Bending MySQL Connector/J to Your Liking

Mark Matthews
Database Technology Group - Sun Microsystems

JDBC is Nice, But...

Sometimes You'd Like to Do More

- Profiling
- Security
- Query Rewrite
- Caching
- Semi-transparent scale-out

Sometimes Vendor Extensions Aren't Enough

- Usually can alter state, configuration of a session
- Can't inject new behavior
- Can't participate in "lifecycle" events

Sometimes You'd Like to Change Behavior

We don’t always know what the users want...

Sometimes... We guess wrong!

MySQL is Open Source!

Hack on the JDBC driver?

What happens when the next version ships?

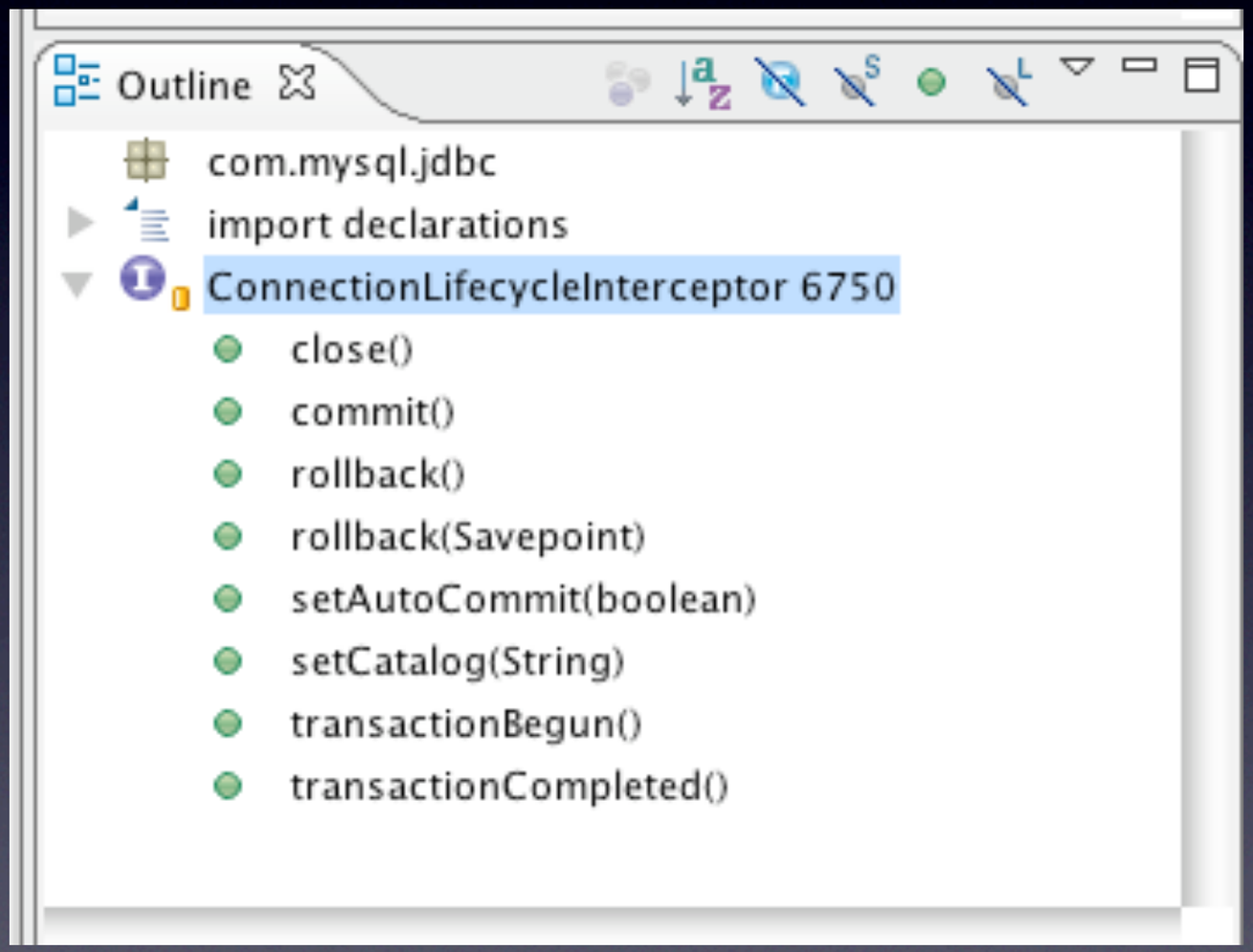
One Guess - Common Extension Points

Ho Hum Plugability

- Logging - pluggable
- Sockets - pluggable
- Profiler, usage advisor reporting - pluggable (but interesting to my team!)

