2008

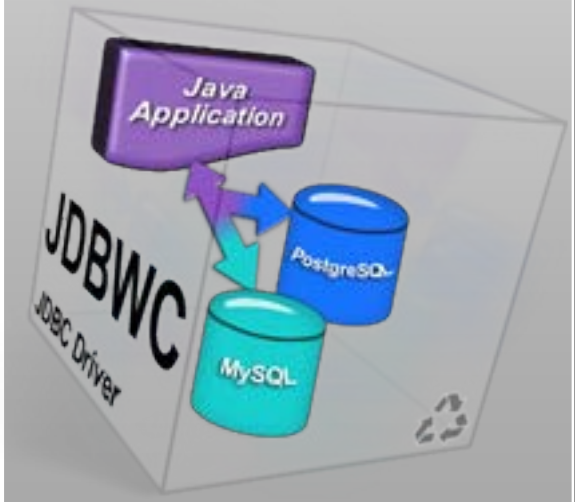
JDBWC

Open-Source Java JDBC Driver

JDBWC – JDBC Driver

Java to web-database connector (introduction)

Abstract: The JDBWC Driver connects a Java Desktop Application to a remote web-servers database using the servers native scripting language. The JDBWC Driver doesn't require a remote database to be visible through the servers firewall to access it. Currently the JDBWC Driver only supports PHP but can be ported to other scripting languages that support $_POST variables and local database interaction. The JDBWC Driver is a 2 part system. A JDBC Java Driver for the application and a scripting library for the server-side.



Tim Gall
Oz-DevWorX
7/9/2008

Table of Contents:

Introduction: Is JDBWC suitable for your Java Project?

First, if you're trying to find out if this driver is applicable to your JDBC requirements, I've included some scenarios below to highlight when the JDBWC should be considered as a possible solution. Even if you're just looking, it's probably good to read it anyway.

Question: My remote database is MySql; w*ouldn't it be better to use MySql-Connector/J?*

This ultimately depends on the location of your database/s, your server's capabilities and your JDBC SQL requirements.

The JDBWC Driver and MySql Connector/J are both **JDBC Drivers**, so they are used in the same way, using the same Java classes and methods.

MySql Connector/J communicates directly with a MySql server, requiring the MySql-server to be visible through firewalls.

JDBWC does not require the database to be visible through firewalls. JDBWC communicates with a MySql (or PostgreSql) database via an Apache web-server and PHP (or other scripting language with similar abilities to PHP).

***I recommend* MySql Connector/J when**:

Your database is on the same computer as the Java application or on a trusted private network or you are using some form of VPN-Tunnelling that allows you to keep the MySql server inaccessible to other forms of direct access. In these sorts of situations, you should use MySql Connector/J as it will provide better performance and increased functionality.

***I recommend JDBWC when**:*

Your database is on the internet or other un-trusted network and the server is not a Java Server. In these situations you should consider the JDBWC Driver as it will provide significantly increased security and extended batch handling abilities designed to assist with communicating across wide area networks of unknown quality where packets often get out of sync and in some cases lost entirely.

Alternately:

If you have a known number of clients accessing a database from fixed locations using dedicated IP's that will not change often, you could just modify the servers firewall settings to allow those IP's access to the database directly and use MySql Connector/J.

If the IP's are unknown (dynamic), then you would need to allow access to everyone, thus exposing the entire database engine to the internet. This situation is not usually recommended and if your server is on a shared-hosting account, it's often unlikely the server's admins would expose the server's database engine for you as it would put their other customers at risk. In this case you should consider using the JDBWC Driver.

About the JDBWC Driver:

