Java程序设计

2019春季

彭启民

pengqm@aliyun.com

数据库

- 数据库
  - 数据的集合，它由一个或多个表组成。每一个表中都存储了对一类对象的数据描述
- 关系数据库
  - 将数据表示为表的集合，通过建立简单表之间的关系来定义结构的一种数据库
  - 一个关系数据库通常包括一系列相互关连的表，在每一个表中存有一类与应用系统相关的数据。

关系数据库

- 关系型数据库可看作二维表的集合
  - 一个数据库由一个或多个表组成,每个表由行,列组成,列代表字段(即属性),行代表记录
    - 表中的记录是唯一的
    - 主关键字唯一

studentno	firstname	lastname	birthday	gender
090001	三	张	1985-6-3	M
090002	四	李	1986-8-20	F

数据库中表的结构由它们的 范式指出:通常说表属于第一、第二或第三范式……（1NF、2NF 、3NF、BCNF、4NF、5 NF)

第一范式(1NF)：表中的所有列无重复且不能再拆分(原子性),表中的每个表元应该只有一个值（永远不可能是一个数组）。

第二范式(2NF)：满足 1NF，并且每一个副键列完全依赖于主键列(主键和该行中的剩余表元之间是 1 对 1 的关系)。

第三范式(3NF)：满足 2NF，并且所有副键列是互相独立的,不能存在传递依赖(任何一个数据列中包含的值都不能从其他列的数据计算得到)。

NF太高需要大量连接查询，数据库的性能急剧下降，不可取。
现在几乎所有的数据库都是基于3NF创建的(这也表示通常数据库中都有相当多的表，每个表中的信息列相对较少)。

- 搜索(查找)
- 添加,插入,删除记录
- 更新纪录
- 投影和连接操作

- SQL语言
  - Sequence Query Language（结构化查询语言），是操纵数据库的标准语言。
  - 以X/open SQL Call Level Interface(调用层接口)做为API的基础，它也是Microsoft's ODBC（开放数据库互连）的基础
  - 非过程化语言
  - 统一语言
  - 关系数据库的公共语言

SQL语句

- 建立一个表

  create table 表名（字段名）;

```
CREATE TABLE STUDENT(
studentno CHAR(8) NOT NULL,
firstname CHAR(10) NOT NULL,
lastname CHAR(10) NOT NULL,
birthday DATE,
gender CHAR(1) DEFAULT ‘M’
);
```

- 删除表

  drop table 表名;

- SELECT[ALL|DISTINCT|DISTINCTROW|TOP] {*|talbe.*|[table.]field1[AS alias1][,[table.]field2[AS alias2][,…]]}
  FROM tableexpression[,…][IN externaldatabase]
  [WHERE…]

SELECT * FROM STUDENT WHERE StudentNo=’090001’;

查询学号为090001的学生

SELECT STUDENT.FirstName, STUDENT.LastName FROM STUDENT WHERE LastName LIKE 'A%';

查询所有姓名以A开头的学生的姓名

SELECT * FROM STUDENT ORDER BY StudentNo DESC;

将所有学生按学号顺序升序排列

SELECT * FROM STUDENT ORDER BY StudentNo ASC;将所有学生按学号顺序升序排列

- UPDATE 表名 SET 列名1＝表达式1，列名2＝表达式2，… WHERE 条件；

```
UPDATE STUDENT SET firstname=’老三’ WHERE
    StudentNO='090001';
```

将学号为090001的名字改为‘老三’

- DELETE子句的语法:
  DELETE [表名.*]
  FROM 来源表
  WHERE 准则
  - 不能删除不存在的数据

DELETE FROM STUDENT WHERE Studentno＝ ‘090001’;

- INSERT 子句的语法:
  INSETR INTO 目的表或查询(字段1,字段2,…)
  valueS(数值1,数值2,…)

  INSERT INTO STUDENT VALUES(‘090001’, ‘三’,’张’ ,’1982-2-23’,’M’);

JDBC

- Java Database Connectivity
  - JDBC是由java语言编写的，采用与供应商无关的标准，使得JDBC代码可在所有java平台上运行，这样使得程序的可移植性和安全性显著提高了。
  - JDBC 提供到多种 SQL 数据库的跨 DBMS 连接以及提供对其它表格式数据源（例如，电子表格或平面文件）的访问，可以从标准化 Java 编程语言虚拟访问任何表格式数据源。

- JDBC是为Java提供的一个平台无关的数据库标准API，它提供了一个通用的SQL数据库存取机制，该机制为多数关系型DBMS提供统一接口。
  - JDBC API，即一组JDBC类库 .
    - 在java.sql包中(JDK.1.1以上)，提供了核心的JDBC API，它包含了访问数据库所必须的类、接口和各种访问数据库异常类。
  - JDBC驱动程序
    - 对数据库透明, 不同的数据库制造商提供各自的驱动程序.

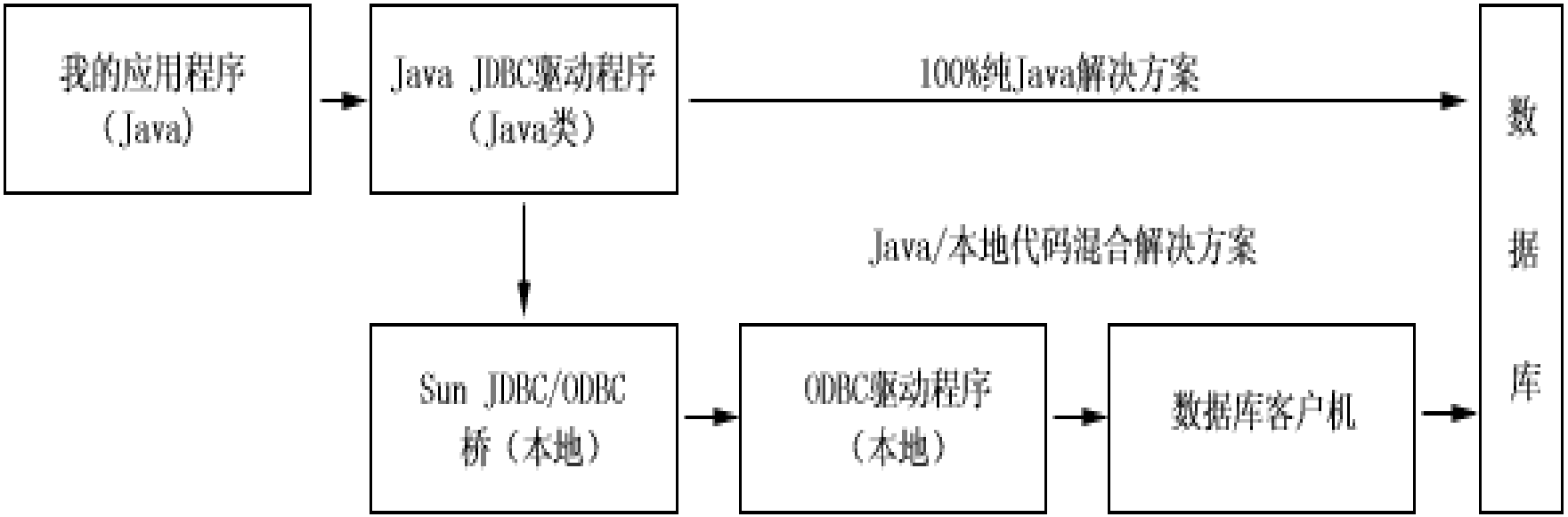
- JDBC产品
  - JavaSoft 框架
    - JDBC 驱动程序管理器
    - JDBC 驱动程序测试工具包
    - JDBC-ODBC 桥
  - JDBC 驱动程序的类型
    - JavaSoft 桥产品利用 ODBC 驱动程序提供 JDBC 访问。
    - 本地 API - 部份用 Java 来编写的驱动程序
    - JDBC 网络纯 Java 驱动程序
    - 本地协议纯 Java 驱动程序

