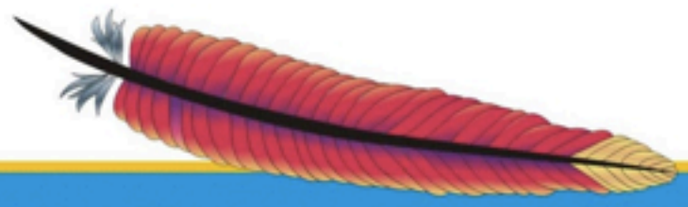
Apachecon

Commons Pool and DBCP

Phil Steitz

Apachecon US, 2010



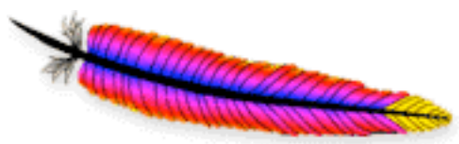
Leading the Wave
of Open Source

Agenda

- Pool and DBCP at 50,000 ft
- What configuration parameters mean
- Handling different kinds of workloads
- Development roadmap
- Getting involved

Pool and DBCP

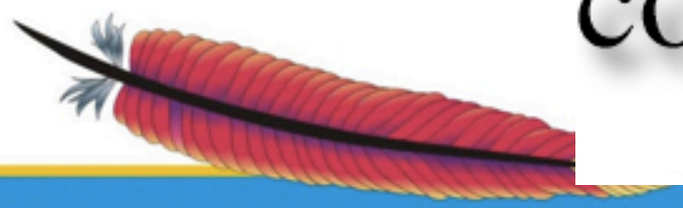
- Born around the same time as Commons itself (spring, 2001)
- DBCP provides the database connection pool for Tomcat
- Pool provides the underlying object pool for DBCP
  - **GenericObjectPool** Connections
  - **GenericKeyedObjectPool** Statements
- Current latest release versions (as of November, 2010)
  - Commons Pool 1.5.5
  - Commons DBCP 1.3 (JDBC 3) and DBCP 1.4 (JDBC 4)

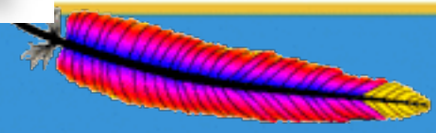


Apache Commons
http://commons.apache.org/

commons Pool

commons DBCP





Pool Features

- Simple object pool and instance factory interfaces (being “generified” in 2.0)
- Multiple pool implementations
- Most widely used impl is **GenericObjectPool**
  - really works as idle instance pool
  - configurable maintenance thread
  - maxActive, maxIdle, minIdle control
  - maxWait, whenExhaustedAction configurable
  - instance validation on borrow, return, while idle
  - LIFO / FIFO behavior configurable

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) and JDBC 4 (JDK 1.6)
  - DBCP 1.3.x implements JDBC 3
  - DBCP 1.4.x implements JDBC 4
  - Incompatibilities between JDBC 3 and 4 APIs have forced split in 1.x version sequence
- 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

GenericObjectPool Configuration

Property	Meaning	Default	Notes
maxActive	Maximum number of object instances in circulation (idle or checked out)	8	Negative means unlimited
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
maxWait	The maximum amount of time that borrowObject will wait for an instance to become available for checkout	Unlimited	Negative means unlimited Only meaningful if exhaustedAction is BLOCK
minIdle	The number of idle instances that the pool will try to keep available	0	Enforced only when pool maintenance thread runs Limited by maxActive

GenericObjectPool Configuration (cont.)