Now, Something More Interesting

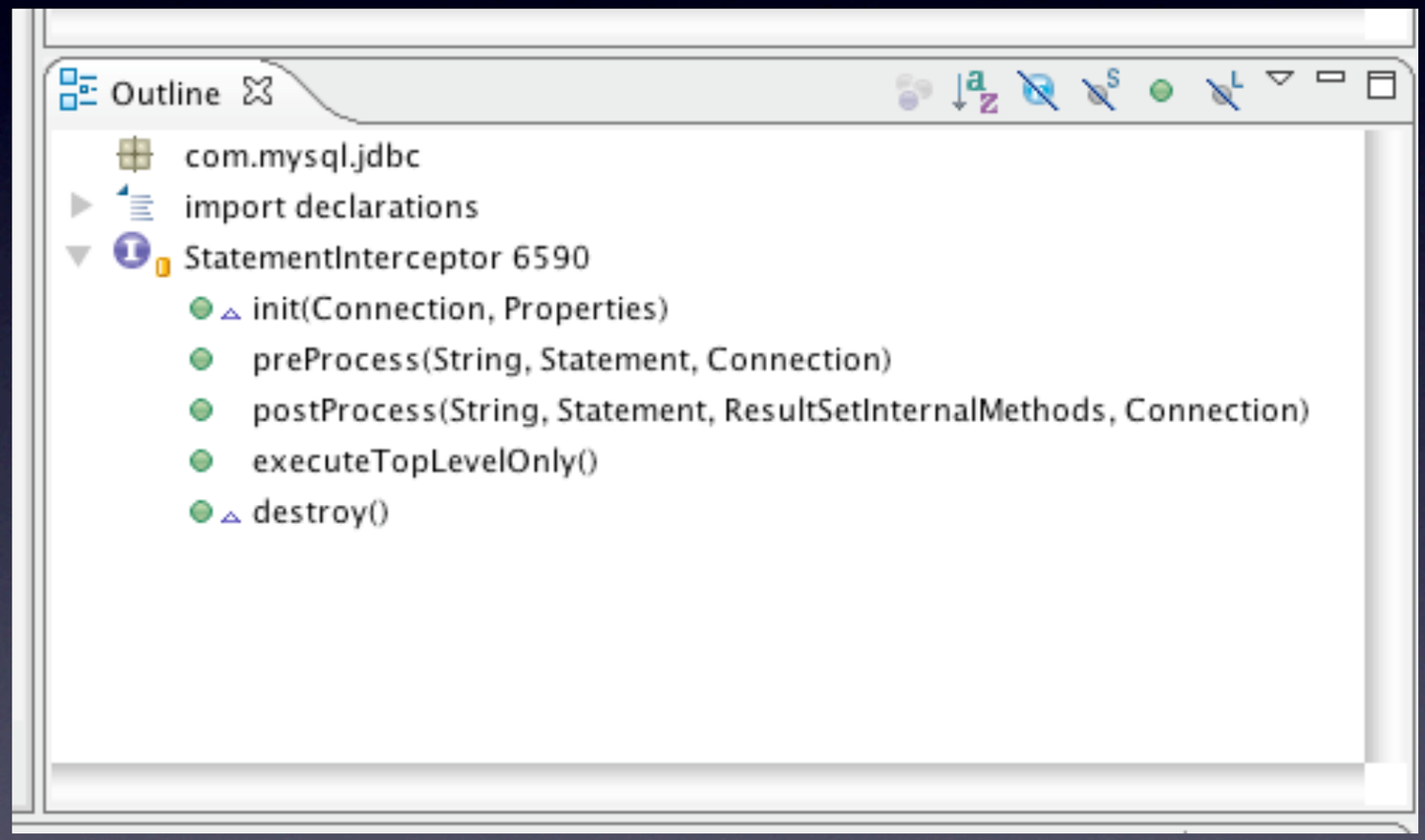
Connection Lifecycle



- Creation/ Destruction
- Auto commit state change
- Application transaction demarcation
- Transaction state change on server

Statement Interception

- Inspired by Jan's work on mysql-proxy
- Pre/Post execution
