

Using Java with MySQL

[Architectures](#) [Install Java](#) [Install Tomcat](#) [Install Connector/J](#) [Install IDEs](#)
[JDBC models](#) [Client-side Connector/J](#) [Access to MySQL from servers](#)
[Queries, Statements and Resultsets](#) [JDBC 3 and connection pooling](#)
[Database-smart graphical controls and IDEs](#)

Some say Java is mainly C++ without the pointers. It is a decade younger than C++, two decades younger than C, and more widely known than either—because it is net-centric, it supports client-side and server-side development, it runs almost anywhere, it hides its pointers, the *Java Database Connectivity* (JDBC) API supports transparent database connections, and it enables multi-tier application architecture.

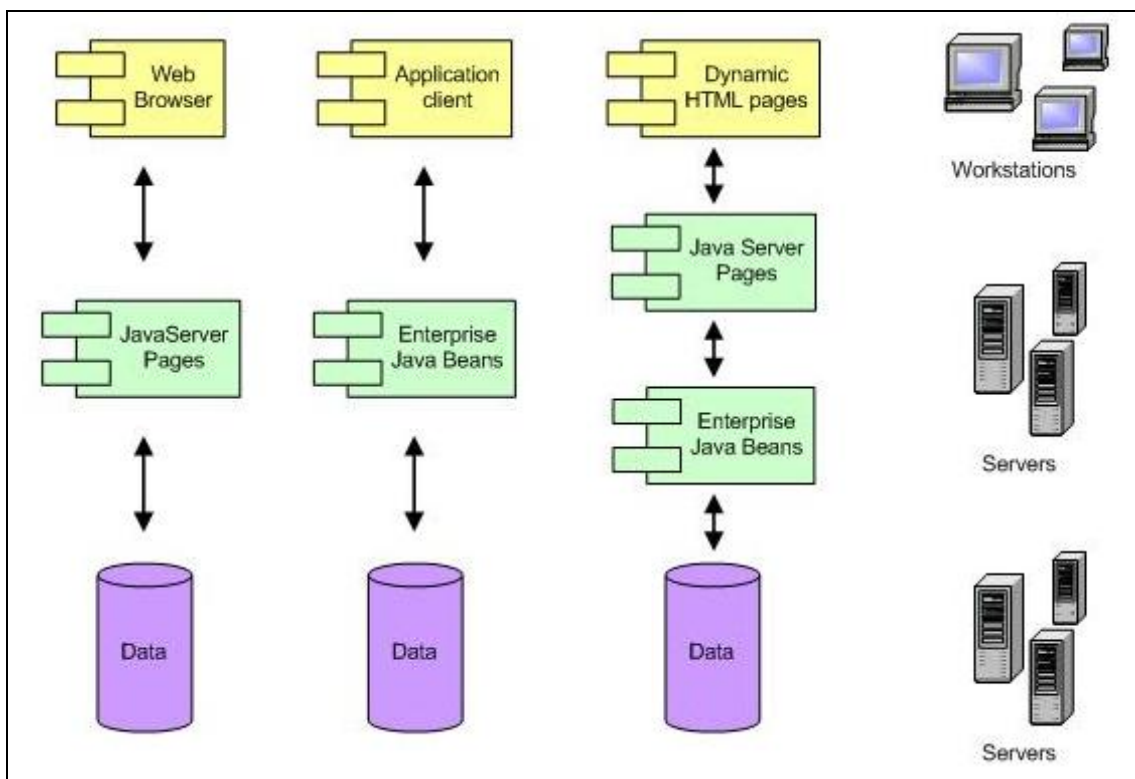


Fig 14-1: Some Java Application Models

A database-driven Java web application generally has *client*, *application server* and *database server* modules. It can be as modest as a few JavaServer Pages sharing a server with a database, or it can be a model-view-controller, multi-tier application distributed

across servers round the world. Application and database servers may be merged physically, but application architecture remains logically three-tiered.

Sun says Java is architecture-neutral, object-oriented, portable, distributed, high-performance, interpreted, multi-threaded, robust, dynamic, secure and simple. All these labels are plausible except the last. James Gosling's first "Oak" interpreter in summer 1991 may have been simple, but even the first Java SDK, released in 1996, had 212 interfaces and classes. By 2004 the enterprise-level product was 250MB unpacked. Now it's many times bigger still. Documentation adds hundreds more megabytes, equivalent to dozens of thick books. *Simple, Java isn't.* It's huge, verbose, and completely object-oriented, like a vast set of Russian dolls. Its learning curve is steep and long. There is no dashing off a wee four-line `Hello MySQL World!` server-side webapp just to get the feel of it.