JDK1.7开始，try-catch-finally 简化为 try-catch(编译时会转化为 try-catch-finally 语句。新的声明包含三部分：try-with-resources 声明、try 块、catch 块。它要求在 try-with-resources 声明中定义的变量实现了 AutoCloseable 接口，这样在系统可以自动调用它们的close方法，从而替代了finally中关闭资源的功能。

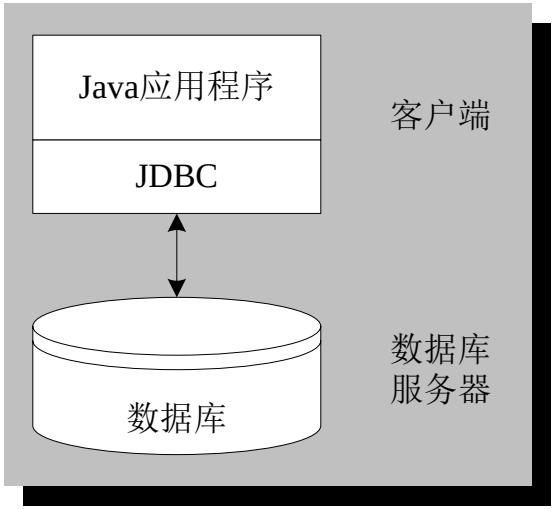
Java7.0 JDBC 4.1
支持使用 try-with-resources 语句进行自动的资源释放，包括连接、语句和结果集 支持 RowSet 1.1

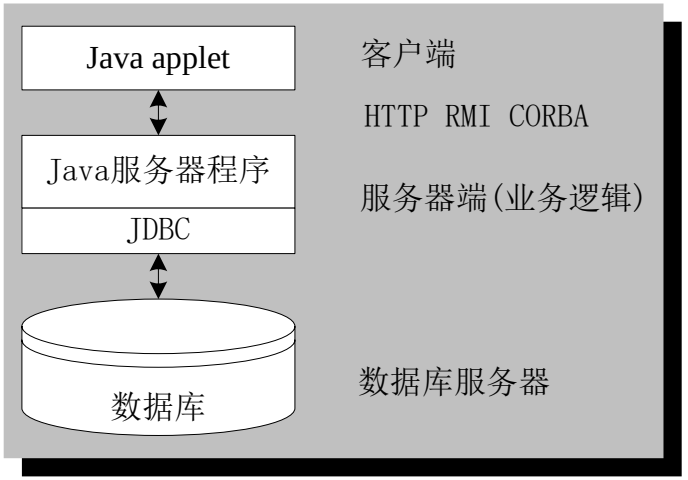
```
public static void main(String[] args) {
    try (FileInputStream inputStream = new
FileInputStream(new File("test"))) {
        System.out.println(inputStream.read());
    } catch (IOException e) { throw new
RuntimeException(e.getMessage(), e);
    }

    try(ResultSet rs=db.getResultSet()){
       //。。。。。。
    }
}
```

- JDBC和JDBC/ODBC解决方案



- Java应用程序即可以直接访问数据库，也可以将其作为中间层服务器应用程序。
  - 两层模型
  - 三层模型





ODBC

- **Open Database Connectivity** （开放式数据库互连）
- 利用ODBC API可通过统一界面和操作不同的数据库
- 四个组成部分**:**
  - 应用程序(Application)
  - ODBC 管理器(ODBC manager)
  - ODBC 驱动程序(ODBC Drivers)
  - 数据源(Data Sources,数据库)
- 访问数据库的模式

你的程序<--> ODBC管理器<--> ODBC驱动程序<--> 数据库

- ODBC管理器(Administrator)负责安装驱动程序，管理数据源，并帮助程序员跟踪ODBC的函数调用。
- 在ODBC中，应用程序不能直接存取数据库，它必须通过管理器和数据库交换信息。
- ODBC管理器负责将应用程序的SQL语句及其他信息传递给驱动程序，而驱动程序则负责将运行结果送回应用程序。
- ODBC管理器主对话框的主要内容是要求用户输入一个数据源，所谓数据源就是数据库位置、数据库类型以及ODBC驱动程序等信息的集成。
- 数据源负责将运行结果送回应用程序。应用程序、ODBC管理在使用之前必须通过ODBC管理器进行登记和连接，启动ODBC管理器后，选取Add按钮，根据自己的数据库类型，选择相应的ODBC驱动程序，然后输入数据源名（Data Source Name）和数据库文件名（Database Name），完成这些步骤后，以后的应用程序就能够通过ODBC管理器的数据源直接操纵数据库