- Access to original statement
- Access to original statement parameters (if prepared)
- Early return of ResultSets
- Can be chained



Some Examples

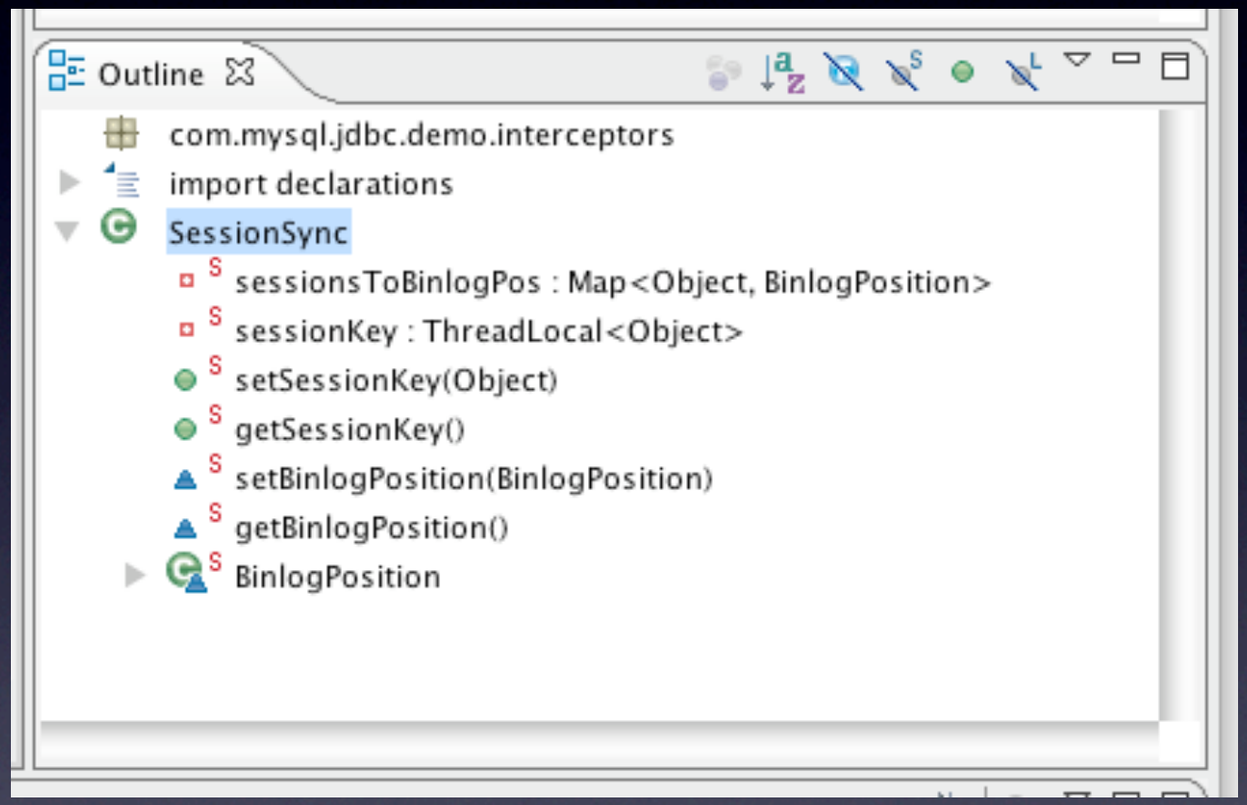
- Real-world Problems
- Thought experiments for the purpose of this talk
- Do you have more intriguing use cases?
- Will you share?