This limits Java's usefulness for quick development of database-driven applications. In Chapter 12 we saw how to quickly write a robust, general-purpose, master-detail database browser with PHP. Can good non-trivial database applications be developed rapidly with Java and JDBC?

Architectures

The client side may consist of only HTML and a web browser—a *thin client*—or may contain compiled Java code in client-side web application modules. Server-side components may reside on any number of servers, often in server plug-ins called *containers*. One or more servers will host the database. Server software will have to include database drivers and their APIs.

On the client side or *tier*, the Java platform has two parts, the *Java Application Programming Interface* (API), and the *Java Virtual Machine* (JVM). From a `.java` source file, the Java compiler writes a `.class` file containing executable bytecodes. The JVM executes these bytecode sequences. The existence of a JVM for an operating system makes it possible for any machine running that OS to run any Java program. Whence the slogan, "write once, run everywhere", or as cynics say, "write once, debug everywhere".

Server tiers (Fig 14-1) may include web servers, application servers and database servers. Servers deliver Java modules to clients in various forms. When there are multiple database servers, data may be packaged for delivery to clients in *Enterprise JavaBeans* (EJB), in which case the beans generally reside on a *business tier*, the databases to which the EJBs relate likely reside in a separate *database tier*, and the *web tier* probably hosts small Java programs, *Java Server Pages* (JSPs) and servlets communicating with web clients, and optional clientside JavaBeans. Servlets and JSPs must run in HTTP plugin *servlet containers*. Sun maintains a list of available servlet containers [here](#).

A page received from the web tier can include an embedded applet—a small application written in Java, compiled by the Java compiler, and executing in the client's browser under control from the Web tier. It may also include JavaBeans.

When the client is an *application client*, it usually includes a graphical user interface (GUI) created from something like *Swing* or Java's *Abstract Window Toolkit* (AWT). It may open an HTTP connection to a *Servlet* or *JavaServer Page* (JSP) in the web tier, or it may directly access *JavaBeans* in the web tier.

A *thin client* does not directly execute business rules, query databases, or connect to legacy applications, these jobs being assigned to modules running on a J2EE server.

JDBC, the specification for Java database connectivity, has two parts:

- *java.sql*, eighteen basic interfaces and seven basic classes for accessing and processing data stored in a data source, usually a relational database, and
- *javax.sql*: 12 interfaces and two classes extending JDBC from client-side to server-side API, supporting connection pooling, distributed transactions, and an advanced data retrieval and update mechanism called *RowSets*.

There are four *types* of JDBC driver, only one of which, type 4, permits a client machine running only a browser and the JVM to call a DBMS directly. Type 4 drivers eliminate custom client code and middleware. The MySQL Type 4 JDBC driver is *Connector/J*.

Pieces of the puzzle

To *develop* MySQL-driven Java web applications, you need to get as many as seven moving (but free) parts working together:

- the *client-side Java bundle*, Java 2 Standard Edition (J2SE);
- the *server-side Java bundle*, Java 2 Enterprise Edition (J2EE);
- a *web server*, for example Apache (*Nix, Solaris, Windows), Internet Information Server (IIS, Windows), Sun Application Server (Linux/Unix, Solaris, Windows);
- for many web servers including Apache, you need a *Servlet container*, for example Tomcat;
- the *MySQL-Java JDBC driver*, Connector/J;
- for connection pooling in webapps you need either pooling resources from your application server or container, or a freestanding connection pool resource like *Jakarta Commons DBCP*;
- for all but the simplest projects, you need an *integrated development environment* like IBM's Eclipse or Sun's NetBeans.

The first two, J2SE and J2EE, are available in a free bundle from Sun. Installation of the third, Apache, is described in *Chapter 12*. For a reliable cross-platform servlet container, nothing beats the simplicity of Tomcat, which is also a standalone Java web server with an integrated copy of Jakarta Commons DBCP for connection pooling. So you can get away with downloading and installing just three bundles—J2SE/J2EE, Tomcat, and Connector/J.

Will a budget web hosting provider also serve Java applets and servlets? Till recently, the usual minimum webserver configuration for Java has been a Virtual Private Server, though that's begun to change.

To read the rest of this and other chapters, *buy a copy of the book*

M Cours.com