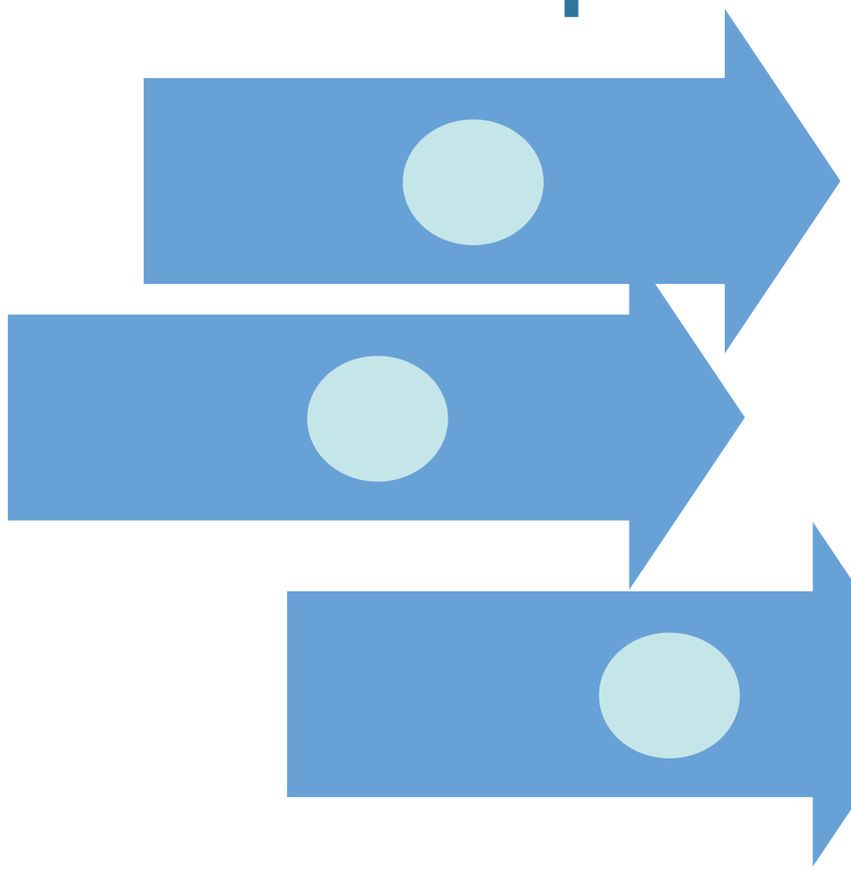
Property	Meaning	Default	Notes
whenExhaustedAction	Action to take when no instance is available to borrow	BLOCK	BLOCK enables maxWait
timeBetweenEvictionRunsMillis	Time between pool maintenance runs in milliseconds	never runs	idle instance eviction, minIdle, testWhileIdle require this > 0
minEvictableIdleTimeMillis	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	0	Enforced only when pool maintenance thread runs Superseded by minEvictableIdleTime
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.)

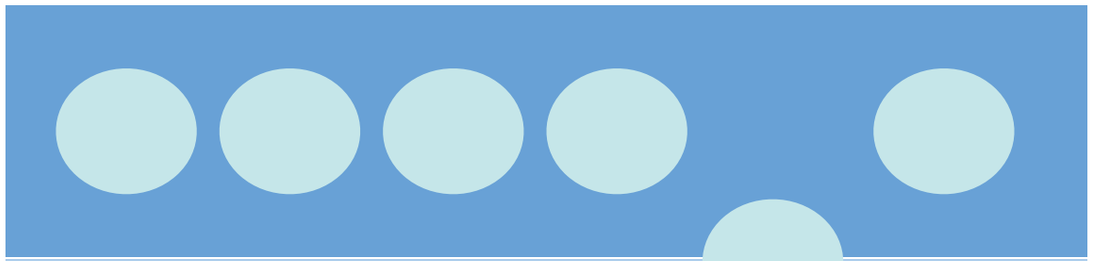
Property	Meaning	Default	Notes
testOnBorrow	Use the object factory's validate method to test instances retrieved from the pool	FALSE	Failing instances are destroyed; borrow is retried until pool is exhausted
testOnReturn	Validate instances before returning them to the pool	FALSE	Failing instances are 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 (i.e. timeBetweenEvictionRuns is positive)
lifo	Pool behaves as a LIFO queue	TRUE	FALSE means pool behaves as a LIFO queue

Simple Example

numActive = 3
numIdle = 6
means
maxActive >= 9



Threads using borrowed instances
numActive = 3



numIdle = 6
maxIdle >= 6

Instance being validated by maintenance thread
testWhileIdle = true, timeBetweenEvictionRuns > 0

Situations to Avoid

maxIdle << maxActive

If active count regularly grows to near maxActive, 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

BasicDataSource Configuration

Property	Meaning	Notes
maxActive, maxIdle, minIdle, maxWait, testOnBorrow, testOnReturn, testWhileIdle, timeBetweenEvictionRunsMillis, numTestsPerEvictionRun	Same meanings and defaults as pool	whenExhaustedAction is BLOCK
initialSize	Number of connections created and placed into the pool on initialization	Cannot exceed the maxActive pool property
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
removeAbandoned	True means the pool will try to identify and remove “abandoned” connections	Abandoned connection removal is triggered when a connection is requested and numIdle < 2 and numActive > maxActive - 3 Connections are considered abandoned if they have not been used (via JDBC operations) in more than removeAbandonedTimeout seconds
logAbandoned	True means stack traces for the code opening “abandoned” connections will be logged	
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	

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:

maxActive = 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 maxActive / (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:

removeAbandoned = true

removeAbandonedTimeout > longest running query time (in seconds)

NOTE:

Abandoned connection cleanup must be triggered by a getConnection() request

➡ *All threads can block until a new thread arrives to trigger cleanup (JIRA:* DBCP-260)

Configuration Example (cont.)

