Commons Pool and DBCP

Version 2 Update

(DBCP 2.0, pool 2.2)

Phil Steitz

Apachecon US, 2014

Apache Commons™
http://commons.apache.org/

APACHECON 2014

Back to Life ...

commons Pool

commons DBCP

Apache Commons™
http://commons.apache.org/

APACHECON 2014

Join Us!

- Version 2’s are new
- Code is scrutable
- Interesting problems
- Patches welcome!

- End of Commercial Message -

Pool and DBCP

- Born around the same time as Commons itself (spring, 2003)
- DBCP provides the database connection pool for Tomcat exposed as a JNDI DataSource
- Pool provides the underlying object pool for DBCP connections in GenericObjectPool and prepared statements in GenericKeyedObjectPool

commons Pool

commons DBCP

Pool Features

- Simple object pool and instance factory interfaces
- Multiple pool implementations
- Most widely used impl is GenericObjectPool (GOP)
  - configurable maintenance thread
  - control over instance count, idle count, instance age, abandonment
  - client wait time, action on pool depletion configurable
  - instance validation on borrow, return, while idle
  - LIFO / FIFO behavior configurable
- GenericKeyedObjectPool provides GOP functionality for a map of pools

Pool Features (cont.)

- StackObjectPool
  - FIFO behavior, simple instance stack
  - No limit to instances in circulation
- SoftReferenceObjectPool
  - Pools soft references
  - No limit to instances in circulation
- KeyedObjectPools
  - GenericKeyedObjectPool
  - StackKeyedObjectPool
- PoolUtils

DBCP Features

- Pool-backed DataSource implementations
  - BasicDataSource
  - PoolingDataSource
  - BasicManagedDataSource
  - SharedPoolDataSource
- Statement pooling
- Abandoned connection cleanup
- Connection validation
- “Eviction” of connections idle too long in the pool

DBCP Features (cont)

- Support for JDBC 3 (JDK 1.4-1.5), JDBC 4 (JDK 1.6) and JDBC 4.1 (JDK 1.7)
  - DBCP 1.3.x implements JDBC 3
  - DBCP 1.4.x implements JDBC 4
  - DBCP 2.x implements JDBC 4.1
  - 1.3.x will not likely see additional releases
- Creates JDBC connections using Driver- DriverManager- and DataSource-based physical ConnectionFactories
- Can expose connection pool via a Driver that can be registered and accessed using DriverManager

Pool 2

- Core pooling algorithms rewritten for scale / performance
- Metrics and JMX instrumentation
- Robust instance tracking, proxy support
- Factories given access to instance tracking data
- More flexible idle instance “eviction” configuration
- Cleaned up and rationalized configuration API

DBCP 2

- JMX instrumentation
- Improved connection lifecycle management
- Logging
- Performance improvements
- SecurityManager integration

Pool 2 API Changes

Version 2 object factories create and manage PooledObject wrappers

- Borrow / Return methods still deliver / take unwrapped object instances; but factory methods create and manage **PooledObject** wrappers
- Wrappers encapsulate pooled object state and tracking data
- Factories can use tracking data in instance validation

Example: DBCP using PooledObject wrapper to limit connection lifetime

```
validateLifetime(PooledObject<PoolableConnection> p)
            throws Exception {
   if (maxConnLifetimeMillis > 0) {
       long lifetime = System.currentTimeMillis() –
                                p.getCreateTime();
       if (lifetime > maxConnLifetimeMillis) {
            throw …
       }
    }
 }
```

Called by connection factory activate, passivate, validate methods

Apache Commons™
http://commons.apache.org/

APACHECON 2014

Version 2 Factory Interface

```
public interface PooledObjectFactory<T> {
  PooledObject<T> makeObject() throws Exception;
  void destroyObject(PooledObject<T> p) throws Exception;
  boolean validateObject(PooledObject<T> p);
  void activateObject(PooledObject<T> p) throws Exception;
  void passivateObject(PooledObject<T> p) throws Exception;
}
```

Basic lifecycle is the same as v 1 pools

- Pool invokes makeObject to create new instances as permitted
- Objects borrowed from the pool are always activated before being provided to clients, optionally validated (testOnBorrow)
- Objects returned to the pool are always passivated, optionally validated (testOnReturn)

