什么是连接池

数据库连接池负责分配、管理和释放数据库连接，它允许应用程序重复使用一个现有的数据库连接，而不是再重新建立一个；

连接池是存储、管理数据库连接的容器，应用程序把获取数据库连接的功能委托给连接池，每个连接池都有一个上限，如果连接池达到上限，应用程序线程申请连接时被堵塞，等待其他线程释放连接，每个线程使用完连接后并不马上关闭，至少把它返还给连接池。由于连接的共享，不会频繁的创建、销毁连接，因此就不会增加创建连接的开销，也不会出现socket释放延迟现象。

<https://www.nowcoder.com/questionTerminal/4fd5701df25a47ff8c1ac7bf43ea5828?toCommentId=73847>

为什么要用连接池

获取一个连接，系统要在背后做很多消耗资源的事情，大多时候，创建连接的时间比执行sql语句的时间还要长。用户每次请求都需要向数据库获得链接，而数据库创建连接通常需要消耗相对较大的资源，创建时间也较长。

不用连接池如何建立数据库连接：

**JDBCUtil.java**

1. //获取一个数据库连接
2. **public** **static** Connection getConnection() {
3. **try** {
4. //1.加载驱动
5. Class.forName("com.mysql.jdbc.Driver");
6. //2.注册驱动
7. DriverManager.registerDriver(**new** com.mysql.jdbc.Driver());
8. //3.获取连接
9. String dbUrl = "jdbc:mysql://127.0.0.1:3306/xudatabase";
10. conn = DriverManager.getConnection(dbUrl, "root", "123456");
11. System.out.println("========数据库连接成功========");
12. } **catch** (Exception e) {
13. e.printStackTrace();
14. System.out.println("========数据库连接失败========");
15. **return** **null**;
16. }
17. **return** conn;
18. }
19. /\*\*
20. \* 操作数据库
21. \* @param data
22. \*/
23. **public** **void** writeDBByEveryConnNoPool(**int** data){
24. Connection conn =  getConnection();
25. String sql = "insert into n2 (age) values (" + data + ")";
26. **try**{
27. Statement stat = conn.createStatement();
28. stat.executeUpdate(sql);
29. }**catch**(Exception e){
30. e.printStackTrace() ;
31. }**finally**{
32. **try** {
33. conn.close();
34. } **catch** (SQLException e) {
35. e.printStackTrace();
36. }
38. }
39. }

性能对比：

1. @Test
2. **public** **void** useDbPool() **throws** SQLException {
3. **long** start = System.currentTimeMillis();
4. System.out.println(start);
5. **for**(**int** i=0;i<10000;i++){
6. Connection connection = dataSource.getConnection();
7. PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO TEST (ID)VALUES(?)");
8. preparedStatement.setInt(1,i);
9. preparedStatement.execute();
10. connection.close();
11. }
12. //耗时：11384
13. System.out.println(System.currentTimeMillis()-start);
14. }
16. @Test
17. **public** **void** noDbPool() **throws** SQLException {
18. **long** start = System.currentTimeMillis();
19. System.out.println(start);
20. **for**(**int** i=0;i<10000;i++){
21. Connection connection = JDBCUtil.getConnection();
22. PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO TEST (ID)VALUES(?)");
23. preparedStatement.setInt(1,i);
24. preparedStatement.execute();
25. connection.close();
26. }
27. //耗时59077
28. System.out.println(System.currentTimeMillis()-start);
29. }

几种常用的连接池配置介绍

DBCP连接池：

简介：

Apache的连接池，使用DBCP必需的两个JAR包commons-dbcp和commons.pool,如果使用的maven则只引入commons-dbcp的依赖即可，dbcp的pom文件内部已经引入了对pool的依赖



版本对应关系

|  |  |  |
| --- | --- | --- |
| JDK版本 | JDBC版本 | DBCP版本 |
| 1.1 | 1.0 | --- |
| 1.2 | 2.0 |
| 1.3 |
| 1.4 | 3.0 | DBCP 1.3 |
| 1.5 |
| 1.6 | 4.0 | DBCP 1.4 |
| 1.7 | 4.1 | DBCP 2 |
| 1.8 | 4.2 |

JDBC随JDK发布

相关:

JDBC:

