT的实现方式与之类似，其结构本身也十分简单。   
当你选用的容器或框架并不支持transaction timeout这一特性，你可以考虑自己来实现。transaction timeout并没有标准的API。Lucy框架的1.5和1.6版本都不支持transaction timeout，但是你可以通过使用Spring的Transaction Manager来达到与之同样的效果。   
假设某个事务中包含5个statement，每个statement的执行时间是200ms，其他业务逻辑的执行时间是100ms，那么transaction timeout至少应该设置为1,100ms（200 \* 5 + 100）。

**什么是Statement Timeout？**   
statement timeout用来限制statement的执行时长，timeout的值通过调用JDBC的java.sql.Statement.setQueryTimeout(int timeout) API进行设置。不过现在开发者已经很少直接在代码中设置，而多是通过框架来进行设置。   
以iBatis为例，statement timeout的默认值可以通过sql-map-config.xml中的defaultStatementTimeout 属性进行设置。同时，你还可以设置sqlmap中select，insert，update标签的timeout属性，从而对不同sql语句的超时时间进行独立的配置。   
如果你使用的是Lucy1.5或1.6版本，通过设置queryTimeout属性可以在datasource层面对statement timeout进行设置。   
statement timeout的具体值需要依据应用本身的特性而定，并没有可供推荐的配置。

**JDBC的statement timeout处理过程**   
不同的关系型数据库，以及不同的JDBC驱动，其statement timeout处理过程会有所不同。其中，Oracle和MS SQLServer的处理相类似，MySQL和[CUBRID](http://www.cubrid.org/" \t "_blank)类似。

**Oracle JDBC Statement的QueryTimeout处理过程**   
1. 通过调用Connection的createStatement()方法创建statement   
2. 调用Statement的executeQuery()方法   
3. statement通过自身connection将query发送给Oracle数据库   
4. statement在OracleTimeoutPollingThread（每个classloader一个）上进行注册   
5. 达到超时时间   
6. OracleTimeoutPollingThread调用OracleStatement的cancel()方法   
7. 通过connection向正在执行的query发送cancel消息

图4 Query Timeout Execution Process for Oracle JDBC Statement.

**JTDS (MS SQLServer) Statement的QueryTimeout处理过程**   
1. 通过调用Connection的createStatement()方法创建statement   
2. 调用Statement的executeQuery()方法   
3. statement通过自身connection将query发送给MS SqlServer数据库   
4. statement在TimerThread上进行注册   
5. 达到超时时间   
6. TimerThread调用JtdsStatement实例中的TsdCore.cancel()方法   
7. 通过ConnectionJDBC向正在执行的query发送cancel消息

图5 QueryTimeout Execution Process for JTDS (MS SQLServer) Statement.

**MySQL JDBC Statement的QueryTimeout处理过程**   
1. 通过调用Connection的createStatement()方法创建statement   
2. 调用Statement的executeQuery()方法   
3. statement通过自身connection将query发送给MySQL数据库   
4. statement创建一个新的timeout-execution线程用于超时处理   
5. 5.1版本后改为每个connection分配一个timeout-execution线程   
6. 向timeout-execution线程进行注册   
7. 达到超时时间   
6. TimerThread调用JtdsStatement实例中的TsdCore.cancel()方法   
7. timeout-execution线程创建一个和statement配置相同的connection   
8. 使用新创建的connection向超时query发送cancel query（KILL QUERY “connectionId”）

图6 QueryTimeout Execution Process for MySQL JDBC Statement (5.0.8).

**CUBRID JDBC Statement的QueryTimeout处理过程**   
1. 通过调用Connection的createStatement()方法创建statement   
2. 调用Statement的executeQuery()方法   
3. statement通过自身connection将query发送给CUBRID数据库   
4. statement创建一个新的timeout-execution线程用于超时处理   
5. 5.1版本后改为每个connection分配一个timeout-execution线程   
6. 向timeout-execution线程进行注册   
7. 达到超时时间   
6. TimerThread调用JtdsStatement实例中的TsdCore.cancel()方法   
7. timeout-execution线程创建一个和statement配置相同的connection   
8. 使用新创建的connection向超时query发送cancel消息

图7 QueryTimeout Execution Process for CUBRID JDBC Statement.