Version 2 Core Client API

```
public interface ObjectPool<T> {
    T borrowObject() throws Exception;
    void returnObject(T obj) throws Exception;
    void invalidateObject(T obj) throws Exception;
    void addObject() throws Exception;
    …
```

Clients borrow / return unwrapped instances (like v 1)

- returnObject has to be able to find the PooledObject corresponding to the returning instance – **distinct instances must be discernible by equals**
- returnObject checks to make sure the returning instance came from the pool and has not already been returned (new in v 2)

PooledObject

```
interface PooledObject<T>
    T getObject();
    boolean allocate();
    boolean deallocate();
    void invalidate();
    void setLogAbandoned(boolean logAbandoned);
    PooledObjectState getState();
    void markAbandoned();
    void markReturning();
    void use();
```

+ getters for creation time, last idle time, last borrowed / returned / used time

PooledObject States

State	Meaning
IDLE	In the available instance queue, not in use
ALLOCATED	Allocated to a client – in use
EVICTION	In the available instance queue, currently being tested for possible eviction
VALIDATION	In the queue, being validated
EVICTION_RETURN_TO_HEAD	Attempt to borrow while being eviction tested
INVALID	Failed validation or eviction test – has been or will be destroyed
ABANDONED	Deemed abandoned – will be invalidated
RETURNING	Returning to the idle instance queue

Apache Commons™
http://commons.apache.org/

APACHECON 2014

DefaultPooledObject

- Default implementation of **PooledObject** interface
- Should generally use this as a base for user implementations
- **DefaultPooledObjectInfo** exposes properties in **DefaultPooledObjectInfoMBean** interface
- **GenericObjectPool** and **GenericKeyedObjectPool** provide **listAllObjects** methods that return these properties for all objects under management by the pool

Example: DBCP 2 PoolableConnectionFactory

```
PooledObject<PoolableConnection> makeObject() throws
Exception {
    Connection conn = _connFactory.createConnection();
  …
    PoolableConnection pc = new PoolableConnection(
                                  conn,_pool, connJmxName);
    return new DefaultPooledObject<>(pc);
}
void destroyObject(PooledObject<PoolableConnection> p)
            throws Exception {
    p.getObject().reallyClose();
}
```

Pool 2 JMX – *Pool Level*

- All pool config properties
- Total number of instances borrowed, returned, created, destroyed, destroyed by evictor, destroyed on validation failure
- Rolling stats: mean active / idle times, mean / max borrow wait time
- Number of pool waiters
- Default names “pool”, “pool1”, …

Pool 2 JMX – *Instance Level*

- Creation time, last borrowed time, last returned time
- Number of times borrowed
- Stack trace of last borrow (if abandoned instance tracking is enabled) and use (if usage tracking enabled)

Eviction Policy

Interface has one method:

```
boolean evict(EvictionConfig config, PooledObject<T> underTest,
                                                  int idleCount);
```

DefaultEvictionPolicy behaves similarly to pool 1

- "soft" criteria honors minIdle setting, "hard" ignores it

PooledObject tracking info enables advanced schemes

Some Implementation Details

- Idle instances for GOP / GKOP are held in (slightly modified) LinkedBlockingDeque
  - Harmony 1.6 sources modified to enable take waiters to be interrupted on pool close
- References to all PooledObjects under management held in

  ConcurrentHashMap<T, PooledObject<T>>

Some Implementation Details

BorrowObject Algorithm:

```
while (unserved) {
  try to get an instance from the idle instance queue
  if none available, try to create an instance
  else pollFirst on the queue (waiting borrowMaxWaitMillis if set to block)
  once served, activate and if so configured, validate instance
  activation / validation failures destroy instances and revert to unserved
}
```

Borrowing thread does create, destroy, validate actions

GenericObjectPool Configuration

Property	Meaning	Default	Notes
maxTotal	Maximum number of object instances in circulation (idle or checked out)	8	Negative means unlimited **Called "maxActive" in version 1**
maxIdle	The maximum number of instances that can be idle in the pool	8	Negative means unlimited Enforced when instances are returned to the pool

GenericObjectPool Configuration

Property	Meaning	Default	Notes
maxBorrowWaitTimeMillis	The maximum amount of time that borrowObject will wait for an instance to become available for checkout	-1	Negative means unlimited Only meaningful if blockWhenExhausted is true **Called "maxWait" in v1**
minIdle	The number of idle instances that the pool will try to keep available	0	Enforced when pool maintenance thread runs or when instances are destroyed by invalidate or validation failures. Limited by maxTotal

GenericObjectPool Configuration (cont.)