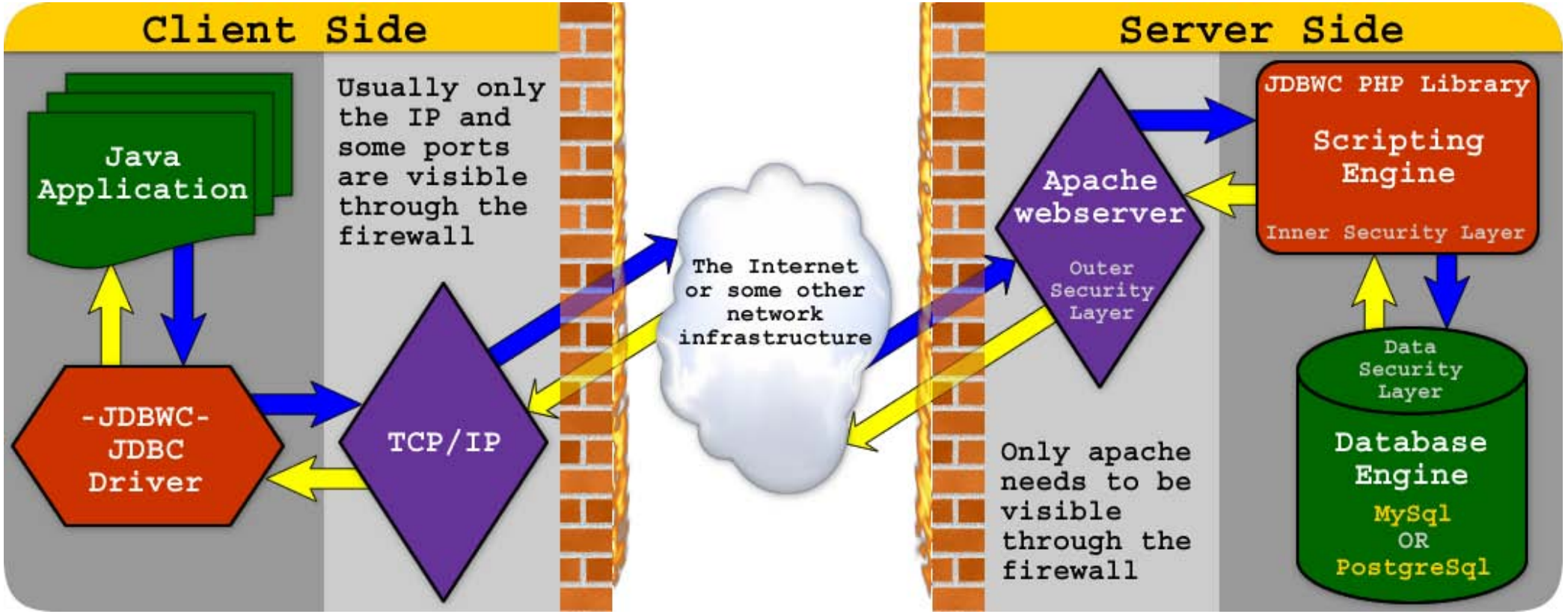
JDBWC is an acronym for **Java-Data-Base-Web-Connector**.

It's a JDBC Driver based on implementations of the standard java.sql interfaces.

By OZ-DevWorX 2008
For more information on JDBWC visit:
http://jdbwc.sourceforge.net/

The overall design and aims of this Driver are to allow [i]Java desktop applications to access remote databases without compromising the remote server's security.
This Driver is suitable for communicating with shared, virtual and dedicated web-server platforms and in most cases doesn't require any special settings or software to be installed on the remote server except the web-based JDBWC scripts. For best results, I recommend the server be running [ii]PHP-5xx with [iii]MySql-5xx (with the mysqli extension enabled) and or PostgreSQL >= 7.4.x.

The **JDBWC PHP Library** is basically a machine friendly website with heavy security restrictions.
The parent folder of the script-set (called: ***jdbwc***) should have apache folder security setup on it. The login credentials you use there will be required by your Java application to gain access to the main script-file (called: ***index.php***). Your Java application will also require database name/s, user/s and password/s so it can gain access to your database/s once it has accessed the jdbwc main script. Communications are best done using SSL for obvious reasons. The database names, users and passwords required by your Java application are always **transmitted to the remote server as a one-way-hash**, which is then verified on the server-side before access is granted to any of the servers databases.
In the diagram below, you can clearly see the 2 sides of the JDBWC Driver in red.
One part goes on the web-server; the other is used in your Java application using the standard mechanisms for working with JDBC connections.