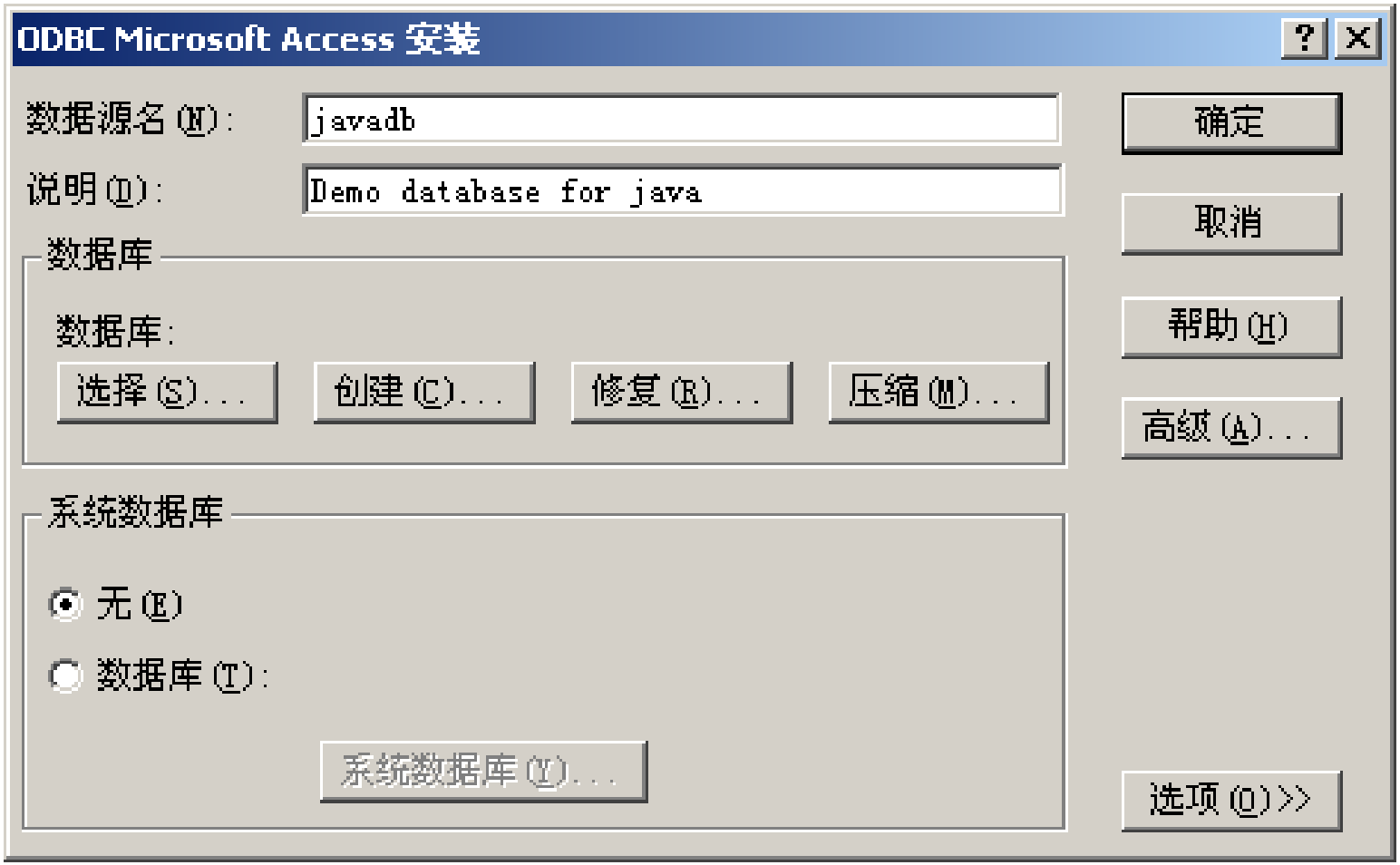
- ODBC的设置
- 在Windows资源管理器》控制面板》管理工具》ODBC数据源(可能操作系统的不同会有差别)。然后单击用户DSN页。


ODBC 数据源管理器
用户 DSN
系统 DSN
文件 DSN
驱动程序
跟踪
连接池
关于
用户数据源(U):
名称
驱动程序
dBASE Files
Microsoft dBase Driver (*.dbf)
DeluxeCD
Microsoft Access Driver (*.mdb
Excel Files
Microsoft Excel Driver (*.xls)
jsnw
SQL Server
MQIS
SQL Server
MS Access Database
Microsoft Access Driver (*.mdb
Visual FoxPro Database
Microsoft Visual FoxPro Driver
Visual FoxPro Tables
Microsoft Visual FoxPro Driver
添加(D)...
删除(R)
配置(C)...
ODBC 用户数据源存储了如何与指定数据提供者连接的信息。用户数据源只对当前用户可见，而且只能用于当前机器上。
确定
取消
应用(A)
帮助

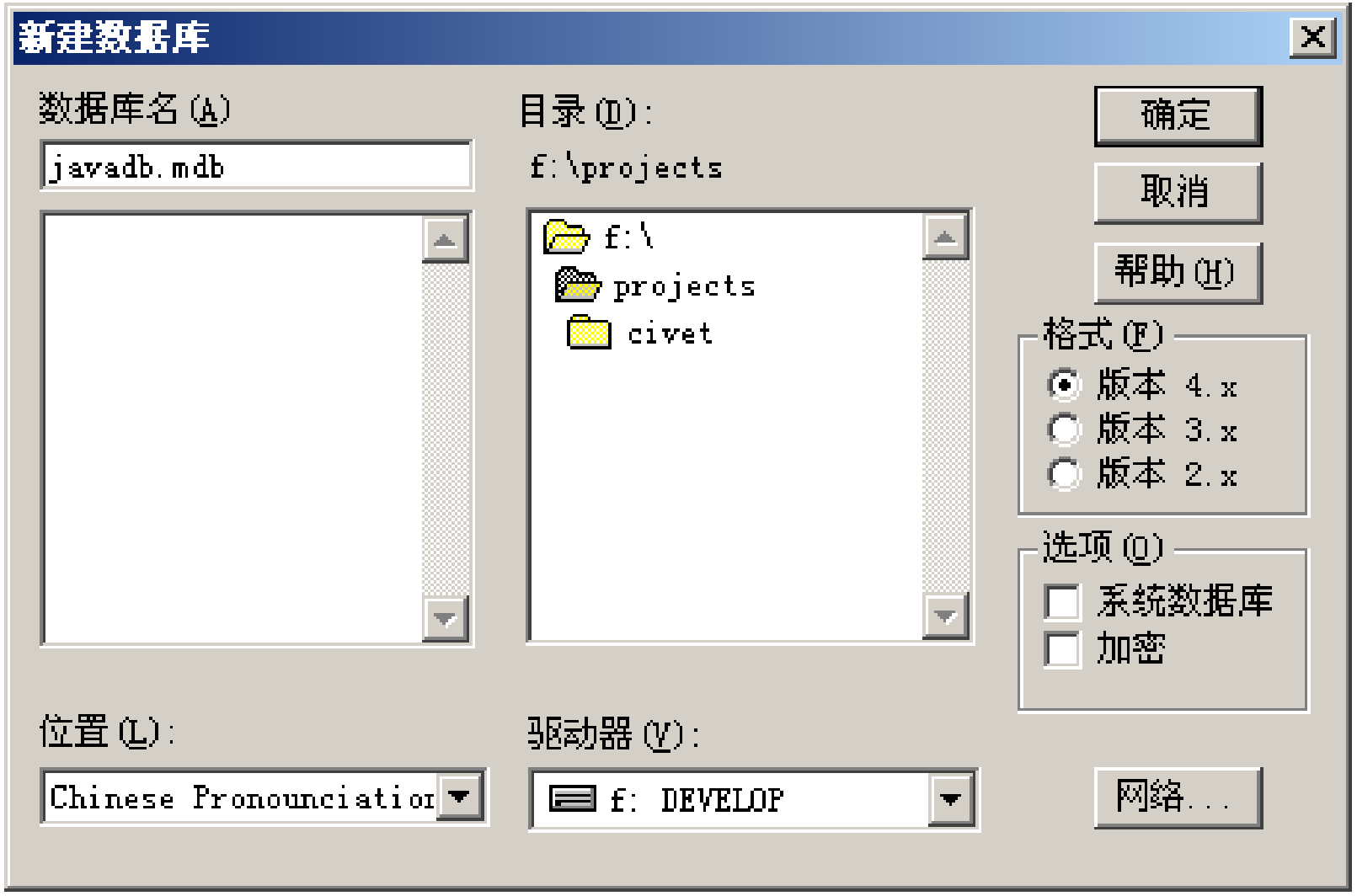
- 用户DSN：ODBC用户数据源，只对当前用户可见，而且只能用于当前机器上，但可配置局域网中另一台机器上的数据库。
- 系统DSN: ODBC系统数据源,对当前机器上的所有用户都是可见的.
- 文件DSN：ODBC文件数据源，可以由安装了相同驱动程序的用户共享。这是界于用户DSN和系统DSN之间的一种共享情况

