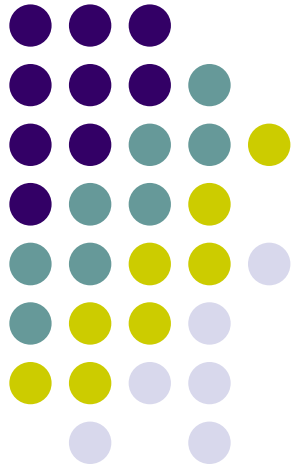
# 第八章 基于Java的数据库访问接口 JDBC



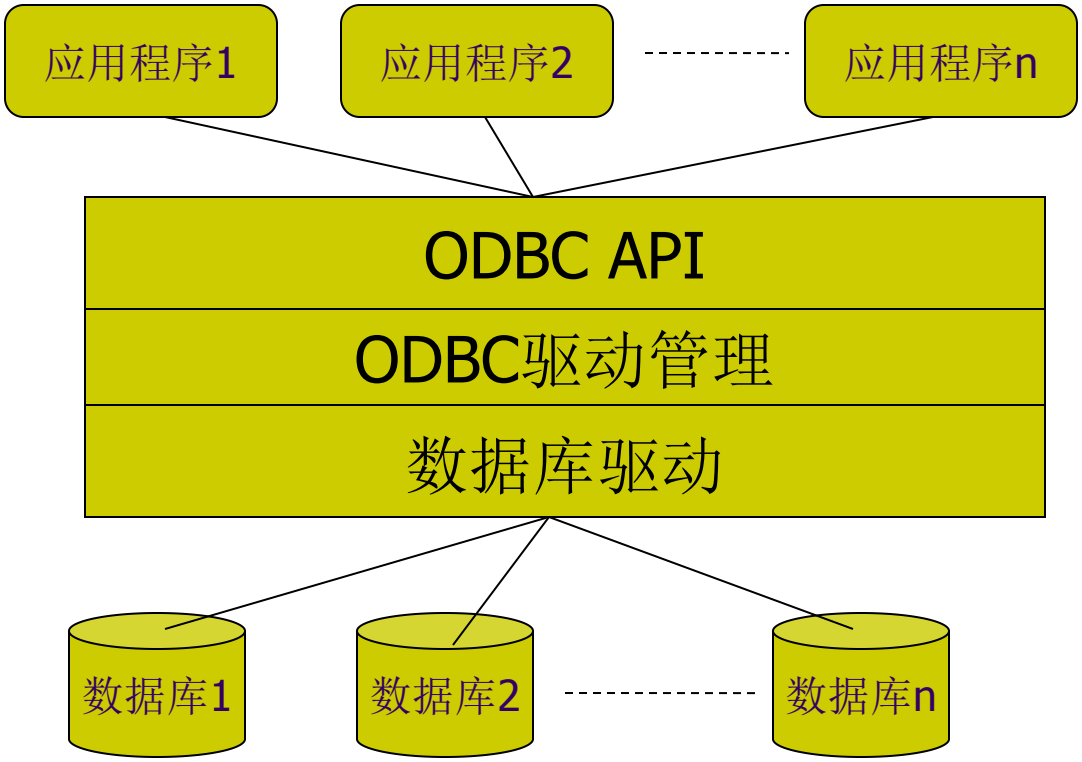
# 主要内容

- JDBC概述
- JDBC标准以及主要接口和类
- JDBC的体系结构
- JDBC程序设计
- JDBC程序示例

# JDBC概述

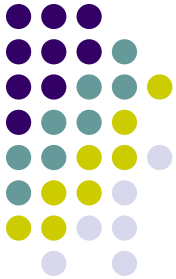
大家都知道，SQL(Structured Query Language,结构化查询语言)是一种用来管理和操纵数据库的ANSI标准语言。然而在各种数据库系统之间还有着细微的实现差别，因此我们需要用一个桥来将通用的数据库调用方式转换成各种数据库特殊的调用方式，最普遍的桥就是MS的ODBC(Open Database Connectivity,开方式数据库连接)。

**ODBC是用纯C语言开发的，用于访问多种格式的数据库的应用程序API，其体系结构如图所示:**



ODBC的效率较差且其驱动程序并不能夸平台，但是JDBC却很好的解决了这些问题。JDBC是Java程序和数据库管理系统之间的应用编程接口，它是一组由Java类、接口组成的API(Application Programming Interface，应用程序设计接口)。它设计的目的是以平台独立的方式实现Java应用程序和小程序对不同类型的数据库进行访问。