Property	Meaning	Default	Notes
blockWhenExhausted	True means threads block waiting for an instance; false means fail with NoSuchElementException	true	true enables maxBorrowWaitTimeMillis Replaces v1 **whenExhaustedAction** – FAIL is replaced by false setting; GROW is needless
timeBetweenEvictionRunsMillis	Time between pool maintenance runs in milliseconds	never runs	idle instance eviction and minIdle require maintenance thread

GenericObjectPool Configuration (cont.)

Property	Meaning	Default	Notes
minEvictableIdleTimeMillis	The number of milliseconds that an instance can sit idle in the pool before being eligible to be destroyed	30 minutes	Eviction only happens when the maintenance thread runs and visits the instance
softMinEvictableIdleTimeMillis	Like minEvictableIdleTime but with the additional requirement that there are at least minIdle instances in the pool at idle timeout	disabled	Enforced only when pool maintenance thread runs
numTestsPerEvictionRun	The maximum number of idle instances that the maintenance thread will visit when it runs	3	Cycles through the pool across runs

GenericObjectPool Configuration (cont.)

Property	Meaning	Default	Notes
testOnBorrow	Use the object factory's validate method to test instances retrieved from the pool	false	Failing instances are destroyed; borrow is retried until pool is exhausted
testOnReturn	Validate instances before returning them to the pool	false	Failing instances are destroyed
testWhileIdle	Test idle instances visited by the pool maintenance thread and destroy any that fail validation	false	Only meaningful if pool maintenance is enabled (timeBetweenEvictionRuns is positive)
lifo	Pool behaves as a LIFO queue	true	false means pool behaves as a LIFO queue

GenericObjectPool Config (v2 update)

Property	Meaning	Notes
removeAbandonedOnBorrow, removeAbandonedOnMaintenance	True means the pool will try to identify and remove "abandoned" connections on borrow / maintenance	Abandoned connection removal is triggered on borrow when a connection is requested and numIdle < 2 and numActive > maxActive – 3 On maintenance (eviction runs) no idle / active test is performed Objects are considered abandoned if they have not been used in more than removeAbandonedTimeout seconds
logAbandoned	True means stack traces for the code borrowing and (if available) last using abandoned instances is logged	
removeAbandonedTimeout	The amount of time between uses of a object before it is considered "abandoned."	Default value is 300 seconds; ignored if removeAbandoned is false
logWriter	PrintWriter used to log abandoned object info	

Apache Commons™
http://commons.apache.org/

APACHECON 2014

GenericObjectPool Config (new in v2)

Property	Meaning	Notes
evictionPolicyClassName	EvictionPolicy used by pool maintenance thread	Has no impact unless timeBetweenEvictionRunsMillis is positive Defaults to DefaultEvictionPolicy
jmxName	JMX name requested for the pool	If the requested name is not valid or available, or is null, an alternative is chosen
testOnCreate	True means instances will be validated on creation	Default is false Added in 2.2
usageTracking	True means the pool may record a stack trace every time an instance is used	Useful debugging abandoned object problems; adds overhead; configured via AbandonedConfig

Situations to Avoid

maxIdle << maxActive

If active count regularly grows to near maxActive and load is variable, setting maxIdle too small will result in lots of object churn (destroy on return, create on demand)

maxIdle too close to minIdle with frequent maintenance

Results in object churn as the pool struggles to keep the idle instance count in a narrow range

Too frequent maintenance

Maintenance thread can contend with client threads for pool and instance access

Poorly performing factory methods

Especially applies to validation methods if testOnBorrow, testOnReturn and / or testWhileIdle are enabled