是Java语言中用来规范客户端程序如何来访问数据库的应用程序接口，提供了诸如查询和更新数据库中数据的方法。JDBC也是Sun Microsystems的商标。JDBC是面向关系型数据库的。

https://zh.wikipedia.org/wiki/Java%E6%95%B0%E6%8D%AE%E5%BA%93%E8%BF%9E%E6%8E%A5

基本配置：

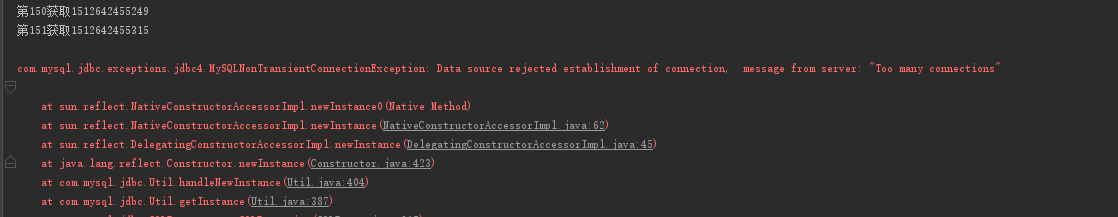
|  |  |
| --- | --- |
| 属性名 | 说明 |
| driverClassName | 驱动类名 |
| url | URL |
| password | 密码 |
| username | 用户名 |

关键配置：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 默认 | 说明 | 备注 |
| defaultReadOnly | null | **连接池初始化**(调用这些方法的其中之一后getConnection, setLogwriter, setLoginTimeout, getLoginTimeout, getLogWriter)后再设置该属性无效 | 连接池在调用这些方法时才会完成初始化过程getConnection …… |
| initialSize | 0 | 连接池启动时创建的初始化连接数量（默认值为0） |  |
| maxTotal | 8 | 当连接池中没有空闲的连接，又有线程需要去访问数据库时，连接池此时会创建一个新的数据库连接。但是此时，连接数已经达到了MaxTotal设定的最大值。则连接池就不会为等待线程新建数据库连接。而是强制让该线程进入等待队列。直到有其他线程归还数据库连接，再进行分配。MaxTotal起到限流保护数据库的作用，进入队列的线程不可能无限制的等待(这里可以参考maxWait)。 | total=active+idle  需要注意 |
| maxIdle | 8 | 最大的空闲的连接数，超过的空闲连接将被释放，如果设置为负数表示不限制（默认为8个，maxIdle不能设置太小，因为假如在高负载的情况下，连接的打开时间比关闭的时间快，会引起连接池中idle的个数 上升超过maxIdle，而造成频繁的连接销毁和创建，类似于jvm参数中的Xmx设置) |  |
| minIdle | 0 | 最小的空闲的连接数，低于这个数量会被创建新的连接（默认为0，调整为5，**该参数越接近maxIdle，性能越好**，因为连接的创建和销毁，都是需要消耗资源的；但是不能太大，因为在机器很空闲的时候，也会创建低于minidle个数的连接，类似于jvm参数中的Xmn设置） | 空闲连接低于此值自动创建新的连接的前提是你开启了连接回收器  timeBetweenEvictionRunsMillis |
| maxWait | 无限(ms) | 最大等待时间，当没有可用连接时，连接池等待连接释放的最大时间，超过该时间限制会抛出异常，如果设置-1表示无限等待（默认为无限，调整为60000ms，避免因线程池不够用，而导致请求被无限制挂起） |  |
| testOnBorrow | true | 指明在从池中租借对象时是否要进行验证有效，如果对象验证失败，则对象将从池子释放  借出连接时不要测试，否则很影响性能 |  |
| validationQuery |  | 测试SQL  SELECT |  |
| **timeBetweenEvictionRunsMillis** | -1L(ms) | 空闲连接回收器运行频率 |  |
| minEvictableIdleTimeMillis |  | 把空闲时间超过minEvictableIdleTimeMillis毫秒的连接断开, 直到连接池中的连接数到minIdle为止 |  |
| softMinEvictableIdleTimeMillis | -1L | 一个连接多久可以置为空闲连接，注意，如果设置了minEvictableIdleTimeMillis这个参数则无效 |  |
| numTestsPerEvictionRun | 3 | 每次空闲连接回收器线程(如果有)运行时检查的连接数量 |  |
| removeAbandonedOnMaintenance | false | 如果设置为true，则一个连接在超过removeAbandonedTimeout所设定的时间未使用即被认为是应该被抛弃并应该被移除的,注意，只有当  timeBetweenEvictionRunsMillis 为正值，从而启用池维护时，removeAbandonedOnMaintenance 才能生效。 | [防止数据库连接池泄露](http://wiki.jikexueyuan.com/project/tomcat/jdbc-data-sources.html) |
| removeAbandonedOnBorrow | false | true means abandoned connections may be removed when connections are borrowed from the pool. |
| removeAbandonedTimeout | 300 | 一个被抛弃连接可以被移除的超时时间，单位为秒 |