- 驱动程序：这页列出了本机上所有安装的数据库驱动程序。里面列举了每个驱动程序的名称，版本，提供商公司，驱动程序文件名，以及安装日期
- 跟踪：ODBC跟踪允许创建调用ODBC的日志，提供给技术人员查看。里面可设定日志的路径和文件名。技术人员通过这里面的信息可以看到本机上所有的数据库访问的时间，用户，以及出错信息等情况。也可以通过这个辅助调试应用程序，可以启动Visual Studio的分析器，来进行ODBC的跟踪
- 连接池：连接池允许应用程序重用原来打开的的连接句柄，这样可以节省到服务器的往返过程
- 关于：列出了所有的ODBC的核心文件

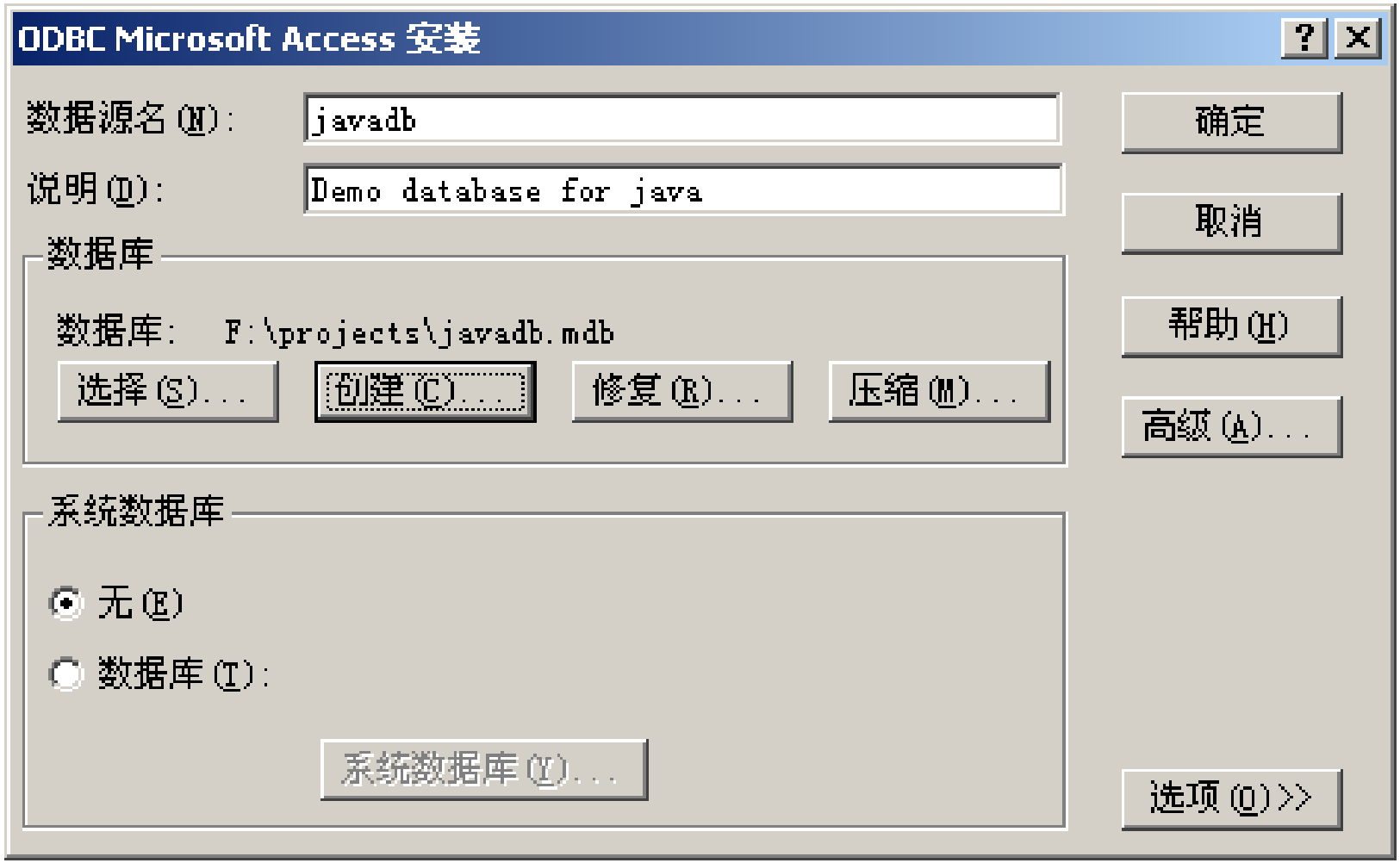

创建新数据源
选择您想为其安装数据源的驱动程序(S):
名称
Driver do Microsoft dBase (*.dbf)
Driver do Microsoft Excel(*.xls)
Driver do Microsoft Paradox (*.db)
Driver para o Microsoft Visual FoxPro
Microsoft Access Driver (*.mdb)
Microsoft Access-Treiber (*.mdb)
Microsoft dBase Driver (*.dbf)
Microsoft dBase VFP Driver (*.dbf)
Microsoft dBase-Treiber (*.dbf)
Microsoft Excel Driver (*.xls)
< 上一步(B)
完成
取消