JDBWC Driver (introduction)
Released as Open-Source

By OZ-DevWorX 2008
For more information on JDBWC visit:
http://jdbwc.sourceforge.net/

Connecting with JDBWC: *(versions <= 1.0.0.2_2beta)*

To make the Driver accessible from your application you only need two lines of code (shown in red below). This is the standard Java mechanism for creating a new JDBC Connection. The following code should be wrapped in `ClassNotFoundException` and `SQLException` catchers.

```
/* register the JDBWC Driver with the Java Runtime Engine */
Class.forName("com.jdbwc.core.Driver");

final String wcDataBase = "somedb_name";
final String wcUser = "someDbUserName";
final String wcPass = "someDbPassword";

final int port = 443;
final String httUrlStr = "https://subDomain-IfAny.yourDomain.ext/subFolder-IfAny/";
final String user = "Apache User";
final String pass = "Apache Password";

/* Valid Vendor IDs:
 * -----------------
 * USE DEFAULT DB : jdbc:jdbwc://
 * USE MYSQL DB   : jdbc:jdbwc:mysql//
 * USE POSTGRESQL : jdbc:jdbwc:postgresql//
 * -----------------
 * MySql is the default type.
 */
final String jdbwcUrlStr = "jdbc:jdbwc:mysql//"
      + httUrlStr
      + "?port=" + port
      + "&db_database=" + wcDataBase
      + "&db_user=" + wcUser
      + "&db_password=" + wcPass;

/* initialise a new JDBC Connection using the JDBWC Driver package */
Connection connection = DriverManager.getConnection(jdbwcUrlStr, user, pass);
```

jdbwcUrlStr contains the URL of the remote JDBWC library as well as some connection related information and database access credentials (The URL String is explained in more detail in the JDBWC installation and usage instructions).

user and **pass** contain the [iv]Apache credentials to allow access to the JDBWC PHP Library.

A few points of interest about the JDBWC Driver:

As you may have gathered, the server-side security is multilayered; this is because productive web-servers are usually exposed the internet and we really don't want anyone accessing our data unless they are authorized to.

So, step by step:

First the application needs to log in via Apache folder security to gain access to the JDBWC Script files. The application remains logged in until the JDBC Connection is closed.

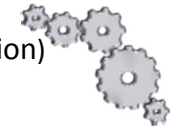
Second, the application needs to authenticate itself with the JDBWC script library before it can access the database server. Once authenticated, database access is restricted to the database that was selected during authentication until the JDBC Connection is closed.

Third, the JDBWC script library provides the actual database credentials (securely stored on the server-side) to the database server and logs into the specified database only when the application is communicating with the script or the server is building a response for the application. Application responses are compressed by the server and streamed back to the application for processing.

Lastly, if SSL is used for the Connection, your data will be secure during transit. It's highly recommended to use SSL connections.

The JDBWC Driver can return multiple ResultSets from a mixed batch query (normally this would trigger an exception). The JDBWC Driver can return multiple ResultSets from any execute, executeQuery and executeBatch method the JDBWC Driver supports. This is to help with data consistency in relational databases when using sequential SQL commands across wide area networks.

Currently the JDBWC Driver doesn't directly handle as many data-types as MySql Connector/J but should be sufficient for most applications. Since its default data type is an Object, most directly unsupported data types should be restorable to their real type manually by casting or parsing from an Object. Supported Data types will be expanded over time. JDBWC works with MySql and PostgreSql databases (further support may be added over time).



Optimal use of JDBWC and important operating notes:

One major point about communications across the internet is the ping time (or round trip time) each transmission requires. On a local machine or very fast local network it's likely to be 0ms, but across a wide area network it will take a little longer EG: 35ms is considered a fast response time for the internet.
So if your application exchanges information with the remote server 50 times, you need to multiply your round trip ping time by the number of accesses to get a rough approximate of the accumulative delay you will encounter. EG: 50 x 35ms = 175ms (175ms = 0.18 second). There is also the likelihood some packets will be damaged or lost and then resent by the Driver, increasing the cumulative delay by the average error rate of the internet connection.

To get the best performance out of your Java application in circumstances where you require continuous intensive database interaction and high performance, I recommend using a duplicate local mirror database for the general running of the application and have the application synchronise with your remote master database according to the schedule that suits you best. The mirror database doesn't need to be the same type of database as the master but should be very close syntactically to save you having to write customised SQL for each database.

EG: You could use MySql-Connector/J to exchange data with the local database for maximum performance and JDBWC to synchronise with the remote database. MySql retrieves and stores data in the local database, JDBWC retrieves and stores data in the remote database. This also gives your applications the ability to roam without requiring continuous internet access; this is particularly valuable for mobile (as in laptop) based applications (This is a common requirement from many modern businesses).