<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close">

<!-- 基本属性 url、user、password -->

<property name="url" value="${jdbc\_url}" />

<property name="username" value="${jdbc\_user}" />

<property name="password" value="${jdbc\_password}" />

<!-- 配置初始化大小、最小、最大 -->

<property name="initialSize" value="5" />

<property name="minIdle" value="10" />

<property name="maxActive" value="20" />

<!-- 配置获取连接等待超时的时间 -->

<property name="maxWait" value="60000" />

<!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->

<property name="timeBetweenEvictionRunsMillis" value="2000" />

<!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->

<property name="minEvictableIdleTimeMillis" value="600000" />

<property name="maxEvictableIdleTimeMillis" value="900000" />

<property name="validationQuery" value="select 1" />

<property name="testWhileIdle" value="true" />

<property name="testOnBorrow" value="false" />

<property name="testOnReturn" value="false" />

<property name="keepAlive" value="true" />

<property name="phyMaxUseCount" value="1000" />

<!-- 配置监控统计拦截的filters -->

<property name="filters" value="stat" />

</bean>

<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource"

init-method="init" destroy-method="close">

<!-- 基本属性 url、user、password -->

<property name="url" value="${jdbc\_url}" />

<property name="username" value="${jdbc\_user}" />

<property name="password" value="${jdbc\_password}" />

<!-- 配置初始化大小、最小、最大 -->

<property name="initialSize" value="5" />

<property name="minIdle" value="5" />

<property name="maxActive" value="10" />

<!-- 配置从连接池获取连接等待超时的时间 -->

<property name="maxWait" value="10000" />

<!-- 配置间隔多久启动一次DestroyThread，对连接池内的连接才进行一次检测，单位是毫秒。

检测时:1.如果连接空闲并且超过minIdle以外的连接，如果空闲时间超过minEvictableIdleTimeMillis设置的值则直接物理关闭。2.在minIdle以内的不处理。

-->

<property name="timeBetweenEvictionRunsMillis" value="600000" />

<!-- 配置一个连接在池中最大空闲时间，单位是毫秒 -->

<property name="minEvictableIdleTimeMillis" value="300000" />

<!-- 设置从连接池获取连接时是否检查连接有效性，true时，每次都检查;false时，不检查 -->

<property name="testOnBorrow" value="false" />

<!-- 设置往连接池归还连接时是否检查连接有效性，true时，每次都检查;false时，不检查 -->

<property name="testOnReturn" value="false" />

<!-- 设置从连接池获取连接时是否检查连接有效性，true时，如果连接空闲时间超过minEvictableIdleTimeMillis进行检查，否则不检查;false时，不检查 -->

<property name="testWhileIdle" value="true" />

<!-- 检验连接是否有效的查询语句。如果数据库Driver支持ping()方法，则优先使用ping()方法进行检查，否则使用validationQuery查询进行检查。(Oracle jdbc Driver目前不支持ping方法) -->

<property name="validationQuery" value="select 1 from dual" />

<!-- 单位：秒，检测连接是否有效的超时时间。底层调用jdbc Statement对象的void setQueryTimeout(int seconds)方法 -->

<!-- <property name="validationQueryTimeout" value="1" /> -->

<!-- 打开后，增强timeBetweenEvictionRunsMillis的周期性连接检查，minIdle内的空闲连接，每次检查强制验证连接有效性. 参考：https://github.com/alibaba/druid/wiki/KeepAlive\_cn -->

<property name="keepAlive" value="true" />

<!-- 连接泄露检查，打开removeAbandoned功能 , 连接从连接池借出后，长时间不归还，将触发强制回连接。回收周期随timeBetweenEvictionRunsMillis进行，如果连接为从连接池借出状态，并且未执行任何sql，并且从借出时间起已超过removeAbandonedTimeout时间，则强制归还连接到连接池中。 -->

<property name="removeAbandoned" value="true" />

<!-- 超时时间，秒 -->

<property name="removeAbandonedTimeout" value="80"/>

<!-- 关闭abanded连接时输出错误日志，这样出现连接泄露时可以通过错误日志定位忘记关闭连接的位置 -->

<property name="logAbandoned" value="true" />

<!-- 根据自身业务及事务大小来设置 -->

<property name="connectionProperties"

value="oracle.net.CONNECT\_TIMEOUT=2000;oracle.jdbc.ReadTimeout=10000"></property>