**什么是JDBC的socket timeout？**   
第4种类型的JDBC使用socket与数据库连接，数据库并不对应用与数据库间的连接超时进行处理。   
JDBC的socket timeout在数据库被突然停掉或是发生网络错误（由于设备故障等原因）时十分重要。由于TCP/IP的结构原因，socket没有办法探测到网络错误，因此应用也无法主动发现数据库连接断开。如果没有设置socket timeout的话，应用在数据库返回结果前会无期限地等下去，这种连接被称为dead connection。   
为了避免dead connections，socket必须要有超时配置。socket timeout可以通过JDBC设置，socket timeout能够避免应用在发生网络错误时产生无休止等待的情况，缩短服务失效的时间。

不推荐使用socket timeout来限制statement的执行时长，因此socket timeout的值必须要高于statement timeout，否则，socket timeout将会先生效，这样statement timeout就变得毫无意义，也无法生效。

下面展示了socket timeout的两个设置项，不同的JDBC驱动其配置方式会有所不同。

* socket连接时的timeout：通过Socket.connect(SocketAddress endpoint, int timeout)设置
* socket读写时的timeout：通过Socket.setSoTimeout(int timeout)设置

通过查看CUBRID，MySQL，MS SQL Server (JTDS)和Oracle的JDBC驱动源码，我们发现所有的驱动内部都是使用上面的2个API来设置socket timeout的。

下面是不同驱动的socket timeout配置方式。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| JDBC Driver | connectTimeout配置项 | socketTimeout配置项 | url格式 | 示例 |
| MySQL Driver | connectTimeout（默认值：0，单位：ms） | socketTimeout（默认值：0，单位：ms） | jdbc:mysql://[host:port],[host:port]…/[database][?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]… | jdbc:mysql://xxx.xx.xxx.xxx:3306/database?connectTimeout=60000&socketTimeout=60000 |
| MS-SQL DriverjTDS Driver | loginTimeout（默认值：0，单位：s） | socketTimeout（默认值：0，单位：s） | jdbc:jtds:<server\_type>://<server>[:<port>][/<database>][;<property>=<value>[;...]] | jdbc:jtds:sqlserver://server:port/database;loginTimeout=60;socketTimeout=60 |
| Oracle Thin Driver | oracle.net.CONNECT\_TIMEOUT （默认值：0，单位：ms） | oracle.jdbc.ReadTimeout（默认值：0，单位：ms） | 不支持通过url配置，只能通过OracleDatasource.setConnectionProperties() API设置，使用DBCP时可以调用BasicDatasource.setConnectionProperties()或BasicDatasource.addConnectionProperties()进行设置 |  |
| CUBRID Thin Driver | 无独立配置项（默认值：5,000，单位：ms） | 无独立配置项（默认值：5,000，单位：ms） |  |  |

* connectTimeout和socketTimeout的默认值为0时，timeout不生效。
* 除了调用DBCP的API以外，还可以通过properties属性进行配置。

通过properties属性进行配置时，需要传入key为“connectionProperties”的键值对，value的格式为“[propertyName=property;]\*”。下面是iBatis中的properties配置。

**Xml代码**

1. **<transactionManager** type=“JDBC”**>**
2. **<dataSource** type=“com.nhncorp.lucy.db.DbcpDSFactory”**>**
3. ….
4. **<property** name=“connectionProperties” value=“oracle.net.CONNECT\_TIMEOUT=6000;oracle.jdbc.ReadTimeout=6000″**/>**
5. **</dataSource>**
6. **</transactionManager>**

**操作系统的socket timeout配置**   
如果不设置socket timeout或connect timeout，应用多数情况下是无法发现网络错误的。因此，当网络错误发生后，在连接重新连接成功或成功接收到数据之前，应用会无限制地等下去。但是，通过本文开篇处的实际案例我们发现，30分钟后应用的连接问题奇迹般的解决了，这是因为操作系统同样能够对socket timeout进行配置。公司的Linux服务器将socket timeout设置为了30分钟，从而会在操作系统的层面对网络连接做校验，因此即使JDBC的socket timeout设置为0，由网络错误造成的数据库连接问题的持续时间也不会超过30分钟。