ODBC Microsoft Access 安装
数据源名(N):
javadb
说明(D):
Demo database for java
确定
取消
帮助(H)
高级(A)...
数据库
数据库:
选择(S)...
创建(C)...
修复(R)...
压缩(M)...
系统数据库
无(E)
数据库(T):
系统数据库(Y)...
选项(O)>>

- 在高级按纽里面可以确定本数据源的访问密码，指定系统数据库
- 在选项按纽里面可以设定缓冲区的大小等。
  - 通过ODBC数据源的操作实际上都是和缓冲区里面的数据打交道。由应用程序正常退出的时候，或者需要别的应用程序也需要访问硬盘上的数据，驱动程序发现硬盘上的数据不是最新的数据，就用缓冲区的数据来更新硬盘上的数据。
    - 如果发生忽然断电的情况，硬盘上的数据来不及更新则会丢失数据。


新建数据库
数据库名(A)
javadb.mdb
目录(D):
f:\projects
f:\
projects
civet
确定
取消
帮助(H)
格式(F)
版本 4.x
版本 3.x
版本 2.x
选项(O)
系统数据库
加密
位置(L):
Chinese Pronounciatio
驱动器(V):
f: DEVELOP
网络...

- 远程访问数据库
  - 单击窗口右下角的"网络(N)..."按钮,去选择远程的数据库(MDB文件)
    - 前提条件是MDB文件已经共享出来了,而且你有权限去访问

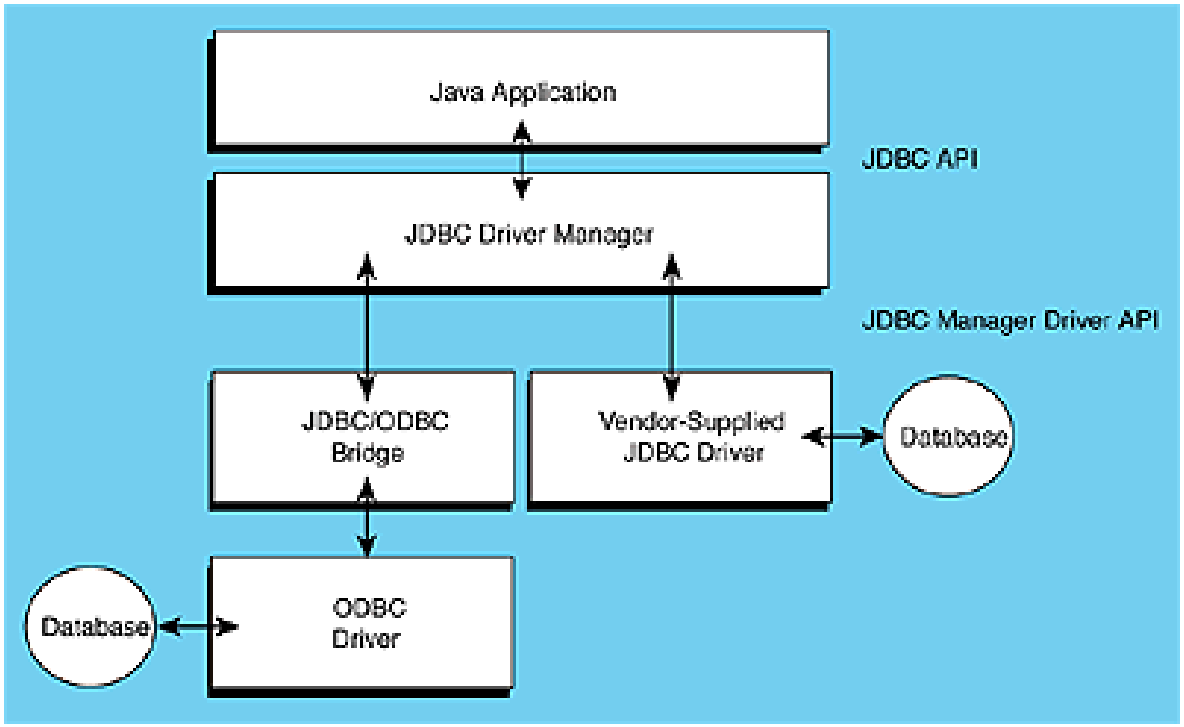

ODBC Microsoft Access 安装
数据源名(N):
javadb
说明(D):
Demo database for java
确定
取消
帮助(H)
高级(A)...
数据库
数据库: F:\projects\javadb.mdb
选择(S)...
创建(C)...
修复(R)...
压缩(M)...
系统数据库
无(E)
数据库(T):
系统数据库(Y)...
选项(O)>>

- 保存ODBC配置
  - ODBC 配置了很多的数据服务器，每次配都很烦，将配置保留下来，重新安装系统时拷贝过来就行了
  - 把注册表中的对应信息导出为.reg文件,重新安装系统后双击.reg文件重新导入注册表
  - 注册表中odbc配置信息的位置:[HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ODBC.INI\ODBC Data Sources]

两层模型和三层模型

当数据库和查询它的应用程序在同一台机器上，而且没有服务器代码的干预时，则将生成的程序称为 两层模型，一层是应用程序，而另一层是数据库(JDBC-ODBC 桥系统中通常是这种情况)。

当一个应用程序调用服务器，服务器再去调用数据库时，则称其为 三层模型(调用称为“服务器”的程序时通常是这种情况) 。



JDBC 由两层构成：第一层称为 JDBC API，其任务是将 SQL 命令传达到称为 JDBC Manager Driver API 的第二层。JDBC Manager Driver API与各种实际连接到数据库的第三方驱动程序通信，然后返回查询信息或者执行由查询指定的操作。