通过JDBC，我们可以在绝大多数的关系数据库上执行SQL查询。我们也可以把它看作是Java的平台无关性扩展到了数据库操作的方面。



简单的说JDBC能够完成以下3件事情：

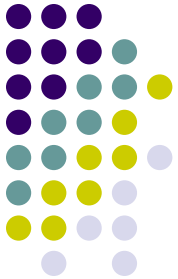
1、与一个数据库建立连接

2、向已连接的数据库发送SQL语句，进行SQL查询

3、对从数据库中返回的结果进行处理。

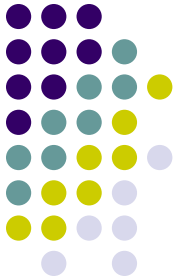
那么JDBC技术的主要优点是：

1、充分利用已有的数据资源。使用JDBC，Java程序可以很方便地访问已有的数据库资源。这使得我们可以引入Java这一新技术，而不必因此丢弃以前的系统和数据，这也是对Java技术的一个重要扩展。



2、开发简便。JDBC API是比较简单的，至少从基本的API方面说是这样的，这也使得使用Java开发数据库应用系统十分方便。

3、零配置。JDBC的一个突出的特点就是JDBC本身不需要任何配置。JDBC API已经是Java SDK的标准组成部分，因此安装了Java SDK，就安装了JDBC。而在JDBC中，所有建立数据库连接所需的信息都是通过JDBC URL定义，不需要任何配置。



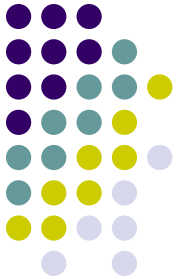
而且在JDBC 2.0中，还引入了一个十分好的机制，那就是JNDI(The Java Naming and Directory Interface)和数据源对象，JDBC可以通过注册在Java名字服务中的服务器信息建立连接，而数据源对象使得对数据源管理更方便，也带来了更好的移植性。

# JDBC标准以及主要接口和类

1、JDBC标准

和所有的Java技术一样，JDBC API一直处于发展之中，JDBC的标准也在不断的发展变化。目前JDBC的标准有3个：JDBC 1.0、JDBC 2.0以及才制订完不久的JDBC 3.0。

JDBC 1.0 API以及包含了数据举操作的基本功能，实际上我们在进行Web编程时使用的绝大多数API都还是JDBC 1.0标准里面定义的API，当然它也是JDBC的基础。



JDBC 2.0 API 则是在JDBC 1.0的基础上增加了两个包：JDBC 2.0核心API和JDBC 2.0扩展API，它们主要是在以下方面对1.0进行了扩展：

(1) 可滚动的结果集(Scrollable result sets)

(2) Java命名和目录接口(JNDI)

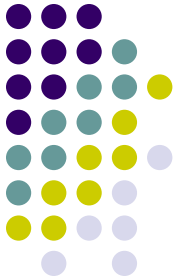
(3) 批处理更新(Batch updates)

(4) 连接池(Connection Pooling)

(5) 高级数据类型

(6) 分布式事务处理支持

(7) 列集(Rowsets)



那么JDBC 3.0 API又有了以下的一些新特性：

1、提供支持数据库连接池的框架

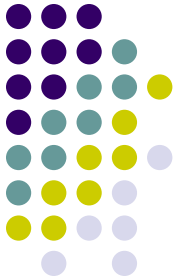
2、检索自动产生的关键字

3、返回多重结果

4、在事务中使用savepoint

5、支持更多的数据类型

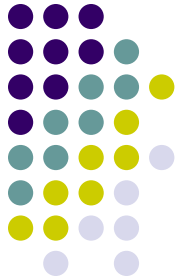
本章主要是介绍JDBC的基本功能，但是JDBC本身是一个相当复杂的技术，要想全面的深入的了解它，最好是参阅JDBC的规范、技术稳当以及专门的书籍等。



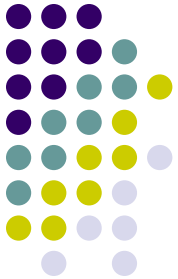
## 2、JDBC标准中主要的接口和类

JDBC中主要的接口和类包括：

Java.sql.CallableStatement
Java.sql.Connection
Java.sql.DataTruncation
Java.sql.Date
Java.sql.Driver
Java.sql.DriverManager
Java.sql.DriverPropertyInfo



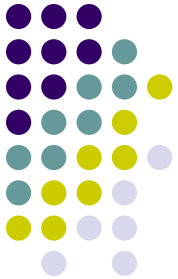
Java.sql.PreparedStatement
Java.sql.DatabaseMetaData
Java.sql.ResultSet
Java.sql.ResultSetMetaData
Java.sql.SQLException
Java.sql.SQLWarning
Java.sql.Statement
Java.sql.Time
Java.sql.Timestamp
Java.sql.Types