<!-- 打开PSCache，并且指定每个连接上PSCache的大小，Oracle等支持游标的数据库，打开此开关，会以数量级提升性能，具体查阅PSCache相关资料 -->

<property name="poolPreparedStatements" value="true" />

<property name="maxPoolPreparedStatementPerConnectionSize"

value="20" />

<!-- 配置监控统计拦截的filters -->

<!-- <property name="filters" value="stat,slf4j" /> -->

<property name="proxyFilters">

<list>

<ref bean="log-filter" />

<ref bean="stat-filter" />

</list>

</property>

<!-- 配置监控统计日志的输出间隔，单位毫秒，每次输出所有统计数据会重置，酌情开启 -->

<property name="timeBetweenLogStatsMillis" value="120000" />

</bean>

Spring boot autoconfig下能否优化下setMinEvictableIdleTimeMillis与setMaxEvictableIdleTimeMillis顺序问题

if (idleMillis < minEvictableIdleTimeMillis) {//此处是break无问题,空闲连接从0开始排,取从最后开始取

break;

}

if (checkTime && i < checkCount) {//checkTime为true,此处验证上方的最小连接数,释放到最小连接后不再释放

evictList.add(connection);

} else if (idleMillis > maxEvictableIdleTimeMillis) {//此处验证最大存活时间,无视最小连接数强制回收

evictList.add(connection);

————————————————

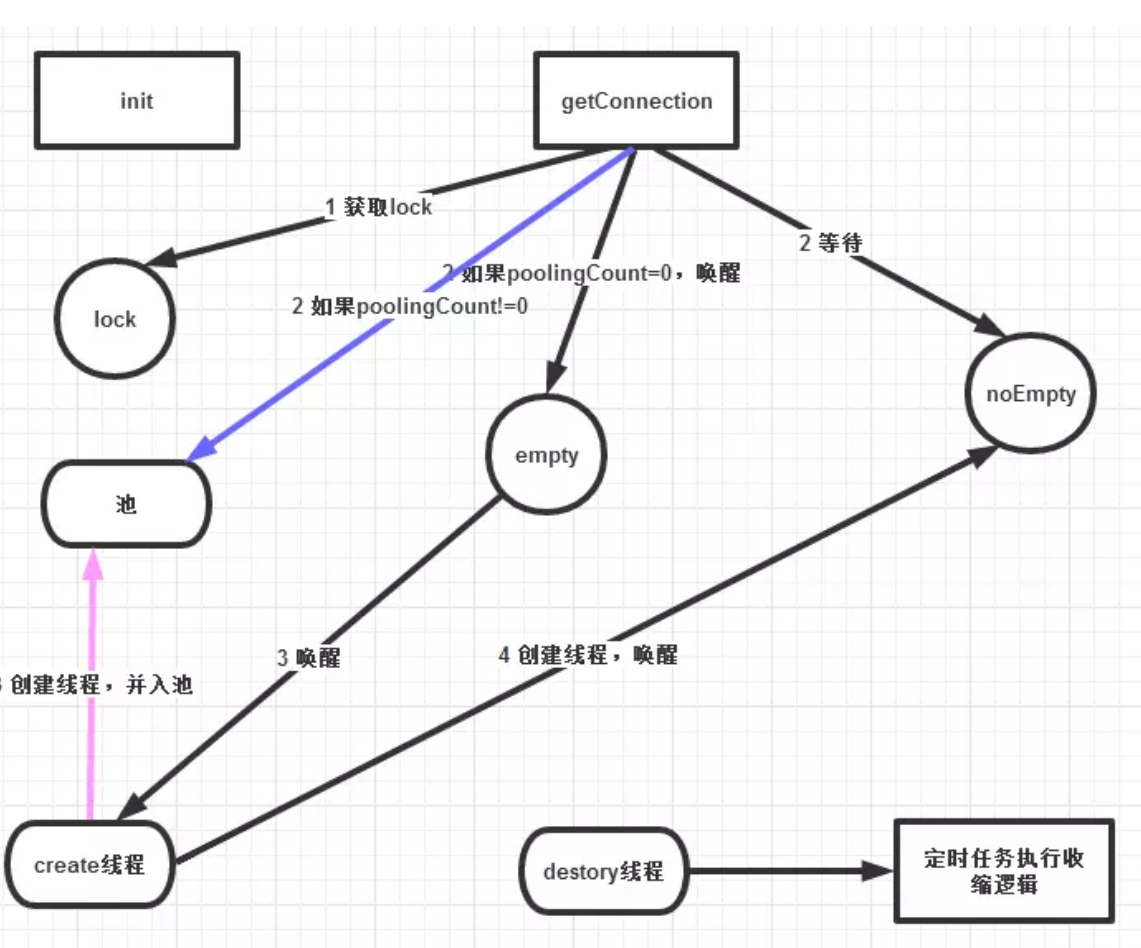
版权声明：本文为CSDN博主「一渣程序猿」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：<https://blog.csdn.net/leiyong0326/article/details/527>

