**Basic Data Source Configuration Parameters**

|  |  |
| --- | --- |
| **Parameter** | **Description** |
| username | The connection user name to be passed to our JDBC driver to establish a connection. |
| password | The connection password to be passed to our JDBC driver to establish a connection. |
| url | The connection URL to be passed to our JDBC driver to establish a connection. |
| driverClassName | The fully qualified Java class name of the JDBC driver to be used. |
| connectionProperties | The connection properties that will be sent to our JDBC driver when establishing new connections.  Format of the string must be [propertyName=property;]\*  **NOTE** - The "user" and "password" properties will be passed explicitly, so they do not need to be included here. |

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default** | **Description** |
| defaultAutoCommit | driver default | The default auto-commit state of connections created by this pool. If not set then the setAutoCommit method will not be called. |
| defaultReadOnly | driver default | The default read-only state of connections created by this pool. If not set then the setReadOnly method will not be called. (Some drivers don't support read only mode, ex: Informix) |
| defaultTransactionIsolation | driver default | The default TransactionIsolation state of connections created by this pool. One of the following: (see [javadoc](http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Connection.html" \l "field_summary))   * NONE * READ\_COMMITTED * READ\_UNCOMMITTED * REPEATABLE\_READ * SERIALIZABLE |
| defaultCatalog |  | The default catalog of connections created by this pool. |
| cacheState | true | If true, the pooled connection will cache the current readOnly and autoCommit settings when first read or written and on all subsequent writes. This removes the need for additional database queries for any further calls to the getter. If the underlying connection is accessed directly and the readOnly and/or autoCommit settings changed the cached values will not reflect the current state. In this case, caching should be disabled by setting this attribute to false. |
| defaultQueryTimeout | null | If non-null, the value of this Integer property determines the query timeout that will be used for Statements created from connections managed by the pool. null means that the driver default will be used. |
| enableAutoCommitOnReturn | true | If true, connections being returned to the pool will be checked and configured with Connection.setAutoCommit(true) if the auto commit setting is false when the connection is returned. |
| rollbackOnReturn | true | True means a connection will be rolled back when returned to the pool if auto commit is not enabled and the connection is not read-only. |

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default** | **Description** |
| initialSize | 0 | The initial number of connections that are created when the pool is started.  Since: 1.2 |
| maxTotal | 8 | The maximum number of active connections that can be allocated from this pool at the same time, or negative for no limit. |
| maxIdle | 8 | The maximum number of connections that can remain idle in the pool, without extra ones being released, or negative for no limit. |
| minIdle | 0 | The minimum number of connections that can remain idle in the pool, without extra ones being created, or zero to create none. |
| maxWaitMillis | indefinitely | The maximum number of milliseconds that the pool will wait (when there are no available connections) for a connection to be returned before throwing an exception, or -1 to wait indefinitely. |

https://commons.apache.org/proper/commons-dbcp/images/icon_warning_sml.gif **NOTE**: If maxIdle is set too low on heavily loaded systems it is possible you will see connections being closed and almost immediately new connections being opened. This is a result of the active threads momentarily closing connections faster than they are opening them, causing the number of idle connections to rise above maxIdle. The best value for maxIdle for heavily loaded system will vary but the default is a good starting point.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default** | **Description** |
| validationQuery |  | The SQL query that will be used to validate connections from this pool before returning them to the caller. If specified, this query **MUST** be an SQL SELECT statement that returns at least one row. If not specified, connections will be validation by calling the isValid() method. |
| validationQueryTimeout | no timeout | The timeout in seconds before connection validation queries fail. If set to a positive value, this value is passed to the driver via the setQueryTimeout method of the Statement used to execute the validation query. |
|  |  |  |
| testOnCreate | false | The indication of whether objects will be validated after creation. If the object fails to validate, the borrow attempt that triggered the object creation will fail. |
| testOnBorrow | true | The indication of whether objects will be validated before being borrowed from the pool. If the object fails to validate, it will be dropped from the pool, and we will attempt to borrow another. |
| testOnReturn | false | The indication of whether objects will be validated before being returned to the pool. |
| testWhileIdle | false | The indication of whether objects will be validated by the idle object evictor (if any). If an object fails to validate, it will be dropped from the pool. |
| timeBetweenEvictionRunsMillis | -1 | The number of milliseconds to sleep between runs of the idle object evictor thread. When non-positive, no idle object evictor thread will be run. |
| numTestsPerEvictionRun | 3 | The number of objects to examine during each run of the idle object evictor thread (if any). |
| minEvictableIdleTimeMillis | 1000 \* 60 \* 30 | The minimum amount of time an object may sit idle in the pool before it is eligible for eviction by the idle object evictor (if any). |
| softMinEvictableIdleTimeMillis | -1 | The minimum amount of time a connection may sit idle in the pool before it is eligible for eviction by the idle connection evictor, with the extra condition that at least "minIdle" connections remain in the pool. When minEvictableIdleTimeMillis is set to a positive value, minEvictableIdleTimeMillis is examined first by the idle connection evictor - i.e. when idle connections are visited by the evictor, idle time is first compared against minEvictableIdleTimeMillis (without considering the number of idle connections in the pool) and then against softMinEvictableIdleTimeMillis, including the minIdle constraint. |
| maxConnLifetimeMillis | -1 | The maximum lifetime in milliseconds of a connection. After this time is exceeded the connection will fail the next activation, passivation or validation test. A value of zero or less means the connection has an infinite lifetime. |
| logExpiredConnections | true | Flag to log a message indicating that a connection is being closed by the pool due to maxConnLifetimeMillis exceeded. Set this property to false to suppress expired connection logging that is turned on by default. |
| connectionInitSqls | null | A Collection of SQL statements that will be used to initialize physical connections when they are first created. These statements are executed only once - when the configured connection factory creates the connection. |
| lifo | true | True means that borrowObject returns the most recently used ("last in") connection in the pool (if there are idle connections available). False means that the pool behaves as a FIFO queue - connections are taken from the idle instance pool in the order that they are returned to the pool. |