通常，应用会在调用Socket.read()时由于网络问题被阻塞住，而很少在调用Socket.write()时进入waiting状态，这取决于网络构成和错误类型。当Socket.write()被调用时，数据被写入到操作系统内核的缓冲区，控制权立即回到应用手上。因此，一旦数据被写入内核缓冲区，Socket.write()调用就必然会成功。但是，如果系统内核缓冲区由于某种网络错误而满了的话，Socket.write()也会进入waiting状态。这种情况下，操作系统会尝试重新发包，当达到重试的时间限制时，将产生系统错误。在我们公司，重新发包的超时时间被设置为15分钟。

至此，我已经对JDBC的内部操作做了讲解，希望能够让大家学会如何正确的配置超时时间，从而减少错误的发生。

最后，我将列出一些常见的问题。

**FAQ**   
Q1. 我已经使用Statement.setQueryTimeout()方法设置了查询超时，但在网络出错时并没有产生作用。

➔ 查询超时仅在socket timeout生效的前提下才有效，它并不能用来解决外部的网络错误，要解决这种问题，必须设置JDBC的socket timeout。

Q2. transaction timeout，statement timeout和socket timeout和DBCP的配置有什么关系？

➔ 当通过DBCP获取数据库连接时，除了DBCP获取连接时的waitTimeout配置以外，其他配置对JDBC没有什么影响。

Q3. 如果设置了JDBC的socket timeout，那DBCP连接池中处于IDLE状态的连接是否也会在达到超时时间后被关闭？

➔ 不会。socket的设置只会在产生数据读写时生效，而不会对DBCP中的IDLE连接产生影响。当DBCP中发生新连接创建，老的IDLE连接被移除，或是连接有效性校验的时候，socket设置会对其产生一定的影响，但除非发生网络问题，否则影响很小。

Q4. socket timeout应该设置为多少？

➔ 就像我在正文中提的那样，socket timeout必须高于statement timeout，但并没有什么推荐值。在发生网络错误的时候，socket timeout将会生效，但是再小心的配置也无法避免网络错误的发生，只是在网络错误发生后缩短服务失效的时间（如果网络恢复正常的话）。

英文原文：[Cubrid](http://www.cubrid.org/blog/dev-platform/understanding-jdbc-internals-and-timeout-configuration/)，翻译：[@哥墨迹](http://www.weibo.com/lc87624" \t "_blank)

译文链接：<http://www.importnew.com/2466.html>

【如需转载，请联系译者 [@哥墨迹](http://www.weibo.com/lc87624)！】

关于作者： [刘翀](http://www.importnew.com/author/gemoji)

**WHAT WE WRITE**

Understanding JDBC Internals & Timeout Configuration

**WRITTEN BY Woon Duk Kang ON 06/07/2017**

An application with a proper JDBC timeout can cut down the failure time. In this article we would like to talk about different kinds of timeout values and recommended timeout application methods when you import values from DBMS.

Web Application Server became unresponsive after a DDos attack one day

*(This is a close reconstitution of an actual event.)*

The entire service did not work normally after a DDos attack. The network was disconnected because L4 was not working, which caused WAS to be inoperable as well. Shortly afterwards, the security team blocked all DDos attacks, and restored the network back to normal. Yet, WAS was still not working.

Through the ThreadDump of WAS, the service team was able to confirm that WAS had stopped during API call from JDBC. After 20 minutes, WAS was still in WAITING status and the service was still not working. About 30 minutes had passed when an exception suddenly occurred, and the service was restored.

***Why was WAS in WAITING status for 30 minutes when QueryTimeout value was set to 3 seconds, and why did WAS start working again after 30 minutes?***

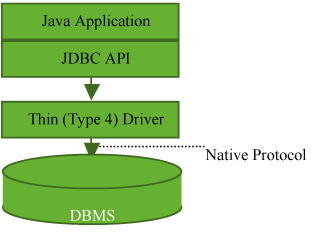
You can find the answer if you understand how the JDBC Timeout works.