DBCP 2 Performance Improvements

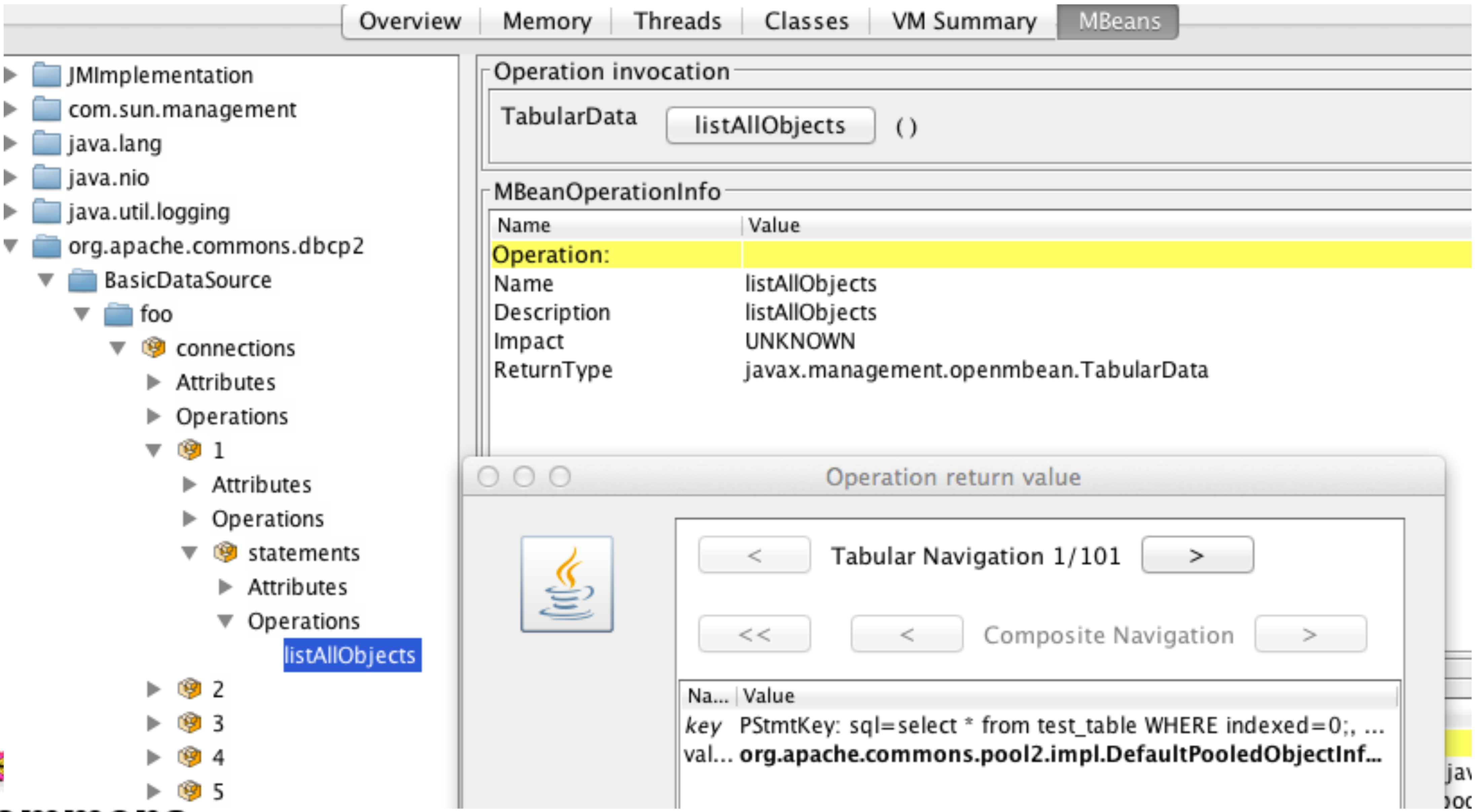
- Fewer database round trips for
  - Validation (can be configured to use isValid)
  - Property setting (can be configured to cache state)
  - Rollback (configurable on return)
- Reduce synchronization scope in key methods
- Better pool ☺

DBCP 2 API Improvements – *Better Control*

- Query timeouts
- Forced connection kill (via JMX)
- Force return to pool (via JMX)
- Rollback (configurable on return)
- Connection validation on creation
- Connection lifetime
- Custom eviction policies
- SecurityManager integration

DBCP 2 API Improvements - *Monitoring*

- Log connection validation failures
- Expose pool and datasource properties via JMX



