JavaOne^{SM}

Sun's Worldwide Java Developer Conference

An Overview of JDBC™

Graham Hamilton

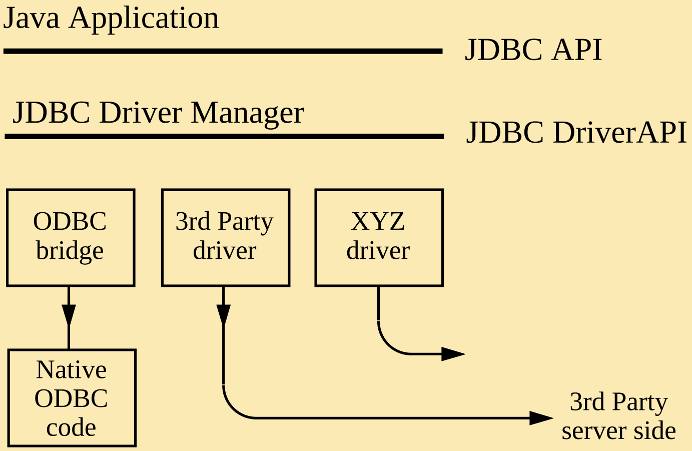
Talk Outline

- Overview of JDBC and its design goals
- Key API classes
- Some examples

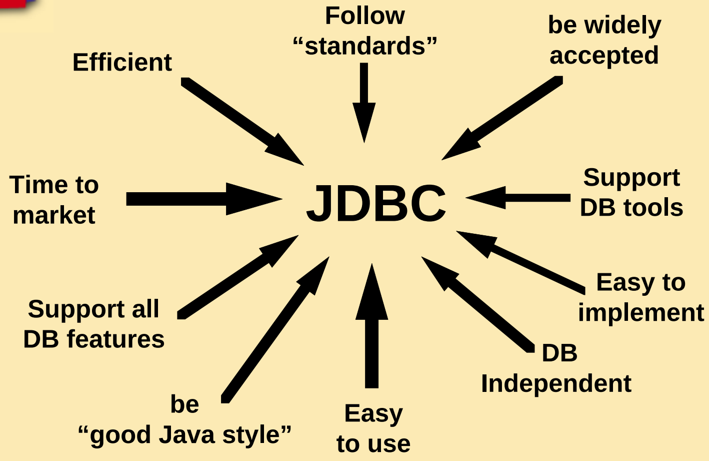
So What Is JDBC?

- ***JDBC is a Java™ API for executing SQL statements***
- It’s deliberately a “low level” API
  - But it’s intended as a base for higher level APIs
  - And for application builder tools
- It’s influenced by existing database APIs
  - Notably the XOPEN SQL CLI
  - And Microsoft’s ODBC

The JDBC Pieces



JDBC Pressures



What’s Good About ODBC:

- It is “adequate” for database access
- There is a lot of experience with it
- It is widely accepted
- It’s widely implemented:
  - For virtually all databases
  - On virtually all platforms
- People (now) know how to implement it efficiently

What’s Bad About ODBC:

- It’s hard to learn:
  - Simple & advanced features are mixed together
  - It has complex options even for simple queries
  - There is widespread use of “void *”
- It’s hard to map to Java:
  - Copious use of pointers
  - Frequent use of multiple results (via pointers)

JDBC Technical Goals

- Re-use key abstractions from ODBC
  - To ease acceptance/implementation by DB vendors
  - To ease learning by ISVs and application writers
- Provide a low-level SQL API
  - But we will add higher level APIs in the future
- Provide simple interfaces for simple tasks
- Support the weird stuff in separate interfaces
- Provide a “natural” and “clean” Java API
  - Test: JDBC applications should “read well”

Main JDBC Classes

- DriverManager
- Connection
- Statement
- PreparedStatement
- CallableStatement
- ResultSet
- ResultSetMetaData
- DatabaseMetaData

DriverManager

- The DriverManager tracks JDBC drivers
- JDBC drivers must register themselves
- DriverManager maps JDBC URLs to Drivers
- The DriverManager opens Connections
  - Taking a URL as the target
  - With a set of argument properties
  - The DriverManager selects a suitable driver

Database URLs

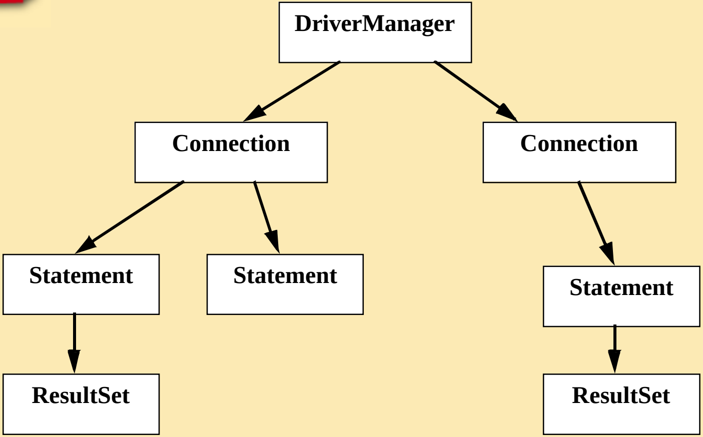
- We need a way to open JDBC connections:
  - For lots of different kinds of database drivers
  - Where different databases need different syntax
  - Without requiring human intervention!
- The answer seemed obvious: use URLs!
  - it’s the internet’s flexible naming scheme
  - you can bridge to other names (e.g. ODBC)
- Typical names use

  jdbc:<subprotocol>:<stuff>
  - e.g. jdbc:odbc:axx
  - or jdbc:odbcnet://wombat:344/fred
  - or jdbc:sybase://wombat:344/fred

