<https://github.com/brettwooldridge/HikariJSON>

hikaricp前度的官网：

[https://jolbox.com](https://jolbox.com/)

<https://github.com/brettwooldridge/HikariCP>

官网：

<http://brettwooldridge.github.io/HikariCP/>

配置参数：

<https://github.com/brettwooldridge/HikariCP#configuration-knobs-baby>

<https://github.com/brettwooldridge/HikariCP>

HikariCP comes with sane defaults that perform well in most deployments without additional tweaking. **Every property is optional, except for the "essentials" marked below.**

📎 HikariCP uses milliseconds for all time values.

🚨 HikariCP relies on accurate timers for both performance and reliability. It is imperative that your server is synchronized with a time-source such as an NTP server. Especially if your server is running within a virtual machine. Why? [Read more here](https://dba.stackexchange.com/a/171020). **Do not rely on hypervisor settings to "synchronize" the clock of the virtual machine. Configure time-source synchronization inside the virtual machine.** If you come asking for support on an issue that turns out to be caused by lack time synchronization, you will be taunted publicly on Twitter.

# Essentials

## 🔠dataSourceClassName

This is the name of the DataSource class provided by the JDBC driver. Consult the documentation for your specific JDBC driver to get this class name, or see the [table](https://github.com/brettwooldridge/HikariCP#popular-datasource-class-names) below. Note XA data sources are not supported. XA requires a real transaction manager like [bitronix](https://github.com/bitronix/btm). Note that you do not need this property if you are using jdbcUrl for "old-school" DriverManager-based JDBC driver configuration. Default: none

- or -

## 🔠jdbcUrl

This property directs HikariCP to use "DriverManager-based" configuration. We feel that DataSource-based configuration (above) is superior for a variety of reasons (see below), but for many deployments there is little significant difference. **When using this property with "old" drivers, you may also need to set the driverClassName property, but try it first without.**Note that if this property is used, you may still use DataSource properties to configure your driver and is in fact recommended over driver parameters specified in the URL itself. Default: none

## 🔠username

This property sets the default authentication username used when obtaining Connections from the underlying driver. Note that for DataSources this works in a very deterministic fashion by calling DataSource.getConnection(\*username\*, password)on the underlying DataSource. However, for Driver-based configurations, every driver is different. In the case of Driver-based, HikariCP will use this username property to set a user property in the Properties passed to the driver's DriverManager.getConnection(jdbcUrl, props) call. If this is not what you need, skip this method entirely and call addDataSourceProperty("username", ...), for example. Default: none

## 🔠password

This property sets the default authentication password used when obtaining Connections from the underlying driver. Note that for DataSources this works in a very deterministic fashion by calling DataSource.getConnection(username, \*password\*)on the underlying DataSource. However, for Driver-based configurations, every driver is different. In the case of Driver-based, HikariCP will use this password property to set a password property in the Properties passed to the driver's DriverManager.getConnection(jdbcUrl, props) call. If this is not what you need, skip this method entirely and call addDataSourceProperty("pass", ...), for example. Default: none

# Frequently used

## ✅autoCommit

This property controls the default auto-commit behavior of connections returned from the pool. It is a boolean value.Default: true

## ⌚connectionTimeout

This property controls the maximum number of milliseconds that a client (that's you) will wait for a connection from the pool. If this time is exceeded without a connection becoming available, a SQLException will be thrown. Lowest acceptable connection timeout is 250 ms. Default: 30000 (30 seconds)

## ⌚idleTimeout

This property controls the maximum amount of time that a connection is allowed to sit idle in the pool. **This setting only applies when minimumIdle is defined to be less than maximumPoolSize.** Idle connections will not be retired once the pool reaches minimumIdle connections. Whether a connection is retired as idle or not is subject to a maximum variation of +30 seconds, and average variation of +15 seconds. A connection will never be retired as idle before this timeout. A value of 0 means that idle connections are never removed from the pool. The minimum allowed value is 10000ms (10 seconds). Default: 600000 (10 minutes)

## ⌚maxLifetime

This property controls the maximum lifetime of a connection in the pool. An in-use connection will never be retired, only when it is closed will it then be removed. On a connection-by-connection basis, minor negative attenuation is applied to avoid mass-extinction in the pool. **We strongly recommend setting this value, and it should be several seconds shorter than any database or infrastructure imposed connection time limit.** A value of 0 indicates no maximum lifetime (infinite lifetime), subject of course to the idleTimeout setting. Default: 1800000 (30 minutes)

## 🔠connectionTestQuery

**If your driver supports JDBC4 we strongly recommend not setting this property.** This is for "legacy" drivers that do not support the JDBC4 Connection.isValid() API. This is the query that will be executed just before a connection is given to you from the pool to validate that the connection to the database is still alive. Again, try running the pool without this property, HikariCP will log an error if your driver is not JDBC4 compliant to let you know. Default: none

## 🔢minimumIdle

This property controls the minimum number of idle connections that HikariCP tries to maintain in the pool. If the idle connections dip below this value and total connections in the pool are less than maximumPoolSize, HikariCP will make a best effort to add additional connections quickly and efficiently. However, for maximum performance and responsiveness to spike demands, we recommend not setting this value and instead allowing HikariCP to act as a fixed size connection pool. Default: same as maximumPoolSize