一些连接异常的处理

一.Too many connections



可能的产生条件：

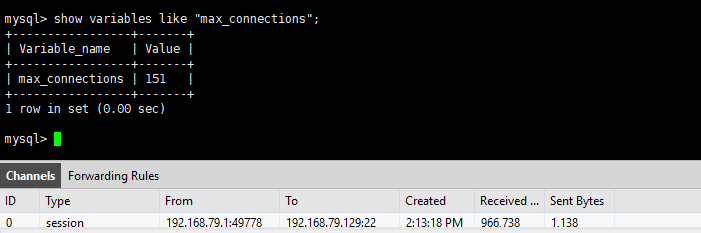
程序中使用连接后未关闭；而且正好DBCP连接池未配置连接泄露检测功能或者removeAbandonedTimeout时间设置过长；在Mysql的 wait\_timeout内就会产生死链(注:sleep状态),一旦死链的数量过多，在超过mysql 的最大连接数(max\_connections)时就会出现该异常；另外DBCP设置的maxTotal超过了mysql的最大连接数也有可能导致这种异常

(注:mysql获取最大连接数的方法)

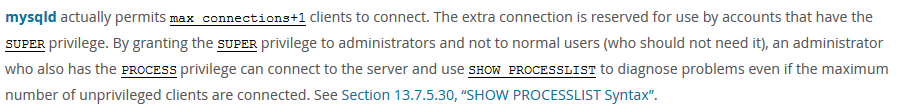
一种不关闭连接的写法

1. **for**(**int** i=0;i<10000;i++){
2. Connection connection = dataSource.getConnection();
3. Statement statement = connection.createStatement();
4. work(connection,i,40L);
5. }

(注:mysql获取最大连接数的方法)



关于mysql的max\_connections



<https://dev.mysql.com/doc/refman/5.5/en/too-many-connections.html>

Mysql Sleep线程：

定义：

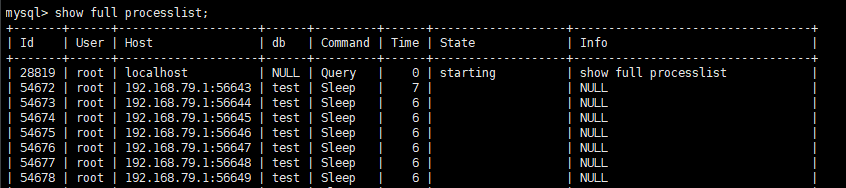
The thread is waiting for the client to send a new statement to it.

查看：

调用命令：

**SHOW [FULL] PROCESSLIST**

SHOW PROCESSLIST shows you which threads are running. You can also get this information from the INFORMATION\_SCHEMA PROCESSLIST table or the mysqladmin processlist command. If you have the PROCESS privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using). If you do not use the FULL keyword, only the first 100 characters of each statement are shown in the Info field.



解决方法：

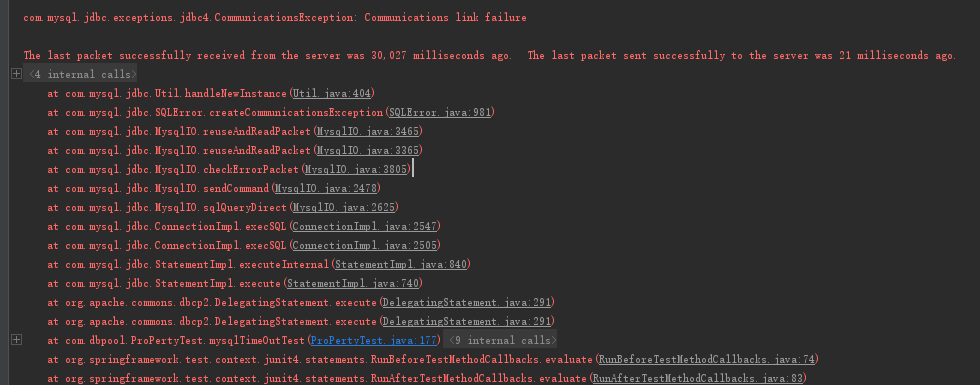
1.程序上用过的连接一定要关闭，防止死链

2.dbcp设置的maxTotal<mysql的最大连接数

3.dbcp启用定时检测泄露连接处理功能;