Why Do We Need to Know about the JDBC Driver?

When there is a performance issue or an error, WAS and DBMS are the two important tiers we pay attention to. In NHN, WAS and DBMS are generally handled by different departments, so each department tries to figure out this situation by focusing on their own area of expertise. When this happens, you get a blind spot between WAS and DBMS, that does not receive much attention. For Java applications, the blind spot would be between DBCP and JDBC. In this article we will focus on JDBC.

What is a JDBC Driver?

JDBC is a standard API that you use to access the DBMS in Java applications. There are [**4 types of JDBC drivers**](http://en.wikipedia.org/wiki/JDBC_driver)(Wikipedia) defined by Sun. NHN mainly uses the *type 4*. JDBC type 4 driver is written entirely in Java (pure Java) and communicates with a DBMS using sockets in Java applications.

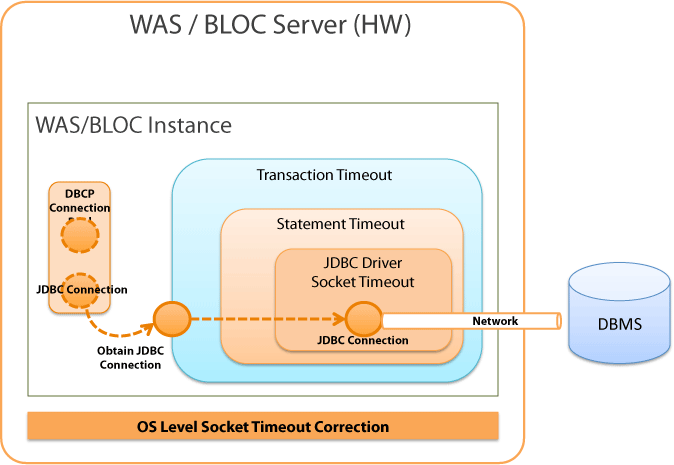


**Figure 1: JDBC Type 4.**

Type 4 drivers process byte stream via sockets, and have the same basic operations as a network library like *HttpClient*. This uses up a lot of CPU resources and loses response timeout, while sharing the same error points with other network libraries. If you have used HttpClient before, then you must have encountered errors from not setting the timeout value. Type 4 driver may have the same error (a hang occurs) if the socket timeout value is not set properly.

Let's learn about how to configure the socket timeout value for JDBC driver, and what needs to be considered.

Timeout Class at WAS - DBMS Communication



**Figure 2: Timeout Class.**

Figure 2 above shows a simplified version of the timeout class when WAS and DBMS are communicating.

The higher level timeout is dependent on the lower level timeout. The higher level timeout will operate normally only if the lower level timeout operates normally as well. If the JDBC driver socket timeout does not work properly, then higher level timeouts such as statement timeout and transaction timeout will not work properly either.

We have received a lot of comments that said:

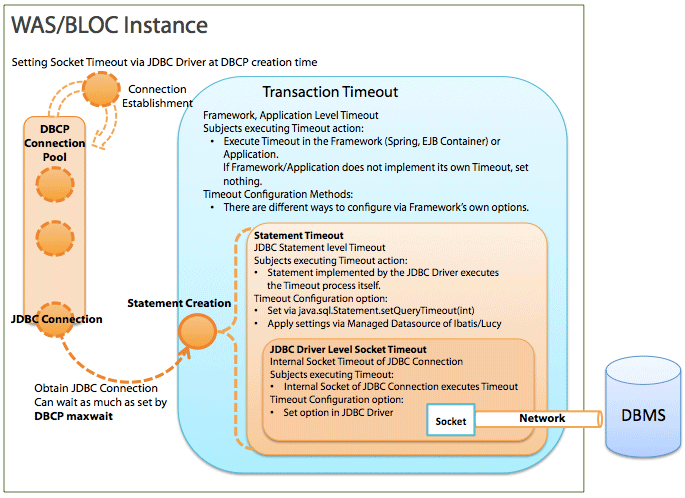
*Even after the statement timeout was configured, the application still did not recover from the error because the statement timeout did not work at the time of network failure.*

**The statement timeout does not handle the timeouts at the time of network failure.** Statement timeout does only one thing: *restricts the operation time of 1 statement*. Handling timeout to prevent network failure must be done by JDBC Driver.