|  |
| --- |
| if (idleMillis < minEvictableIdleTimeMillis |
|  | && idleMillis < keepAliveBetweenTimeMillis |
|  | ) { |
|  | break; |
|  | } |
|  |  |
|  | if (idleMillis >= minEvictableIdleTimeMillis) { |
|  | if (checkTime && i < checkCount) { |
|  | evictConnections[evictCount++] = connection; |
|  | continue; |
|  | } else if (idleMillis > maxEvictableIdleTimeMillis) { |
|  | evictConnections[evictCount++] = connection; |
|  | continue; |
|  | } |
|  | } |
|  |  |
|  | if (keepAlive && idleMillis >= keepAliveBetweenTimeMillis) { |
|  | keepAliveConnections[keepAliveCount++] = connection; |
|  | } |

80267

1. 初始化连接池时会填充到minIdle数量。
2. 连接池中的minIdle数量以内的连接，空闲时间超过minEvictableIdleTimeMillis，则会执行keepAlive操作。
3. 当网络断开等原因产生的由ExceptionSorter检测出来的死连接被清除后，自动补充连接到minIdle数量。



当网络断开或者数据库服务器Crash时，连接池里面会存在“不可用连接”，连接池需要一种机制剔除这些“不可用连接”。在Druid和JBoss连接池中，剔除“不可用连接”的机制称为ExceptionSorter，实现的原理是根据异常类型/Code/Reason/Message来识别“不可用连接”。

**每个连接都有一个缓存的原因是因为：JDBC驱动程序就是这样工作的。任何prepared statement都是由指定的连接所返回的。**

**显然，我不会在这里写出很多的细节，我们只关注最关键的部分。当一个数据库收到一个statement后，数据库引擎会先解析statement，然后检查其是否有语法错误。一旦statement被正确的解析，数据库会选出执行statement的最优途径。遗憾的是这个计算开销非常昂贵。数据库会首先检查是否有相关的索引可以对此提供帮助，不管是否会将一个表中的全部行都读出来。数据库对数据进行统计，然后选出最优途径。当决创建查询方案后，数据库引擎会将它执行。**  
  
**存取方案（Access Plan）的生成会占用相当多的CPU。理想的情况是，当我们多次发送一个statement到数据库，数据库应该对statement的存取方案进行重用。如果方案曾经被生成过的话，这将减少CPU的使用率。**  
   
   
Statement Caches  
   
**数据库已经具有了类似的功能。它们通常会用如下方法对statement进行缓存。使用statement本身作为key并将存取方案存入与statement对应的缓存中。这样数据库引擎就可以对曾经执行过的statements中的存取方案进行重用。举个例子，如果我们发送一条包含SELECT a, b FROM t WHERE c = 2的statement到数据库，然后首先会将存取方案进行缓存。当我们再次发送相同的statement时，数据库会对先前使用过的存取方案进行重用，这样就降低了CPU的开销。**  
   
**注意，这里使用了整个statement为key。也就是说，如果我们发送一个包含SELECT a, b FROM t WHERE c = 3的statement的话，缓存中会没有与之对应的存取方案。这是因为“c=3”与曾经被缓存过的“c=2”不同**

**Cache是连接不共用的，导致数据库重复存储**

Using a statement cache at the pooling layer is an [anti-pattern](https://en.wikipedia.org/wiki/Anti-pattern), and will negatively impact your application performance compared to driver-provided caches.

同时Druid不仅仅是一个数据库连接池，Druid 核心主要包括三部分：

* 基于Filter－Chain模式的插件体系。
* DruidDataSource 高效可管理的数据库连接池。
* SQLParser

Druid的主要功能如下:

* 是一个高效、功能强大、可扩展性好的数据库连接池。
* 可以监控数据库访问性能。
* 数据库密码加密
* 获得SQL执行日志
* 扩展JDBC

作者：芋道*源码*  
链接：https://zhuanlan.zhihu.com/p/63018846  
来源：知乎  
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