MBeanOperationInfo

Name	Value
Operation:	
Name	listAllObjects
Description	listAllObjects
Impact	UNKNOWN
ReturnType	javax.management.openmbean.TabularData

Na...	Value
key	PStmtKey: sql=select * from test_table WHERE indexed=0;, ...
val...	**org.apache.commons.pool2.impl.DefaultPooledObjectInf...**

Apache Commons
http://commons.apache.org/

APACHECON 2014

DBCP JMX

Properties

- Datasource config properties
- Connection pool and statement pool properties
- Connection properties and PooledObject state
- Pooled statement PooledObject state
- Datasource closed / open

Operations (connection level)

- Force close physical
- Return to pool
- Refresh cached properties
- Clear warnings

BasicDataSource Configuration

Property	Meaning	Notes
maxTotal, maxIdle, minIdle, maxWaitMillis, testOnBorrow, testOnReturn, testWhileIdle, timeBetweenEvictionRunsMillis, numTestsPerEvictionRun	Same meanings and defaults as pool	blockWhenExhausted default is true **maxTotal used to be "maxActive" in v 1**
initialSize	Number of connections created and placed into the pool on initialization	Cannot exceed the maxActive pool property; default is 0
defaultAutoCommit defaultCatalog defaultReadOnly defaultTransactionIsolation connectionProperties	properties of connections provided by this DataSource	Clients can change these properties on checked out connections; but the value is reset to default on passivation
driverClassName	fully qualified class name of JDBC Driver used to manage physical connections	Must be available on the classpath at runtime
url username password	JDBC connection parameters shared by all connections in the pool	If set, username and password supersede values in connectionProperties

BasicDataSource Configuration (cont.)

Property	Meaning	Notes
validationQuery	SQL Query used to validate connections	Validation succeeds iff this query returns at least one row; testOnBorrow, testOnReturn, testWhileIdle are ignored if validationQuery is null
validationQueryTimeout	Timeout for validation queries	
connectInitSQLs	Initialization SQL executed once when a connection is first created	If any of the statements in the list throw exceptions, the connection is destroyed
poolPreparedStatements	true means a prepared statement pool is created for each connection	Pooling PreparedStatements may keep their cursors open in the database, causing a connection to run out of cursors
maxOpenPreparedStatements	maximum number of prepared statements that can be pooled per connection	Default is unlimited

BasicDataSource Configuration (cont.)

Property	Meaning	Notes
removeAbandonedOnBorrow, removeAbandonedOnMaintenance	True means the pool will try to identify and remove "abandoned" connections on borrow / maintenance	See pool notes
logAbandoned	True means stack traces for the code borrowing and (if abandonedUsageTracking) last using abandoned connections is logged	See pool notes
removeAbandonedTimeout	The amount of time between uses of a connection before it is considered "abandoned."	Default value is 300 seconds; ignored if removeAbandoned is false
logWriter	PrintWriter used to log abandoned connection info	

BasicDataSource Configuration (new in v2)

Property	Meaning	Notes
driverClassLoader	Classloader used to load JDBC driver	
defaultQueryTimeout	Timeout for queries made by connections from this datasource	Applied to Statement objects; null means use the driver default
enableAutoCommitOnReturn	True means connections being returned to the pool will set to auto commit if the auto commit setting is false on return	Default is true
cacheState	true means cache readOnly and autoCommit properties of connections managed by the pool	Default is true

Apache Commons™
http://commons.apache.org/

APACHECON 2014

BasicDataSource Configuration (new in v2)

Property	Meaning	Notes
evictionPolicyClassName	EvictionPolicy used by pool maintenance thread	Has no impact unless timeBetweeenEvictionRunsMillis is positive
jmxName	JMX name requested for the datasource	If the requested name is not valid or available, or is null an alternative is chosen
maxConnLifetimeMillis	The maximum lifetime of a connection in ms	Negative means unlimited, which is the default. Enforced when connections are borrowed, returned or visited by pool maintenance.