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default** | **Description** |
| poolPreparedStatements | false | Enable prepared statement pooling for this pool. |
| maxOpenPreparedStatements | unlimited | The maximum number of open statements that can be allocated from the statement pool at the same time, or negative for no limit. |

https://commons.apache.org/proper/commons-dbcp/images/icon_info_sml.gif This component has also the ability to pool PreparedStatements. When enabled a statement pool will be created for each Connection and PreparedStatements created by one of the following methods will be pooled:

* public PreparedStatement prepareStatement(String sql)
* public PreparedStatement prepareStatement(String sql, int resultSetType, int resultSetConcurrency)

https://commons.apache.org/proper/commons-dbcp/images/icon_warning_sml.gif **NOTE** - Make sure your connection has some resources left for the other statements. Pooling PreparedStatements may keep their cursors open in the database, causing a connection to run out of cursors, especially if maxOpenPreparedStatements is left at the default (unlimited) and an application opens a large number of different PreparedStatements per connection. To avoid this problem, maxOpenPreparedStatements should be set to a value less than the maximum number of cursors that can be open on a Connection.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default** | **Description** |
| accessToUnderlyingConnectionAllowed | false | Controls if the PoolGuard allows access to the underlying connection. |

When allowed you can access the underlying connection using the following construct:

Connection conn = ds.getConnection();

Connection dconn = ((DelegatingConnection) conn).getInnermostDelegate();

...

conn.close()

https://commons.apache.org/proper/commons-dbcp/images/icon_info_sml.gif Default is false, it is a potential dangerous operation and misbehaving programs can do harmful things. (closing the underlying or continue using it when the guarded connection is already closed) Be careful and only use when you need direct access to driver specific extensions.

https://commons.apache.org/proper/commons-dbcp/images/icon_warning_sml.gif **NOTE:** Do not close the underlying connection, only the original one.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default** | **Description** |
| removeAbandonedOnMaintenance  removeAbandonedOnBorrow | false | Flags to remove abandoned connections if they exceed the removeAbandonedTimout. A connection is considered abandoned and eligible for removal if it has not been used for longer than removeAbandonedTimeout. Creating a Statement, PreparedStatement or CallableStatement or using one of these to execute a query (using one of the execute methods) resets the lastUsed property of the parent connection. Setting one or both of these to true can recover db connections from poorly written applications which fail to close connections. Setting removeAbandonedOnMaintenance to true removes abandoned connections on the maintenance cycle (when eviction ends). This property has no effect unless maintenance is enabled by setting timeBetweenEvictionRunsMillis to a positive value.  If removeAbandonedOnBorrow is true, abandoned connections are removed each time a connection is borrowed from the pool, with the additional requirements that   * getNumActive() > getMaxTotal() - 3; and * getNumIdle() < 2 |
| removeAbandonedTimeout | 300 | Timeout in **seconds** before an abandoned connection can be removed. |
| logAbandoned | false | Flag to log stack traces for application code which abandoned a Statement or Connection. Logging of abandoned Statements and Connections adds overhead for every Connection open or new Statement because a stack trace has to be generated. |
| abandonedUsageTracking | false | If true, the connection pool records a stack trace every time a method is called on a pooled connection and retains the most recent stack trace to aid debugging of abandoned connections. There is significant overhead added by setting this to true. |

https://commons.apache.org/proper/commons-dbcp/images/icon_info_sml.gif If you have enabled removeAbandonedOnMaintenance or removeAbandonedOnBorrow then it is possible that a connection is reclaimed by the pool because it is considered to be abandoned. This mechanism is triggered when (getNumIdle() < 2) and (getNumActive() > getMaxTotal() - 3) and removeAbandonedOnBorrow is true; or after eviction finishes and removeAbandonedOnMaintenance is true. For example, maxTotal=20 and 18 active connections and 1 idle connection would trigger removeAbandonedOnBorrow, but only the active connections that aren't used for more then "removeAbandonedTimeout" seconds are removed (default 300 sec). Traversing a resultset doesn't count as being used. Creating a Statement, PreparedStatement or CallableStatement or using one of these to execute a query (using one of the execute methods) resets the lastUsed property of the parent connection.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Default** | **Description** |
| fastFailValidation | false | When this property is true, validation fails fast for connections that have thrown "fatal" SQLExceptions. Requests to validate disconnected connections fail immediately, with no call to the driver's isValid method or attempt to execute a validation query. The SQL\_STATE codes considered to signal fatal errors are by default the following:   * 57P01 (ADMIN SHUTDOWN) * 57P02 (CRASH SHUTDOWN) * 57P03 (CANNOT CONNECT NOW) * 01002 (SQL92 disconnect error) * JZ0C0 (Sybase disconnect error) * JZ0C1 (Sybase disconnect error) * Any SQL\_STATE code that starts with "08"   To override this default set of disconnection codes, set the disconnectionSqlCodes property. |
| disconnectionSqlCodes | null | Comma-delimited list of SQL\_STATE codes considered to signal fatal disconnection errors. Setting this property has no effect unless fastFailValidation is set to true. |