Use Lifecycle and Injection for Replication Consistency

Enabling Features

- MySQL's JDBC "ReplicationConnection"
  - jdbc:mysql:replication://master, slave1, slave2,slaveN.../
- For extra credit - mysql-proxy-based MySQL Enterprise Load Balancer with slave-awareness
  - jdbc:mysql:replication://master,load-balancer/
- Allows dynamic management of slaves

Multi Transaction Unit-of-Work



Opaque token...
HTTP session ID?
Current thread?

Set in business logic
Set in ServletFilter (similar to “open session in view”)

Finding Where to Wait

```
// We have to send the following query *after* commit on the master
// and we only get this callback for transactional storage engines
public boolean transactionCompleted() throws SQLException {
        if (!this.ownerConn.isReadOnly()) {
                // assume master
                ResultSet rs = this.ownerConn.createStatement().executeQuery(
                                "SHOW MASTER STATUS");

                try {
                        rs.next();

                        SessionSync.setBinlogPosition(new
                SessionSync.BinlogPosition(rs.getString(1), rs.getLong(2)));
                } finally {
                        if (rs != null) {
                                rs.close();
                        }
                }
        }

        return true;
}
```

Hurry Up and Wait...

```
public ResultSetInternalMethods preProcess(String sql,
        Statement interceptedStatement, Connection connection)
        throws SQLException {
    if (connection.isReadOnly() && !connection.getAutoCommit()
            && !this.hasWaited) {
        // assume slave

        PreparedStatement waitStmt = connection
                .prepareStatement("SELECT MASTER_POS_WAIT(?,?,?)");
        ResultSet rs = null;

        try {
            SessionSync.BinlogPosition binlogPosition = SessionSync
                    .getBinlogPosition();
            waitStmt.setString(1, binlogPosition.filename);
            waitStmt.setLong(2, binlogPosition.position);
            waitStmt.setInt(3, 30 /* timeout */);

            rs = waitStmt.executeQuery();

            rs.next();

            int waitResult = rs.getInt(1);

                // error checking....
        } finally {
            ...
        }
    }

    return null; // process the original statement as-is
}
```

Note, this code only runs when the query is being run against a slave, and the transaction has just begun.

Use Statement Interception for External Cache Coherency

Piggyback a magic token on the transaction

```
private static final String MEMCACHED_TOKEN = "/* memcached:";
private static final int TOKEN_OFFSET = MEMCACHED_TOKEN.length();

public ResultSetInternalMethods preProcess(String sql,
                Statement interceptedStatement, Connection connection)
                throws SQLException {
        // Simple example, where we just throw in a "plain" statement
        // with a special comment that becomes part of the transaction
        // e.g. "/* memcached: my_cached_object_key */ SELECT 1".
        //
        // One could also imagine inspectng the paramters of prepared
        // statements and picking off keys for memcached...

        if (sql.startsWith(MEMCACHED_TOKEN)) {
                int endIndex = sql.indexOf("*/", TOKEN_OFFSET);

                if (endIndex != -1) {
                        this.evictKey = sql.substring(TOKEN_OFFSET, endIndex).trim();
                }
        }

        return null;
}
```

Queue for eviction on DML success

```
    public ResultSetInternalMethods postProcess(String sql,
            Statement interceptedStatement,
            ResultSetInternalMethods originalResultSet, Connection
connection)
            throws SQLException {
        if (originalResultSet.getUpdateCount() > 0) {
            if (this.evictKey != null) {
                connectionLocal.get(connection).evict(this.evictKey);

                this.evictKey = null;
            }
        }

        return null;
    }
```