The JDBC driver's socket timeout is affected by the OS's socket timeout configuration. This would explain why JDBC connection hang recovers 30 minutes after the network connection failure, even when the JDBC driver's socket timeout is not configured.

DBCP Connection Pool is located on the left side of Figure 2. You can see that the timeout classes and DBCP are separated. DBCP is in charge of creating and managing connections, and is not involved in processing timeouts. When a connection is created within DBCP or a validation query is sent to check the validity of the connection, the socket timeout does affect these processes but does not affect the application directly.

However, when **getConnection()** is called to DBCP from the application logic, then you can specify the timeout until the application acquires the connection. However, this has nothing to do with the JDBC's connect timeout.



**Figure 3: Timeout for Each Levels.**

What is Transaction Timeout?

**Transaction timeout** is a timeout valid in *frameworks* (Spring, EJB container) or at the *application level*.

Transaction timeout can be an unfamiliar concept. Simply put, transaction timeout is  
**Statement Timeout \* N (number of statements being processed) + @ (garbage collection, etc.).**  
Transaction timeout is used to limit the total statement processing time to the maximum amount allowed.

For example, if it takes 0.1 second to process 1 statement, processing a few statements would not be a problem, but processing 100,000 statements would take 10,000 seconds (approx. 7 hours). Statement timeout can be used here.

EJB CMT (Container Managed Transaction) would be a typical example of actual implementations. EJB CMT varies in its implementation methods and operating process depending on developers. NHN does not use EJB Container, so transaction timeout of Spring Framework would be the most common example. In Spring, you may use XML as shown below or use **@Transactional** from Java source codes, for configuration.

**<tx:attributes>**

**<tx:method name="…" timeout="3"/>**

**</tx:attributes>**

Statement timeout provided by Spring is very simple. It records the starting time and the elapsed time for each transaction, and checks the elapsed time when an event occurs. If the timeout is abnormal, it generates an exception.

In Spring, the connection is stored in, and used from **ThreadLocal**. This is called **Transaction Synchronization**. When a connection is saved in ThreadLocal, the starting time and the timeout time of the transaction is also recorded. When a statement is being created by using the proxy connection, the elapsed time is checked to generate an exception.

The EJB CMT implementation is done in a similar way. The structure itself is very simple. If the transaction timeout is very important but the container or the framework you are using does not provide this feature, you could implement it yourself without major problems. There is no standard API for transaction timeout.

Lucy 1.5 and 1.6 Framework does not have a transaction timeout feature, but you can get the same result by using Transaction Manager from Spring.

If the processing time of the statement (5 or less) is 200 ms and the processing time of other business logics or framework operation is 100 ms, the transaction timeout time should be set to 1,100 ms ((200 \* 5) + 100) or more.

What is Statement Timeout?

It is a *limitation on how long a statement should run*. It sets the timeout value for the statement, which is a JDBC API. The JDBC driver processes the statement timeout based on this value. Statement timeout is configured via **java.sql.Statement.setQueryTimeout(int timeout)**, which is a JDBC API. In recent developing environments, the developers rarely configure the statement timeout value directly through Java source codes, but often configure it by using the framework.

To use **iBatis** as an example, the default value can be configured by using **@defaultStatementTimeout** value in *sqlMapConfig/settings* of *sql-map-config.xml*. By using **@timeout** value, you can configure statement, select, insert and update syntax of *sql-map.xml* separately.

When **MangedDatasource** of Lucy 1.5 and 1.6 is used, the **queryTimeout** option can be used to get a statement of which timeout is configured at the datasource level.