其中在java.sql.Statement接口中包含了两个借口：

Java.sql.PreparedStatement：用于处理预编译的SQL查询语句。

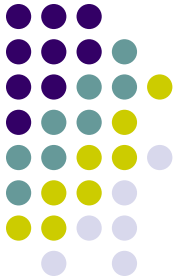
Java.sql.CallableStatement：用于处理数据库的存储过程



最重要的基本类介绍

(1) Connection：数据库的当前连接，通过它Java程序可以读取和写入数据，以及开发数据库的结构和功能，一般情况下，我们会使用DriverManager.getConnection()或者DataSource.getConnection()来调用创建连接

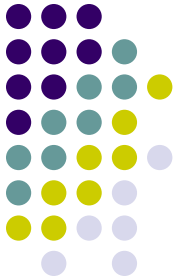
(2) Statement：允许通过一个连接发送SQL语句并检索结果集和修改其产生的各行数据的对象。



Statement：用于执行静态SQL字符串，一般使用Connection.createeStatement()创建一个Statement.

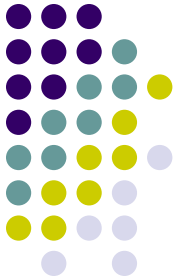
PreparedStatement：使用预编译SQL的Statement的扩展，可能会动态设置输入参数。PreparedStatement对象常用于一个SQL插入操作的循环，一般使用Connection.prepareStatement(sqlstring)创建PreparedStatement。

CallableStatement：调用一存储过程的PreparedStatement。但是并不是所有的数据库管理系统都支持存储过程，只要支持的，CallableStatement都提供标准调用语法。以供我们调用数据库的存储过程。



(3) ResultSet：一个SQL查询或对某元数据函数的调用产生的表格行的排序集合。一个ResultSet更常用做一个Statement.executeQuery(sqlstring)方法调用的返回。JDBC API提供了next()方法循环ResultSet的行，通过getXXX()方法来获取列值，注意这里的XXX是java代码类型。

(4) ResultSetMetaData：描述Resultset列的接口，通过调用结果集的getMetaData()方法获得，它包含描述列数、以及每一列的名字、显示尺寸、数据类型以及类名等方法。



(5) DriverManager： JDBC的管理器，作用于Java程序和驱动程序之间，用于管理JDBC驱动程序，跟踪可用的驱动程序并在数据库和驱动程序之间建立连接。一般的使用方法是DriverManager.getConnection()来得到连接。