Evict on txn completion

```
private boolean didCommit = false;
private boolean didRollback = false;

public boolean transactionCompleted() throws SQLException {
        if (!this.didRollback && this.didCommit) {
                connectionLocal.get(this.myConnection).transactionCompleted();
        }

        return true;
}

public boolean transactionBegun() throws SQLException {
        this.didCommit = false;
        this.didRollback = false;

        return true;
}

public boolean commit() throws SQLException {
        this.didCommit = true;

        return true;
}

public boolean rollback() throws SQLException {
        this.didRollback = true;

        return true;
}
```

Wiring it All Together

```
class CacheSynchronizer {
	MemCachedClient memcached = null;
	SockIOPool pool = null;

	List<String> evictions = new LinkedList<String>();

	CacheSynchronizer(String sockIoPoolName, Properties props) {
		...

		memcached = new MemCachedClient(sockIoPoolName);
	}

	void evict(String key) {
		evictions.add(key);
	}

	void transactionCompleted() {
		for (String key : this.evictions) {
			// TODO: Handle failure?

			this.memcached.delete(key);
		}

		evictions.clear();
	}

	void destroy() {
		this.pool.shutDown();
	}
}
```

Use Statement Interception to Prevent Database Extrusion

Prevent get*() Via Regex

```
public ResultSetInternalMethods postProcess(String sql, Statement interceptedStatement,
        ResultSetInternalMethods originalResultSet, Connection connection)
        throws SQLException {

    // requirement of anonymous class
    final ResultSetInternalMethods finalResultSet = originalResultSet;

    return (ResultSetInternalMethods)Proxy.newProxyInstance(originalResultSet.getClass().getClassLoader(),
            new Class[] {ResultSetInternalMethods.class},
            new InvocationHandler() {

                public Object invoke(Object proxy, Method method,
                        Object[] args) throws Throwable {

                    Object invocationResult = method.invoke(finalResultSet, args);

                    String methodName = method.getName();

                    if (invocationResult != null && invocationResult instanceof String
                            || "getString".equals(methodName)
                            || "getObject".equals(methodName)
                            || "getObjectStoredProc".equals(methodName)) {
                        Matcher matcher = regexP.matcher(invocationResult.toString());

                        if (matcher.matches()) {
                            throw new SQLException("value disallowed by filter");
                        }
                    }

                    return invocationResult;
            }});

}
```

Horizonal Partitioning with Proxy

Use statement interception to inject...

```
private String lastShardKey = null;

private PreparedStatement variableStmt = null;

private static ThreadLocal<String> threadLocalKey = new ThreadLocal<String>();

public static void setShardKey(String key) {
      threadLocalKey.set(key);
}

public ResultSetInternalMethods preProcess(String sql,
            Statement interceptedStatement, Connection connection)
            throws SQLException {

      String candidateKey = threadLocalKey.get();

      if (lastShardKey == null || !lastShardKey.equals(candidateKey)) {
            this.variableStmt.setString(1, candidateKey);
            this.variableStmt.execute();
            this.lastShardKey = candidateKey;
      }

      return null; // process the original statement as-is
}
```

Port a black box application...

```
public ResultSetInternalMethods preProcess(String sql,
                Statement interceptedStatement, Connection connection)
                throws SQLException {
        if (interceptedStatement instanceof PreparedStatement) {
                String preparedSql = ((PreparedStatement) interceptedStatement)
                                .getPreparedSql();

                if (preparedSql.equals("SELECT TOP ? * FROM foo WHERE a=? AND b=?")) {
                        java.sql.PreparedStatement pstmt = connection
                                        .prepareStatement("SELECT * FROM foo WHERE a=? AND b=? LIMIT ?");
                        ParameterBindings bindings = ((PreparedStatement) interceptedStatement)
                                        .getParameterBindings();
                        pstmt.setObject(1, bindings.getObject(2));
                        pstmt.setObject(2, bindings.getObject(3));
                        pstmt.setObject(3, bindings.getObject(1));

                        return (ResultSetInternalMethods) pstmt.executeQuery();
                }
        }

        return null;
}
```

Evil Testing

How well does your application...