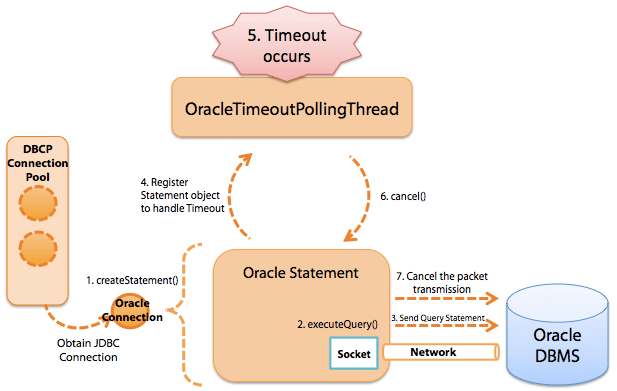
The statement timeout time is configured based on the features of each application, so there is no recommended configuration value.

Statement Timeout Execution Process for JDBC Driver

Statement timeout works differently per DBMS and driver. The way it works is similar between Oracle and MS SQLServer. It is also similar between MySQL and CUBRID.

QueryTimeout for Oracle JDBC Statement

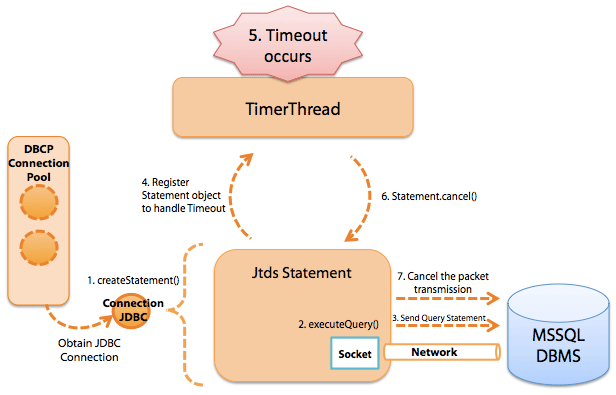
1. Creates a statement by calling **Connection.createStatement()**.
2. Calls **Statement.executeQuery()**.
3. The statement transmits the Query to Oracle DBMS by using its own connection.
4. The statement registers a statement to **OracleTimeoutPollingThread** (1 for each classloader) for timeout process.
5. Timeout occurs.
6. OracleTimeoutPollingThread calls **OracleStatement.cancel()**.
7. Sends a cancel message through the connection and cancels the query being executed.



**Figure 4: Query Timeout Execution Process for Oracle JDBC Statement.**

QueryTimeout for JTDS (MS SQLServer) Statement

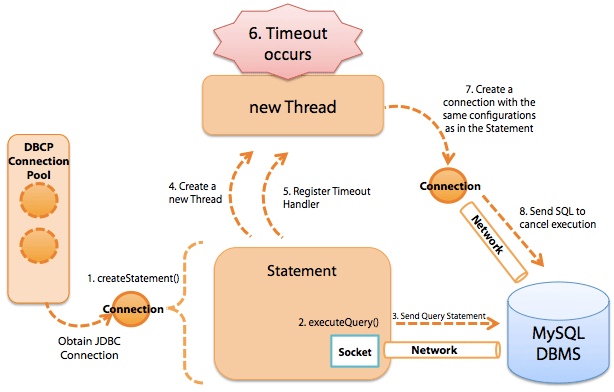
1. Creates a statement by calling **Connection.createStatement()**.
2. Calls **Statement.executeQuery()**.
3. The statement transmits the Query to MS SqlServer by using the internal connection.
4. The statement registers a statement in TimerThread for timeout process.
5. Timeout occurs.
6. TimerThread calls up **TsdCore.cancel()** inside the JtdsStatement object.
7. Sends a cancel message through the ConnectionJDBC and cancels the query being executed.



**Figure 5: QueryTimeout Execution Process for JTDS (MS SQLServer) Statement.**

QueryTimeout for MySQL JDBC Statement (5.0.8)