JDBC标准：
type1 jdbc-odbc桥形式进行连接，是作为jdk1.1后的一部分，当然也是sun.jdbc.odbc包的一部分连接形式：
application--->jdbc-odbc bridge(type1 jdbc driver)---->jdbc-odbc library--->odbc driver-->database
也就是说：应用程序通过JDBC-ODBC桥的形式对数据库进行连接的一种驱动类型，适用于快速的原型系统，没有提供jdbc驱动的数据库。如Access。
type2 java to native api
利用开发商提供的本地库来直接与数据库通信。
application--->jdbc driver(type2 jdbc driver)---->native database library---->database
也就是我们常说的直连，比type1性能略好。

type3 java to net
application--->jdbc driver(type3 jdbc driver)----->java middleware--->jdbc driver---->database
具有最大的灵活性，通常由那些非数据库厂商提供，是四种类型中最小的。
type4 java to native dababase
application--->jdbc driver(type4 jdbc driver)----->database engine--->database
最高的性能，通过自己的本地协议直接与数据库引擎通信，具备在internet装配的能力。

- **Driver类**
  - 用来与数据库通信
- **DriverManager类**
  - 是JDBC基础,用来管理和卸载JDBC驱动程序
    - **getConnection()**方法用于验证JDBC数据源,并返回Connection对象

- **Connection**类
  - **CreateStsatement()**方法连接JDBC数据源,返回Statement对象.
- **Statement**类
  - 将SQL行为封装起来交给数据库引擎
  - **execute()**方法, 返回resultSet对象.
  - **executeUpdate()**方法,执行SQL更新的命令,包括Create,Update,Delete,Insert等

Connection常用数据库操作方法：

Statement createStatement throws SQLException: 该方法返回一个Statement对象。
PreparedStatement prepareStatement(String sql) throws SQLException;该方法返回预编译的Statement对象，
　即将SQL语句提交到数据库进行预编译。
CallableStatement prepareCall(String sql) throws SQLException：该方法返回CallableStatement对象，
　该对象用于存储过程的调用。
上面的三个方法都是返回执行SQL语句的Statement对象，PreparedStatement、CallableStatement的对象是Statement的子类，
只有获得Statement之后才可以执行SQL语句。

Connection控制事务的方法：
Savepoint setSavepoint(): 创建一个保存点
Savepoint setSavepoint(String name)：创建一个带有名称的保存点
void setTransactionIsolation(int level):设置事务隔离级别
void rollback()：回滚事务
void rollback(Savepoint savepoint)：回滚到指定保存点
void setAutoCommit(boolean autoCommit): 关闭自动提交，打开事务
void commit()：提交事务

Statement

用于执行SQL语句的API接口，该对象可以执行DDL、DCL语句，也可以执行DML语句，还可以执行SQL查询语句，当执行查询语句是返回结果集，常用方法如下：

ResultSet executeQuery(String sql) throws SQLException：该方法用于执行查询语句，并返回查询结果对应的ResultSet对象，该方法只用于查询语句。

int executeUpdate(String sql) throws SQLException：该方法用于执行DML语句，并返回受影响的行数；该方法也可以执行DDL，执行DDL返回0；

boolean execute(String sql) throws SQLException：该方法可以执行任何SQL语句，如果执行后第一个结果是ResultSet对象，则返回true；如果执行后第一个结果为受影响的行数或没有任何结果，则返回false；

- **PreparedStatement类**
  - 预编译SQL语句
    - 如果每次执行的SQL语句只是改变WHERE子句中的值，然后重复发送到数据库执行。为了提高执行效率，这时可以考虑使用预编译SQL语句。使用预编译语句不需要数据库对相同的SQL语句进行分析和编译，即可直接运行。因此这种方法大大提高了执行效率。

- **ResultSet**类
  - 封装了一个由SQL查询返回的结果，可进行按行、栏的数据存取
  - **getString()**
  - **getInt()**
  - **next()**

ResultSet

void close() throws SQLException：释放、关闭ResultSet对象
boolean absolute(int row):将结果集移动到第几行，如果row是负数，则移动到倒数第几行。如果移动到的记录指针指向一条有效记录，则该方法返回true；
void beforeFisrt(): 将ResultSet的记录指针定位到首行之前，这是ResultSet结果集记录指针的初始状态：记录指针的起始位置位于第一行之前。
boolean first()：将ResultSet的记录指针定位到首行。如果移动后的记录指针指向一条有效记录，则该方法返回true。
boolean previous()：将ResultSet的记录指针定位到上一行，如果移动后的记录指针指向一条有效记录，则该方法返回true。
boolean next()：将ResultSet的记录指针定位到下一行。如果移动后的记录指针指向一条有效记录，则返回true。
boolean last()：将ResultSet的记录指针定位到最后一行。如果移动后的记录指针指向一条有效记录，则返回true。
void afterLast()：将ResultSet的记录指针定位到最后一行之后。
注意：在JDK1.4以前只支持next移动，且每次移动一个位置。到JDK1.5就可以随意定位。

ResultSet中的二进制Blob数据处理

如果要插入图片到数据库，不能直接设置SQL参数拼接字符串进行插入。因为二进制常量无法表示。

Blob类型通常用来存储文件，如：图片、音频、视频文件。将文件转换成二进制保存在数据库中，取出来的时候可以二进制数据恢复成文件。

但是将Blob类型数据插入到数据可以用PrepareStatement，通过PrepareStatement对象的setBinaryStatement
方法将参数传入到二进制输入流；也可以用Blob对象的getBytes方法直接取出数据。

利用ResultSetMetaData操作ResultSet结果集

在我们查询数据返回的结果集中，我们不清楚结果集存放的数据类型、数据列数。
那样我们就可以用ResultSetMetaData来读取ResultSet的信息。
通过ResultSet的getMetaData()的方法可以获取ResultSetMetaData对象。
然后可以用ResultSetMetaData对象的方法来操作ResultSet，常用方法如下：
int getColumnCount()：返回ResultSet的列名数量
int getColumnType(int column)：返回指定索引的类型
String getColumnName(int column)：返回指定索引的列名

java.sql.Driver

方法	说明
acceptURL()	返回一个boolen值，说明数据库驱动程序是否可以连接到指定的URL
connect()	建立数据库连接，返回应用程序中所用的Connection对象
getMajorVersion()	读取数据库驱动程序的主版本号
getMinorVersion()	读取数据库驱动程序的次版本号
getPropertyInfo()	利用当前的数据库驱动建立连接时，需要用户提供的基本属性（用户、口令等）
jdbcCompliant()	返回一个boolean值，说明当前Driver对象是否与JDBC兼容