4.合理设置mysql的wait\_timeout值

二、Mysql8小时问题：



产生条件：

在mysql 的wait\_timeout设置的时间内，某个连接失效，而这个连接又被程序使用时会出现此异常

1. /\*\*
2. \* 测试Mysql 8小时问题
3. \* 需要关闭dbcp的维护功能(即将timeBetweenEvictionRunsMillis设置为一个负数或者做生意一个大于Mysql timeout
4. \* 的时间)
5. \* 查看mysql 的timeout:show global variables like '%timeout%'
6. \* 设置mysql 的timeout:set global wait\_timeout=20 (单位是秒默认8h)
7. \*
8. \* 解决方案：
9. \*
10. \* @throws SQLException
11. \* @throws InterruptedException
12. \*/
13. @Test
14. **public** **void** mysqlTimeOutTest() **throws** SQLException, InterruptedException {
15. //设置mysql timeout时间为20秒
16. Connection connection = dataSource.getConnection();
17. System.out.println("activeNumber:"+dataSource.getNumActive());
18. System.out.println("idleNumber:"+dataSource.getNumIdle());
19. connection.close();
20. TimeUnit.SECONDS.sleep(30);
21. //查看池状态
22. System.out.println("activeNumber:"+dataSource.getNumActive());
23. System.out.println("idleNumber:"+dataSource.getNumIdle());
24. connection = dataSource.getConnection();
25. Statement statement = connection.createStatement();
26. //随便取一个连接来测试
27. statement.execute("SELECT '\*' FROM DUAL");
28. statement.close();
29. connection.close();
30. }

解决方法：

1.开启dbcp的维护线程timeBetweenEvictionRunsMillis 其时间要小于wait\_timeout

2.idle连接设置一个合理的minEvictableIdleTimeMillis时间，小于wait\_timeout

C3P0连接池：

简介：

是一个开源的JDBC连接池，它实现了数据源和JNDI绑定，支持JDBC3规范和JDBC2的标准扩展。目前使用它的开源项目有Hibernate，Spring等

基本配置：

|  |  |
| --- | --- |
| 属性名 | 说明 |
| driverClass | 驱动类名 |
| jdbcUrl | URL |
| password | 密码 |
| user | 用户名 |

关键配置：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性名 | 默认值 参考C3P0Defaults | 说明 | 备注 |
| acquireIncrement | 3 | 连接池在无**空闲连接**可用时一次性创建的新数据库连接数 |  |
| testConnectionOnCheckin | false | 如果为true，则在close的时候测试连接的有效性 |  |
| testConnectionOnCheckout | false | 性能消耗大。如果为true，在每次getConnection的时候都会测试，为了提高性能,尽量不要用 |  |
| preferredTestQuery | null | 测试连接的sql语句 |  |
| maxConnectionAge | 0(秒) | 配置连接的生存时间，超过这个时间的连接将由连接池自动断开丢弃掉。当然正在使用的连接不会马上断开，而是等待它close再断开。配置为0的时候则不会对连接的生存时间进行限制 |  |
| **acquireRetryAttempts** | 30 | 连接池在获得新连接失败时重试的次数，如果小于等于0则无限重试直至连接获得成功 |  |
| **acquireRetryDelay** | 1000(ms) | 连接池在获得新连接时的间隔时间 |  |
| **maxIdleTime** | 0s | 最大空闲时间, 如果超过这个时间，某个数据库连接还没有被使用，则会断开掉这个连接  如果为0，则永远不会断开连接 |  |
| minPoolSize | 3 | 连接池保持的最小连接数，后面的  maxIdleTimeExcessConnections跟这个配合使用来减轻连接池的负载 |  |
| **maxIdleTimeExcessConnections** | 0s | 这个配置主要是为了减轻连接池的负载，比如连接池中连接数因为某次数据访问高峰导致创建了很多数据连接  但是后面的时间段需要的数据库连接数很少，则此时连接池完全没有必要维护那么多的连接，所以有必要将  断开丢弃掉一些连接来减轻负载，必须小于maxIdleTime。配置不为0，则会将连接池中的连接数量保持到minPoolSize。为0则不处理 | 注意与  maxIdlelTime区分 |
| **idleConnectionTestPeriod** | 0 | 用来配置测试空闲连接的间隔时间。测试方式还是上面的两种之一，可以用来解决MySQL8小时断开连接的问题。因为它保证连接池会每隔一定时间对空闲连接进行一次测试，从而保证有效的空闲连接能每隔一定时间访问一次数据库，将于MySQL  8小时无会话的状态打破。为0则不测试 |  |
| breakAfterAcquireFailure | false | 如果为true，则当连接获取失败时自动**关闭数据源**  (dataSource关闭)，除非重新启动应用程序。所以一般不用 | 尝试对一个已经关闭的数据源获取连接你会看到  Attempted to use a closed or broken resource pool |