Connection

- A Connection points at a given database
- A Connection provides Statement objects
- A Connection is a Transaction session.
  - You can implicitly begin transactions
  - And then commit or abort them
  - Or you can be in “auto commit” mode

Key Classes


DriverManager
Connection
Connection
Statement
Statement
Statement
ResultSet
ResultSet

Statement and PreparedStatement

- Statement allows simple SQL execution
  - “executeQuery” can be used for SELECT
  - “executeUpdate” can be used for other simple SQL
  - “execute” covers the weird cases

- PreparedStatement adds support for IN params
  - Through a collection of setXXX methods
  - It can be used for compiled SQL statements

CallableStatement

- CallableStatement extends PreparedStatement
  - For use with stored procedures
- It adds support for OUT parameters
- Unfortunately you have to register OUTs
  - Using “registerOutParameter”
- Then retrieve the value after call execution
  - Using one of the “getXXX” methods

ResultSet

- ResultSet provides results from a SELECT
- You can iterate over the rows using “next”
- Within a row you can retrieve result columns
  - using a set of “getXXX” methods
  - using either column names or column indexes
- For example:

```
ResultSet rs = ...
while (rs.next()) {
    int a = rs.getInt(“a”);
    Numeric b = rs.getNumeric(“b”);
    String key = rs.getString(3);
}
```

Programming with database metadata

- Humans are only one kind of user
  - And they may even be a minority
- Tools also do dynamic database programming
  - They talk to the database to learn the table layouts
  - They talk to the driver to find the DB features
  - Then they generate appropriate browsers/controls
- We support this with two classes:
  - ResultSetMetaData
  - DatabaseMetaData
- These are low-level APIs
  - “For expert use only”

ResultSetMetadata

- ResultSetMetadata describe a ResultSet
- It lets you find the column count
- For each column it provides:
  - the column name
  - the column’s SQL type
  - the column’s width
  - etc.
- This allows generic handling of ResultSets

DatabaseMetaData

- DatabaseMetaData describes a DB connection
- It provides information about feature details:
  - e.g. exact details of supported SQL
- It documents implementation limits:
  - e.g. the maximum number of columns in a table
- It describes the database schema:
  - the names and types of tables
  - the names and types of table columns
  - the names and types of stored procedures
  - etc.

Implementing a Driver

- Drivers can be implemented as:
  - Java bridges to native DB libraries
  - Clients using pure Java talking to database listener
- Drivers must register themselves with DriverManager
  - Best done in class static initialization code
- Drivers must implement the standard JDBC classes
  - Connection, Statement, ResultSet, etc.

Security Model

- JDBC follows the standard applet security model
- An applet can only connect back to its server
  - It can’t connect to random databases
- Drivers must conform to security model
  - It’s mostly automatic for pure Java drivers
  - Some checks are needed for native drivers
- Applications can connect to any server

A simple SELECT example

```
public void doSelect() throws SQLException {
   // Open a database connection.
   Connection con =
          DriverManager.getConnection("jdbc:odbc:wombat");

   // Create and execute a statement.
   Statement stmt = con.createStatement();
   ResultSet rs = stmt.executeQuery(
                        "SELECT a, b, key FROM Table1");

   // Step through the result rows.
   while (rs.next()) {
       // get the values from the current row:
       int a = rs.getInt(1);
       Numeric b = rs.getNumeric(“b”);
       String key = rs.getString(“key”);
       println(“a=” + a + “, b=” + b + “, key=” + key);
   }
}
```

A simple UPDATE example

```
public void doUpdate() throws SQLException {

   // Open a database connection.
   Connection con =
           DriverManager.getConnection("jdbc:odbc:wombat");

  // Create a “prepared” statement.
   PreparedStatement stmt = con.prepareStatement(
                  "UPDATE Table1 SET a = ? WHERE key = ?");

   // Now execute the statement with a couple of parameters
   stmt.setInt(1, 34);
   stmt.setString(2, "count");
   int rows = stmt.executeUpdate();

   System.out.println("Updated " + rows + " rows.");
}
```

The JDBC spec

- Its on-line in postscript and Acrobat.
- See our JDBC page at:
  - http://splash.javasoft.com/jdbc/
- We put it out for public review in March.
- We will freeze it on June 8th.

JavaOne℠

Sun's Worldwide Java Developer Conference