- Handle latency in query processing
  - Implement a random-latency SocketInputStream
- Handle transaction completion failure
  - Hook rollback/commit and throw 08007

Dogfood Time...

Browse Queries : MySQL Enterprise Dashboard

http://localhost:8080/m2trunk/BrowseQueries.action

Google

Dashboard [Hudson] Apple Yahoo! Google Maps YouTube Wikipedia News (496) Popular ET:Main - My...tranet Wiki Bugs & Statistics

MySQL Sun Enterprise Dashboard

Help Log Out

Servers

All Servers (4)

- localhost
- repo-prod
- repo-stage
- repo-test

Monitor | Advisors | Events | Query Analysis | Graphs | Replication | Settings

Browse Queries [1 to 11 of 11]

Query Search | Time Display: Interval | Hours: 03 | Minutes: 00 | View: Group | Query Type: All | Limit: 20 | filter | reset

Query	Total Execution	Num Executions	Max Execution	Avg Execution	Total Rows	Max Rows	Avg Rows	
	6,694 ms	5,720	345 ms	1 ms	0	0	0	alias
	6,694 ms	5,720	345 ms	1 ms	0	0	0	
=? (1)	4,508 ms	5,569	87 ms	1 ms	46,590	13	8	alias
(1)	1,750 ms	1,281	45 ms	1 ms	650	1	1	alias
	1,557 ms	2,962	71 ms	1 ms	0	0	0	alias
	753 ms	2,113	21 ms	0 ms	0	0	0	alias
	100 ms	48	29 ms	2 ms	0	0	0	alias
)	28 ms	60	6 ms	0 ms	60	1	1	alias
(1)	14 ms	60	2 ms	0 ms	0	0	0	alias
	8 ms	1	8 ms	8 ms	0	0	0	alias
	1 ms	1	1 ms	1 ms	0	0	0	alias
=? (1)	0 ms	1	0 ms	0 ms	0	0	0	alias

[1 to 11 of 11]

Summary Details | Example Details | Example Explain

Query Text

truncated | full | formatted

```
select types0_.namespace_id as namespace3_1_, types0_.type_id as
type1_1_, types0_.type_id as type...types0_.namespace_id as
namespace3_69_0_ from inventory_types types0_ where
types0_.namespace_id=3
```

Execution Time
6 ms

Date
Apr 15, 2008 6:37:14 PM

User
merlin@localhost

Thread ID
10204

From Host
mark-matthewss-macbook-pro.local/10.10.27.248

To Host
/127.0.0.1

Comments
From HibType.java:230

hide

/pdate Service | Knowledge Base | Technical Support | About

Logged in as "admin" (Apr 15, 2008 6:38:43 PM)
Monitoring 4 instances on 3 hosts (8,997 hosts remaining). Subscription is up-to-date. More info....

Why, oh why?

```
public void passivateObject(Object obj) throws Exception {
    if(obj instanceof Connection) {
        Connection conn = (Connection)obj;
        if(!conn.getAutoCommit() && !conn.isReadOnly()) {
            conn.rollback();
        }
        conn.clearWarnings();
        conn.setAutoCommit(true);
    }
    if(obj instanceof DelegatingConnection) {
        ((DelegatingConnection)obj).passivate();
    }
}
```

What do I do now????

- Fork DBCP?
- Submit a patch? When will it get in?
- Patch the JDBC Driver?

Aha!

```
public boolean setAutoCommit(boolean flag) throws SQLException {
        if (!flag) {
            return true;
        }

        return false;
    }
```

- Autocommit toggling 10% or more of total execution time
- 2nd largest total execution time in our application
- Fixed with 4 lines of code and a configuration parameter - no patches to core code

Questions?

More Information

- http://forge.mysql.com/
- java@lists.mysql.com
- mark@mysql.com