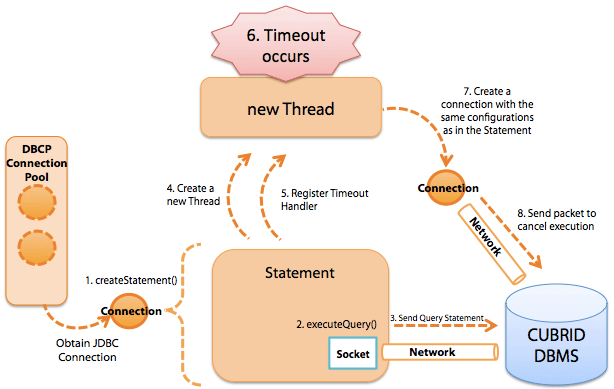
1. Creates a statement by calling **Connection.createStatement()**.
2. Calls **Statement.executeQuery()**.
3. The statement transmits the Query to MySqlServer by using the internal connection.
4. The statement creates a new timeout-execution thread for timeout process.
5. For version 5.1.x, it changes to assign 1 thread for each connection.
6. Registers the timeout execution to the thread.
7. Timeout occurs.
8. The timeout-execution thread creates a connection that has the same configurations as the statement.
9. Transmits the cancel Query (KILL QUERY "connectionId“) by using the connection.



**Figure 6: QueryTimeout Execution Process for MySQL JDBC Statement (5.0.8).**

QueryTimeout for CUBRID JDBC Statement

1. Creates a statement by calling **Connection.createStatement()**.
2. Calls **Statement.executeQuery()**.
3. The statement transmits the Query to CUBRID DBMS by using the internal connection.
4. The statement creates a new timeout-execution thread for timeout process.
5. Registers the timeout execution to the thread.
6. Timeout occurs.
7. The timeout-execution thread creates a connection that has the same configurations as the statement.
8. Transmits the cancel message using the connection.

****

**Figure 7: QueryTimeout Execution Process for CUBRID JDBC Statement.**

What is Socket Timeout for JDBC Driver?

JDBC driver type 4 uses the socket to connect to the DBMS, and the connection timeout process between the application and the DBMS is not carried out by the DBMS.

Socket timeout value for JDBC driver is necessary when the DBMS is terminated abruptly or an network error has occured (equipment malfunction, etc.). Because of the structure of TCP/IP, there are no means for the socket to detect network errors. Therefore, the application cannot detect any disconnection with the DBMS. If the socket timeout is not configured, then the application may wait for the results from the DBMS indefinitely. (This connection is also called a "**dead connection**.") To prevent dead connections, a timeout must be configured for the socket. Socket timeout can be configured via JDBC driver. By setting up the socket timeout, you can prevent the infinite waiting situation when there is a network error and shorten the failure time.

It is not recommended to use the socket timeout value to limit the statement execution time. So **the socket timeout value must be higher than the statement timeout value**. If the socket timeout value is smaller than the statement timeout value, as the socket timeout will be executed first, and the statement timeout value becomes meaningless and will not be executed.

Socket timeout has 2 options listed below, and their configurations vary by driver.

* Timeout at socket connection: Time limit for **Socket.connect(SocketAddress endpoint, int timeout)**
* Timeout at socket reading/writing: Time limit for **Socket.setSoTimeout(int timeout)**

By checking the source for CUBRID, MySQL, MS SQL Server (JTDS) and Oracle JDBC, we confirmed that all the drivers we checked use the 2 APIs above.

How to configure SocketTimeout is as explained below.

| **JDBC Driver** | **connectTimeout** | **Default** | **Unit** | **Application Method** |
| --- | --- | --- | --- | --- |
| **socketTimeout** | **Default** | **Unit** |
| MySQL Driver | connectTimeout | 0 | ms | Specify the option in the DriverURL. **Format:** **jdbc:mysql://[host:port],[host:port].../[database]** **[?propertyName1][=propertyValue1][&propertyName2][=propertyValue2]...**  **Example:**  **jdbc:mysql://xxx.xx.xxx.xxx:3306/database?connectTimeout=60000&socketTimeout=60000** |
| socketTimeout | 0 | ms |
| MS-SQL Driver jTDS Driver | loginTimeout | 0 | sec | Specify the option in the DriverURL. **Format:** **jdbc:jtds:<server\_type>://<server>[:<port>][/<database>][;<property>=<value>[;...]]**  **Example:**  **jdbc:jtds:sqlserver://server:port/database;loginTimeout=60;socketTimeout=60** |
| socketTimeout | 0 | sec |
| Oracle Thin Driver | oracle.net.CONNECT\_TIMEOUT | 0 | ms | Not possible with the driverURL. Must be delivered to the properties object via **OracleDatasource.setConnectionProperties()** API. When DBCP is used, use the following APIs:   * **BasicDatasource.setConnectionProperties()** * **BasicDatasource.addConnectionProperties()** |
| oracle.jdbc.ReadTimeout | 0 | ms |
| CUBRID Thin Driver | No separate configuration | 5,000 | ms | Not possible with the driverURL. Timeout occurs in 5 seconds.   * Note 1: When timeout occurs with althost option specified in the URL, it can be connected to the designated host. * Note 2: C API can be used to state the **login\_time** option in ms in the URL. |
|  |  |  |  |