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

Two ways to reduce “idleness”

1. Set maxIdle < maxActive
2. Enable pool maintenance, set minEvictableIdleTimeMillis > 0

Practical considerations

- MaxIdle is enforced when instances are returned to the pool
- Oscillating load and maxIdle << maxActive 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 close to maxIdle, object churn will result
- If instance creation is slow and load spikes are sudden, new instance creation in a trimmed pool can cause performance problems

Simple Config Code

```
BasicDataSource ds = new BasicDataSource();

ds.setDriverClassName("com.mysql.jdbc.Driver");
ds.setUrl("jdbc:mysql:///test");
ds.setUsername("username");
ds.setPassword("password");
ds.setDefaultAutoCommit(false);

ds.setMaxActive(50);
ds.setMaxIdle(50);
ds.setMaxWait(10000);

ds.setTestWhileIdle(true);
ds.setTestOnBorrow(true);
ds.setTestOnReturn(false);
ds.setValidationQuery("SELECT 1");

ds.setTimeBetweenEvictionRunsMillis(1000 * 60 * 15);
ds.setMinEvictableIdleTimeMillis(1000 * 60 * 2);
ds.setNumTestsPerEvictionRun(10);

ds.setRemoveAbandoned(true);
ds.setRemoveAbandonedTimeout(60 * 60);
ds.setLogAbandoned(true);
```

Driver class must exist on classpath

All new connections created this way and reset to this value before being reused

ds.getConnection() will timeout and throw SQLException after 10 seconds

Connections that have not been used in more than one hour may be closed

“Manual” setup using PoolingDataSource

```
GenericObjectPool pool = new GenericObjectPool();
pool.setMaxActive(15);
pool.setMaxIdle(15);
pool.setMaxWait(10000);

Properties props = new Properties();
props.setProperty("user", "username");
props.setProperty("password", "password");
PoolableConnectionFactory factory =
    new PoolableConnectionFactory(
        new DriverConnectionFactory(new TesterDriver(),
                "com.mysql.jdbc.Driver", props),
        pool, null, "SELECT 1", true, true);
pool.setFactory(factory);
ds = new PoolingDataSource(pool);
```

Manually create GOP used to manage connections - can set properties not exposed by BasicDataSource this way

Manually create object factory and associate it with the pool - BasicDataSource does this on first getConnection() request

Simulations

Using **Commons Performance** (Commons "Sandbox" component)
http://commons.apache.org/sandbox/performance

Effect of maxIdle << maxActive under oscillating load

	No Idle Throttling	Idle Throttling
maxIdle	34	10
mean response latency	35.78 ms	49.58 ms
std dev response latency	5.19 ms	33.54 ms
mean instance idle time	34.07 ms	17.87 ms

maxActive	34
exhausted action	block, no timeout
pool maintenance	off
Instance validation	on borrow only
make latency	100 ms
destroy latency	0
validate latency	10 ms
client hold time	25 ms

client threads	100
min delay	100 ms
max delay	1 second
ramp type	linear
ramp period	500 ms
peak period	3 seconds
trough period	2 seconds
iterations	10000

Simulations (cont.)

Effect of Abandoned Connection Cleanup

	No abandonment	0.001 abandon rate	0.01 abandon rate
mean response latency	7.0 ms	7.57 ms	336.7 ms
std dev response latency	11.4 ms	12.90 ms	2461.9 ms

maxActive	15
maxIdle	15
exhausted action	block, no timeout
pool maintenance	off
instance validation	on borrow only
query type	text scan
client threads	90
peak load	4 requests / client / sec
minimum load	1 request / client / sec

client threads	90
min delay	250 ms
max delay	1 second
ramp type	linear
ramp period	20 seconds
peak period	100 seconds
trough period	20 seconds
iterations	10000

Simulations (cont.)

Effect of Old Pool version		
	Pool 1.5.5	Pool 1.3
mean response latency	35.78 ms	986.22 ms
std dev response latency	5.19 ms	3766.4 ms
mean instance idle time	34.07 ms	10.2 ms

maxActive	34
maxIdle	10
exhausted action	block, no timeout
pool maintenance	off
Instance validation	on borrow only
make latency	100 ms
destroy latency	0
validate latency	10 ms
client hold time	25 ms

client threads	100
min delay	100 ms
max delay	1 second
ramp type	linear
ramp period	500 ms
peak period	3 seconds
trough period	2 seconds
iterations	10000

Roadmap

Pool

Modernize Implementation

- Replace wait/notify with JDK 1.5 concurrency
- Integrate Tomcat jdbc-pool

Improve Operational Control

- Instance management
- Idle instance count management

Pool and DBCP

Modernize API

- Generification
- JMX
- Fix sins of the past

DBCP

Improve Robustness

- Server / Connection Failures
- Exception Management

Roadmap (cont.)

- 2.0 Versions will break backward compatibility and require JDK 1.5+
- 1.x releases will be bugfix only
- Code will be repackaged as org.apache.commons.pool2 / dbcp2
- API refactoring has begun in svn trunk
- Patches welcome!

Get Involved!

1. Subscribe to Commons-dev

   http://commons.apache.org/mail-lists.html
2. Check out the code

   http://commons.apache.org/svninfo.html
3. Review open issues

   http://commons.apache.org/pool/issue-tracking.html
4. Talk about your ideas
5. Attach patches to JIRA tickets

   http://commons.apache.org/patches.html
6. THANKS!!