DRUID连接池：

简介：

是Java语言中最好的数据库连接池。Druid能够提供强大的监控和扩展功能。

基本配置：

|  |  |  |
| --- | --- | --- |
| 属性名 | 说明 | 备注 |
| driverClassName | 驱动类名 | 如果不配置druid会根据url自动识别dbType，然后选择相应的driverClassName |
| url | URL |  |
| password | 密码 |  |
| username | 用户名 |  |

关键配置：

|  |  |  |  |
| --- | --- | --- | --- |
| 属性 | 默认值 | 说明 | 备注 |
| initialSize | 0 | 初始化时建立物理连接的个数。初始化发生在显示调用init方法，或者第一次getConnection时  参见DBCP中的initialSize属性 |  |
| maxActive | 8 | 参见DBCP中的maxTotal属性 |  |
| minIdle | 0 | 最小连接池数量 |  |
| maxWait | -1  ms | 获取连接时最大等待时间，单位毫秒。配置了maxWait之后，**缺省**启用**公平锁**，并发效率会有所下降，如果需要可以通过配置useUnfairLock属性为true使用非公平锁。 | 参考：  DruidDataSource中的线程：  DestroyTask |
| testOnBorrow | true | 申请连接时执行validationQuery检测连接是否有效，做了这个配置**会降低性能。** |  |
| testOnReturn | false | 归还连接时执行validationQuery检测连接是否有效，做了这个配置**会降低性能** |  |
| **keepAlive** | false | 由于Druid的testWhileIdle与DBCP的含意不同，那么如何达到与DBCP的testWhileIdle的效果？这里添加了一个keepAlive属性，其实现的效果与DBCP中的testWhileIdle相同，即维护idle连接的有效性(个人看法)  设置为true后  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。 | 在Druid-1.0.27之前的版本，DruidDataSource建议使用TestWhileIdle来保证连接的有效性，但仍有很多场景需要对连接进行保活处理。在1.0.28版本之后，新加入keepAlive配置，缺省关闭。 |
| **testWhileIdle** | false | 建议配置为true，不影响性能，并且保证安全性。申请连接的时候检测，如果空闲时间大于  timeBetweenEvictionRunsMillis，执行validationQuery检测连接是否有效，如果开启了testOnBorrow则此属性无效，该属性与  timeBetweenEvictionRunsMillis无关，如果没有设置  timeBetweenEvictionRunsMillis，或者设置了一个非法值则采用默认值 1分钟，在该空闲连接的空闲时间大于1分钟的时候才会执行SQL校验，而testOnBorrow是每次取连接时都会执行SQL校验，相对来说testWhileIdle对性能的影响不大，建议开启，相比DBCP而言…… | 源码参考  DruidDataSource中的  getConnectionDirect方法 |
| validationQuery |  | 用来检测连接是否有效的sql，要求是一个查询语句。如果validationQuery为null，testOnBorrow、testOnReturn、 testWhileIdle都不会其作用。在mysql中通常为select ‘x‘，在oracle中通常为select 1 from dual |  |
| **timeBetweenEvictionRunsMillis** | 60 \* 1000L ms | 1) Destroy线程会检测连接的间隔时间  2) testWhileIdle的判断依据 |  |
| **minEvictableIdleTimeMillis** | 1000L \* 60L \* 30L  Ms | Destory线程中如果检测到当前连接的最后活跃时间和当前时间的差值大于  minEvictableIdleTimeMillis，则关闭当前连接。 |  |
| **removeAbandoned** | null | 是否将超过removeAbandonedTimeout的连接关闭 |  |
| **removeAbandonedTimeout** | 300 \* 1000 s | 指定连接建立多长时间就需要被强制关闭 |  |
| **filters** |  | 属性类型是字符串，通过别名的方式配置扩展插件，常用的插件有：  1.监控统计用的filter:stat 2日志用的filter:log4j  3 防御sql注入的filter:wall |  |

对比