* Note 1: The default value for **connectTimeout** and **socketTimeout** is "0," which means that the timeout does not occur.
* Note 2: You can also configure through properties without directly using the separate API of DBCP.

When you configure properties, pass on the character string where the key value is “connectionProperties”, and the format value is “[propertyName=property;]\*”. The following example shows configuring properties through xml in iBatis.

**<transactionManager type="JDBC">**

**<dataSource type="com.nhncorp.lucy.db.DbcpDSFactory">**

**....**

**<property name="connectionProperties" value="oracle.net.CONNECT\_TIMEOUT=6000;oracle.jdbc.ReadTimeout=6000"/>**

**</dataSource>**

**</transactionManager>**

OS Level SocketTimeout Configuration

If the socket timeout or the connect timeout is not configured, most of the time, applications cannot detect network errors. So, until the applications are connected or are able to read data, they will wait indefinitely. However, if you look at the actual issues NHN services encountered, the problems were often resolved after the applications (WAS) tried to reconnect to the network 30 minutes after. This is because the OS can also configure socket timeout time. Linux servers used by NHN have set the socket timeout to 30 minutes. This checks the network connection at the OS level. Because the KeepAlive checking cycle for NHN's Linux servers is 30 minutes, even when socket timeout is set to 0, the DBMS network connection problems caused by network issues do not surpass 30 minutes.

Generally, the application hangs from network issues when the application is calling **Socket.read()**. However, depending on the network composition or the error type, it can rarely be in waiting status while running **Socket.write()**. When the application calls **Socket.write()**, the data is recorded to the OS kernel buffer and then the right to control is returned to the application immediately. Thus, as long as a valid value is recorded to the kernel buffer, **Socket.write()** is always successful. However, if the OS kernel buffer is full due to a special network error, even **Socket.write()** can be put into waiting status. In this case, the OS tries to resend the packet for a certain amount of time, and generates an error when it reaches the limit. In NHN's Linux server environment, the timeout for this situation is set to 15 minutes.

I have explained the internal operations of JDBC so far. I hope that this will help you with the correct timeout configuration and reducing errors.

If you have more questions or any good information related to JDBC, please leave your comments below.

Lastly, I have listed some of the frequently asked questions below.

FAQ

***Q1. I configured the query timeout by using Statement.setQueryTimeout(), but it does not work as expected when there is a network error.***

*Query Timeout only works when it is connected to the socket correctly. Therefore, it cannot be used to solve an exceptional situation with a network error. To be prepared for network errors, socket timeout in JDBC driver must be configured.*

***Q2. How are transaction timeout, statement timeout and JDBC driver socket timeout related to the DBCP configuration values?***

*When the connection is acquired from DBCP to JDBC, nothing but****waitTimeout****is affected.*

***Q3. If JDBC SocketTimeout is configured, wouldn't the connections that stayed in idle status for a long time in DBCP be closed?***

*No. The socket option is applied when the actual data is being written or read, so it does not affect the connections in idle status in DBCP. The socket option can have certain effect when new connections that lack in inside of DBCP are created, old idle connections are removed, or the validation is checked, but this does not cause any significant issues unless the network has an error.*

***Q4. How long should SocketTimeout be set to?***

*As I have mentioned in the main article above, it must be much bigger than the statement timeout, and there is no recommended value. Socket timeout value for the JDBC driver becomes effective after a network error occurs. A careful configuration for the value cannot prevent such the errors from happening, but sometimes shortens the time that the network is disabled (if the network is restored right away).*