## 🔢maximumPoolSize

This property controls the maximum size that the pool is allowed to reach, including both idle and in-use connections. Basically this value will determine the maximum number of actual connections to the database backend. A reasonable value for this is best determined by your execution environment. When the pool reaches this size, and no idle connections are available, calls to getConnection() will block for up to connectionTimeout milliseconds before timing out. Please read [about pool sizing](https://github.com/brettwooldridge/HikariCP/wiki/About-Pool-Sizing). Default: 10

## 📈metricRegistry

This property is only available via programmatic configuration or IoC container. This property allows you to specify an instance of a Codahale/Dropwizard MetricRegistry to be used by the pool to record various metrics. See the [Metrics](https://github.com/brettwooldridge/HikariCP/wiki/Dropwizard-Metrics) wiki page for details. Default: none

## 📈healthCheckRegistry

This property is only available via programmatic configuration or IoC container. This property allows you to specify an instance of a Codahale/Dropwizard HealthCheckRegistry to be used by the pool to report current health information. See the [Health Checks](https://github.com/brettwooldridge/HikariCP/wiki/Dropwizard-HealthChecks) wiki page for details. Default: none

## 🔠poolName

This property represents a user-defined name for the connection pool and appears mainly in logging and JMX management consoles to identify pools and pool configurations. Default: auto-generated

# Infrequently used

## ⌚initializationFailTimeout

This property controls whether the pool will "fail fast" if the pool cannot be seeded with an initial connection successfully. Any positive number is taken to be the number of milliseconds to attempt to acquire an initial connection; the application thread will be blocked during this period. If a connection cannot be acquired before this timeout occurs, an exception will be thrown. This timeout is applied after the connectionTimeout period. If the value is zero (0), HikariCP will attempt to obtain and validate a connection. If a connection is obtained, but fails validation, an exception will be thrown and the pool not started. However, if a connection cannot be obtained, the pool will start, but later efforts to obtain a connection may fail. A value less than zero will bypass any initial connection attempt, and the pool will start immediately while trying to obtain connections in the background. Consequently, later efforts to obtain a connection may fail. Default: 1

## ❎isolateInternalQueries

This property determines whether HikariCP isolates internal pool queries, such as the connection alive test, in their own transaction. Since these are typically read-only queries, it is rarely necessary to encapsulate them in their own transaction. This property only applies if autoCommit is disabled. Default: false

## ❎allowPoolSuspension

This property controls whether the pool can be suspended and resumed through JMX. This is useful for certain failover automation scenarios. When the pool is suspended, calls to getConnection() will not timeout and will be held until the pool is resumed. Default: false

## ❎readOnly

This property controls whether Connections obtained from the pool are in read-only mode by default. Note some databases do not support the concept of read-only mode, while others provide query optimizations when the Connection is set to read-only. Whether you need this property or not will depend largely on your application and database. Default: false

## ❎registerMbeans

This property controls whether or not JMX Management Beans ("MBeans") are registered or not. Default: false

## 🔠catalog

This property sets the default catalog for databases that support the concept of catalogs. If this property is not specified, the default catalog defined by the JDBC driver is used. Default: driver default

## 🔠connectionInitSql

This property sets a SQL statement that will be executed after every new connection creation before adding it to the pool. If this SQL is not valid or throws an exception, it will be treated as a connection failure and the standard retry logic will be followed. Default: none

## 🔠driverClassName

HikariCP will attempt to resolve a driver through the DriverManager based solely on the jdbcUrl, but for some older drivers the driverClassName must also be specified. Omit this property unless you get an obvious error message indicating that the driver was not found. Default: none

## 🔠transactionIsolation

This property controls the default transaction isolation level of connections returned from the pool. If this property is not specified, the default transaction isolation level defined by the JDBC driver is used. Only use this property if you have specific isolation requirements that are common for all queries. The value of this property is the constant name from the Connectionclass such as TRANSACTION\_READ\_COMMITTED, TRANSACTION\_REPEATABLE\_READ, etc. Default: driver default

## ⌚validationTimeout

This property controls the maximum amount of time that a connection will be tested for aliveness. This value must be less than the connectionTimeout. Lowest acceptable validation timeout is 250 ms. Default: 5000

## ⌚leakDetectionThreshold

This property controls the amount of time that a connection can be out of the pool before a message is logged indicating a possible connection leak. A value of 0 means leak detection is disabled. Lowest acceptable value for enabling leak detection is 2000 (2 seconds). Default: 0

## ➡dataSource

This property is only available via programmatic configuration or IoC container. This property allows you to directly set the instance of the DataSource to be wrapped by the pool, rather than having HikariCP construct it via reflection. This can be useful in some dependency injection frameworks. When this property is specified, the dataSourceClassName property and all DataSource-specific properties will be ignored. Default: none

## 🔠schema

This property sets the default schema for databases that support the concept of schemas. If this property is not specified, the default schema defined by the JDBC driver is used. Default: driver default

## ➡threadFactory

This property is only available via programmatic configuration or IoC container. This property allows you to set the instance of the java.util.concurrent.ThreadFactory that will be used for creating all threads used by the pool. It is needed in some restricted execution environments where threads can only be created through a ThreadFactory provided by the application container. Default: none

## ➡scheduledExecutor

This property is only available via programmatic configuration or IoC container. This property allows you to set the instance of the java.util.concurrent.ScheduledExecutorService that will be used for various internally scheduled tasks. If supplying HikariCP with a ScheduledThreadPoolExecutor instance, it is recommended that setRemoveOnCancelPolicy(true) is used.Default: none