BasicDataSource Configuration (new in v2)

Property	Meaning	Notes
abandonedUsageTracking	True means the connection pool will (if possible) record a stack trace every time a connection is used	Useful debugging abandoned connection problems; adds overhead
lifo	True means the connection pool is a LIFO queue – the last returned connection is the first to be reused	Default is true
softMinEvictableIdleTimeMillis	The minimum amount of time a connection may sit idle in the pool before it is eligible for eviction, with the extra condition that at least minIdle connections remain in the pool	If both this property and minEvictableIdleTimeMilliis are positive, the default eviction policy evicts a connection if either one is satisfied

Configuration Example

DBCP Using BasicDataSource

Application code "leaks" connections on some exception paths

Database times out and closes connections after 60 minutes of inactivity

Load varies from non-existent (off hours) to 100 hits / second spikes during peak; ramp begins around 6AM local time, peaking around 11AM, diminishing around 2-3PM

Peak load can be handled (sustained) with 100 database connections

Configuration Considerations

Eliminating connection leaks is much better than relying on abandoned connection cleanup. Even with this configured, spikes in "leaky" execution paths will cause connection churn and pool exhaustion.

Setting testOnBorrow will ensure connections timed out on the server side are not returned to clients; testWhileIdle will remove these before they are checked out (and also keep them alive if frequent enough)

Configuration Example (cont.)

Simplest option

Assumptions:

- Connection leaks can be removed
- You can afford to allocate 100 database connections to the app

Configuration Settings:

maxTotal = 100

maxIdle = 100

testOnBorrow = true

testOnReturn = false

testWhileIdle = false

removeAbandoned = false

poolPreparedStatements = false

timeBetweenEvictionRunsMillis = -1

Set maxIdle to 50 to reduce connections reserved - cost is connection churn

validate connections when borrowed

Set to true and turn on maintenance to keep idle connections alive in the pool

If connection leaks cannot be closed, set removeAbandoned = true and configure timeout to be greater than longest running query

Configuration Example (cont.)

If leaks can’t be closed (or are FIX_LATER)

- Estimate peak incidence rate (how many per unit time)
- Estimate longest running query time
- If maxTotal / (peak incidence rate) < (max query time) you are SOL
- In fact, you need >> above to not be SOL
- If not SOL, configuring abandoned connection cleanup can help

Configuration Settings:

removeAbandonedOnBorrow = true

removeAbandonedTimeout > longest running query time (in seconds

Handling server-side connection timeouts

- Nothing you can do if clients check out and hold connections beyond server-side timeout (other than close them as abandoned)
- Three ways to handle preventing stale connections from being returned by getConnection()
  1. Set testOnBorrow = true
  2. Enable pool maintenance, set minEvictableIdleTimeMillis < server side timeout (in ms)
  3. Enable pool maintenance, set testWhileIdle = true and ensure connections are visited frequently enough to avoid timeOut
- Practical considerations
  - Once physical connections are closed on the server side, validation query may hang
  - When using options 2 or 3 above, make sure to set numTestsPerEvictionRun and timeBetweenEvictionRunsMillis so that connections are visited frequently enough

Conserving Pooled Resources

Trimming idle instance pool when load subsides

- Three ways to reduce "idleness"
  1. Set maxIdle < maxTotal
  2. Enable pool maintenance, set minEvictableIdleTimeMillis > 0
  3. Set maxTotal to a low number
- Practical considerations
  - Oscillating load and maxIdle << MaxTotal can lead to a lot of object churn
  - Running pool maintenance too frequently can lead to performance problems
  - If maintenance is enabled and minIdle is set too close to maxIdle, object churn will result
  - If instance creation is slow and load spikes are sudden and large, instance creation in a trimmed pool can cause performance problems

Be Realistic – *math is inescapable*

- If inter-arrival times and pooled object service are fairly stable, a pool with n instances can be viewed as an M/M/c queue
- There are widely available online calculators that you can use to estimate steady state mean client wait times, instance utilization rates, etc.
- Example: 10 instances, 200 requests / sec, 48 ms mean object hold time (pooled object service time) per request:
  - 151 ms average client service time
  - 96% instance utilization
  - What happens when object hold time > 50 ms?

Effect of pool performance (as long as it is not terrible) is usually swamped by provider latencies

Getting Involved

1. Subscribe to Commons-dev

http://commons.apache.org/mail-lists.html

2. Check out the code

http://commons.apache.org/subversion.html

3. Review open issues

http://commons.apache.org/issues.html

4. Talk about your ideas
5. Attach patches to JIRA tickets
6. THANKS!!