java.sql.DriverManager

方法	说明
deregisterDriver()	从驱动器表中删除某个D river对象
getConnection()	建立连接
getDriver()	查找将要连接到URL上的Driver对象
getDrivers()	返回当前管理器中注册的所有Driver对象数组
getLoginTimout()	返回驱动器等待连接的时间(按秒计算)
println()	向当前日志流发送指定字符串
registerDriver()	在管理器中注册Driver对象
setLoginTimeout()	设置驱动器等待连接的最长时间(按秒计算)
setLogStream()	设置Driver对象的日志流

java.sql.Connection

方法	说明
close()	断开数据库连接
createStatement()	创建用于执行SQL语句的Statement对象
getCatalog()	返回包含当前数据库连接目录的字符串
getMetaData()	返回用于确定数据库特性的DataBaseMetaData对象
getTransactionIsolation()	返回与Connection对象相关的事务的当前隔离状态
isClosed()	判断是否已经断开连接
nativeSQL()	JDBC驱动器向数据库提交SQL语句,返回该语句
prepareStatement()	返回执行动态SQL语句的PrepareStatement对象
rollback()	回滚数据库事务

java.sql.Statement

方法	说明
close()	关闭当前的Statement对象
execute()	执行Statement对象,主要执行返回多个结果集的SQL语句
executeQuery()	执行SQL Select命令
executerUpdate()	执行SQL更新的命令,包括Update,Delete,Insert等
getMoreResults()	移到Statement对象的下一个结果处
getQueryTimeout()	返回JDBC驱动器等待Statement执行SQL的延迟时间(秒计算)
getResultSet()	返回查询结果集

java.sql.ResultSet

方法	说明
close()	关闭ResultSet对象
getBoolean()	将指定列名或列索引的数据作为Boolean类型变量返回
getInt()	将指定列名或列索引的数据作为Int类型变量返回
getString()	将指定列名或列索引的数据作为String类型变量返回
getDate()	将指定列名或列索引的数据作为java.sql.Date类型变量返回
getDouble()	将指定列名或列索引的数据作为Double类型变量返回
getObject()	将指定列名或列索引的数据作为Object类型变量返回
getMetaData()	得到结果集的元数据

- 数据库访问基本步骤
  - 安装JDBC驱动程序（预）
  - 加载JDBC驱动程序
  - 建立与数据库的连接
  - 执行SQL语句，处理结果集
  - 断开连接
  - 关闭数据库。

- 装载驱动程序：调用Class类的**forname（）**方法验证、装载Driver对象

  使用JDBC/ODBC桥驱动程序：

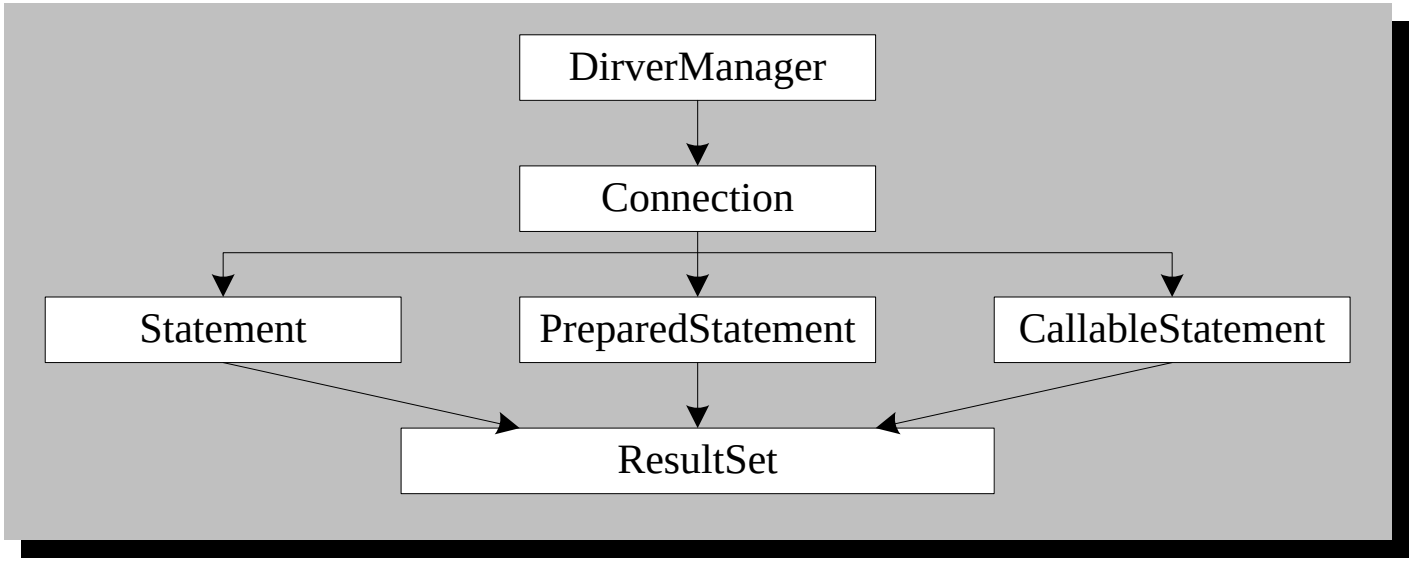
  Class.forName("jdbc.odbc.JdbcOdbcDriver");

- 建立连接：由驱动程序管理器（**DriverManager对象**）打开Driver对象，调用**getConnection（）**方法，以此验证包含JDBC数据源的地址是否有效，并**返回Connection**对象

  用适当的Driver类与DBMS建立一个连接：

  Connection con = DriverManager.getConnection(url,
  "Login", "Password");

- **Connection对象**建立应用程序与数据库的连接，并创建**Statement对象**
- **Statement对象**将SQL语句封装起来交给数据库引擎，执行SQL语句，得到结果集**ResultSet对象**
  - **ResultSet对象**封装了SQL返回结果



```
import java.sql.*;
public class JDBCExample {
    public static void main(String[] args){
        Connection myconnection;
        Statement mystatement;
        ResultSet myset;
        try {
                Class.forname("xxxxxx");//驱动程序名
        }
        catch(java.lang.ClassNotFoundException e) {}
```

```
try {
    myconnection=DriverManager.getConnection("jdbc:msql://主机名:端口/库名"); //"usrname","password"
    mystatement=myconnection.CreateStatement();
    mystatement.execute('select * from xxx');
    myset=mystatement.getResultSet();
    while(myset.next())
    {
        int xxx1=myset.getInt("字段1") ;
        String xxx2=myset.getString("字段2");
        ……
        输出结果;
    }
}
```

```
        catch(SQLException e) {……}
        }
        mystatement.close();
        myconnection.close();
    }
}
```

```
try {
    Class.forName(“jdbc.odbc.JdbcOdbcDriver”);//驱动程序名
    myconnection=DriverManager.getConnection("jdbc:odbc:test");
}
catch(java.lang.ClassNotFoundException e) {}
```

- jdbc:subprotocol_name:driver_dependant_databasename
  - MySQL
    - jdbc:mysql://localhost/name
  - Oracle thin driver
    - jdbc:oracle:thin:@machinename:1521:dbname

```
Class.forName("oracle.jdbc.OracleDriver").new
   Instance();
    String  dbURL="jdbc:oracle:thin:@yourser
   vername:1521:yoursid";
    String  user  =  "youruser";
    String  password  =  "yourpassword";
    Connection  c  =  null;
    c  =  DriverManager.getConnection(dbURL,
   user,password);
```

oracle.jdbc.OracleDriver是一个类，在oracle/jdbc目录下有一个OracleDriver.class的文件 。

Oracle8/8i/9i数据库（thin模式）

```
Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();

String url="jdbc:oracle:thin:@localhost:1521:orcl"; //orcl为数据库的SID

String user="test";

String password="test";

Connection conn= DriverManager.getConnection(url,user,password);
```

DB2数据库

Class.forName("com.ibm.db2.jdbc.app.DB2Driver ").newInstance();

String url="jdbc:db2://localhost:5000/sample"; //sample为你的数据库名

String user="admin";

String password="";

Connection conn= DriverManager.getConnection(url,user,password);

Sql Server7.0/2000数据库

```
Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver").newInstance();

String url="jdbc:microsoft:sqlserver://localhost:1433;DatabaseName=mydb";

//mydb为数据库

String user="sa";

String password="";

Connection conn= DriverManager.getConnection(url,user,password);
```

Sybase数据库

```
Class.forName("com.sybase.jdbc.SybDriver").newInstance();

String url =" jdbc:sybase:Tds:localhost:5007/myDB";//myDB为你的数据库名

Properties sysProps = System.getProperties();

SysProps.put("user","userid");

SysProps.put("password","user_password");

Connection conn= DriverManager.getConnection(url, SysProps);
```

Informix数据库

```
Class.forName("com.informix.jdbc.IfxDriver").newInstance();

String url = "jdbc:informix-
sqli://localhost:1533/myDB:INFORMIXSERVER=myserver;

user=testuser;password=testpassword"; //myDB为数据库名

Connection conn= DriverManager.getConnection(url);
```

MySQL数据库

```
Class.forName("org.gjt.mm.mysql.Driver").newInstance();

String url
="jdbc:mysql://localhost/myDB?user=soft&password=soft1234&us
eUnicode=true&characterEncoding=8859_1"

//myDB为数据库名

Connection conn= DriverManager.getConnection(url);
```

PostgreSQL数据库

```
Class.forName("org.postgresql.Driver").newInstance();

String url ="jdbc:postgresql://localhost/myDB" //myDB为数据库名

String user="myuser";

String password="mypassword";

Connection conn=
DriverManager.getConnection(url,user,password);
```

access数据库用ODBC直连

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver") ;

String url="jdbc:odbc:Driver={MicroSoft Access Driver
(*.mdb)};DBQ="+application.getRealPath("/Data/Demo.mdb");

Connection conn = DriverManager.getConnection(url,"","");

Statement stmtNew=conn.createStatement() ;
```

JDBC 访问 Excel 表

- 建立一个Excel文件 test.xls，编辑一张 worksheet , 命名为data
- 默认情况下，ODBC Driver把WorkSheet的名字作为数据库表的名字，WorkSheet的第一行的列名当成数据库表的字段名。
- 对所建的excel文件创建ODBC数据源XDb
- 准备SQL语句，使用数据库表名时，须在worksheet名字后加上$字符并放到[]中，如 [data$]
- 可以通过ODBC数据源名访问连接Excel 表，也可通过文件名访问连接Excel 表。

```
import java.sql.*;
public class ExcelAccess {
    String dbName = "sun.jdbc.odbc.JdbcOdbcDriver";
    String url = "jdbc:odbc:Test";
    Connection con;
    Statement stmt;
    ResultSet rs;
    public ExcelAccess()   {
        try {
            Class.forName(dbName);
         }  catch(ClassNotFoundException e){
             System.out.print("加载数据库驱动程序错误:");
         }
         try    {
            con = DriverManager.getConnection(url,"","");
            stmt=con.createStatement();
            String sql="select * from [data$]";
             rs=stmt.executeQuery(sql);
```

```
while(rs.next()){
                System.out.println(  rs.getString(1)+" “
                      + rs.getString(2)+" "+ rs.getString(3));
               }
      }  catch(Exception e1)   {
            System.out.println(e1.toString());
      }  finally  {
           try  {
                rs.close();   stmt.close();   con.close();
                rs=null;      stmt=null;    con=null;
            } catch(Exception e){}
       }
    }
   public static void main(String[] args)   {
        ExcelAccess ea = new ExcelAccess();
    }
}
```

通过文件名访问连接Excel 表

```
String strurl="jdbc:odbc:driver=        {Microsoft Excel
Driver (*.xls)};        DBQ=E:\\MyTestDB\\Excel\\Test.xls";

conn = DriverManager.getConnection(strurl);
```

Java DB

Java 6以后内嵌的袖珍数据库（在安装 JDK 时被安装安装于 javadb 目录下）。

纯 Java 实现、开源的数据库管理系统（DBMS），源于 Apache 软件基金会（ASF）名下的项目 Derby，功能齐备，支持几乎大部分的数据库应用所需要的特性，得到了包括 IBM 等大公司以及全世界优秀程序员们的支持。

Java DB是一个先进的全事务处理的基于Java技术的数据库，支持各类开放标准、触发器和存储程序。可以客户端服务器模式使用，也可以直接嵌入到一个Java应用程序中。

```
String driver = "org.apache.derby.jdbc.EmbeddedDriver";
String protocol = "jdbc:derby:mydb;create=true"; // 在工程目录下创建数据库
   "jdbc:derby:db/mydb;create=true";    //在工程目录下db目录中创建数据库
    “jdbc:derby:D:/test/mydb;create=true";    //在D:/test/目录下创建数据库
```

课后：了解Java DB的使用和开发知识。