(6) SQLException：JDBC API使用的异常类，SQLException带有提供任意厂家指定的错误代码的SQLState值的方法。

# JDBC的体系结构

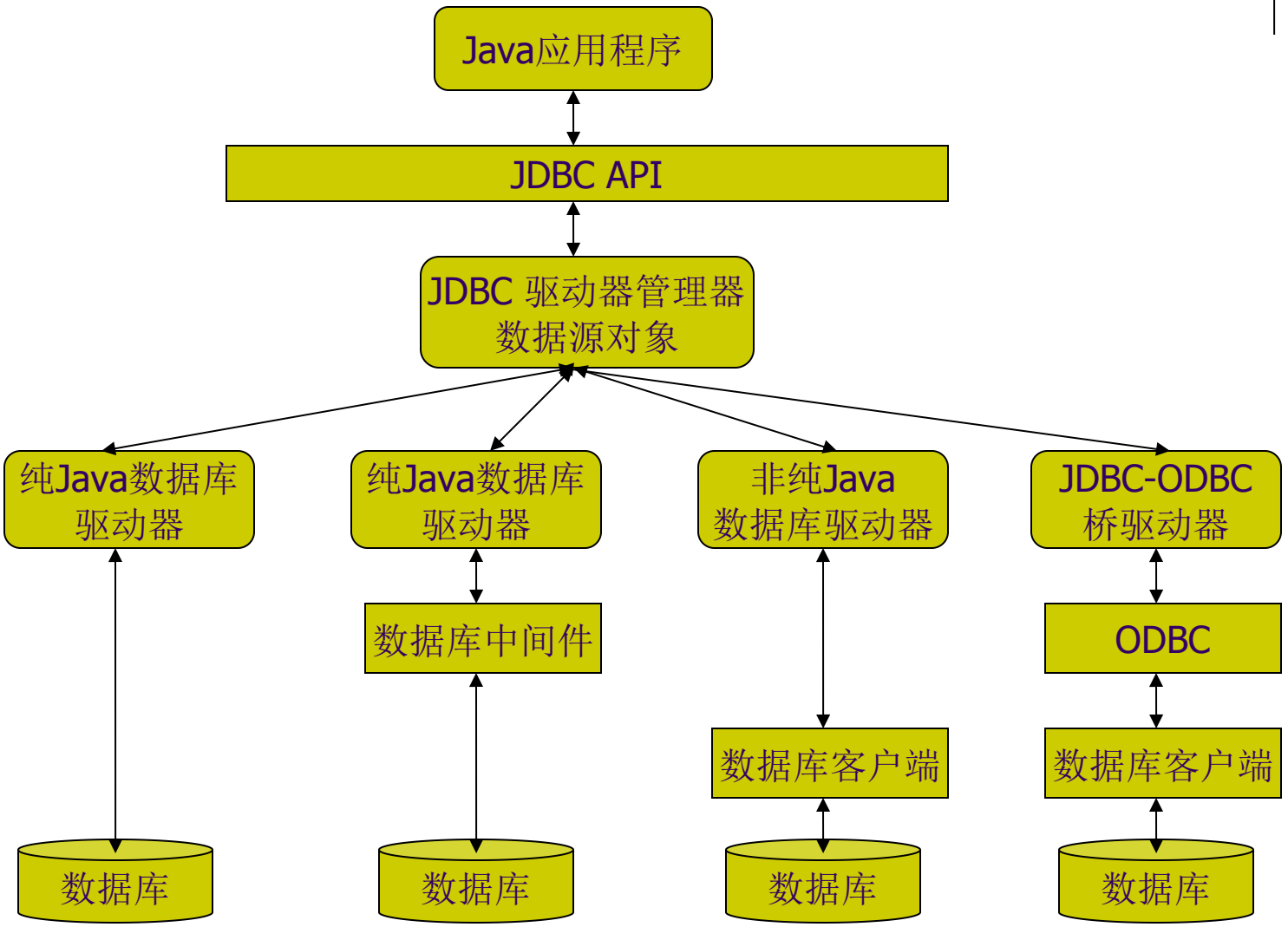
JDBC API主要完成以下3项功能，首先建立与数据库的连接，然后向数据库发送SQL查询操作，在得到数据库返回结果之后进行相应的处理。

这个功能是对数据库应用的一般的高度概括，那么这些功能是通过两组接口实现的：

1、JDBC API，应用程序设计人员使用的API

2、底层的JDBC驱动器API，数据库驱动程序设计人员使用的API

## JDBC的体系结构如下图所示：



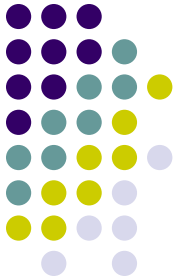
如图所示，Java应用程序使用JDBC API通过JDBC访问数据库，数据库的访问则在JDBC驱动器管理器和数据源对象的管理下进行的，它们还管理着实现数据库访问的底层数据库驱动程序。

JDBC API和JDBC驱动器管理器都内置在Java JDK中，因此对任何驱动器都是一样的，都遵循同样的标准。另外，Type 4类JDBC-ODBC桥也是内置在JDK中。以下便是4类JDBC数据库驱动程序，由于数据库驱动器一般都由数据库厂商提供，所以具体到某一个数据库产品，则可能仅提供了部分数据类型的驱动程序。

4类JDBC驱动程序的介绍：

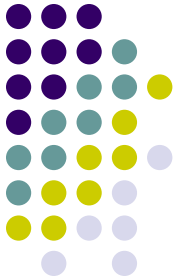
(1) Type 1驱动器

第一类驱动器是直接访问数据库的纯Java驱动器。这类驱动器将JDBC调用转换成数据库系统直接使用的网络协议，从而实现了客户端程序到数据库系统的直接访问。这一类的驱动程序的平台移植性最好，它是纯Java的，但是性能不太高，主要原因就是Java程序执行的性能不太高。



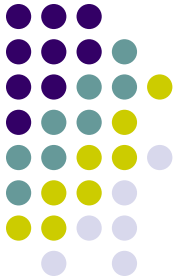
(2) Type 2驱动器

第二类驱动器是访问数据库中间件的纯Java驱动器，这类驱动器将JDBC调用先转换成数据库中间件的协议，在由中间件服务器将它转换成数据库的协议，这类驱动器的优点是可以利用中间件服务器方便地实现多种不同数据库的连接，而步需要像第一类驱动器那样对每一个数据库服务器实现一个JDBC驱动器，它的平台移植性也不错，也是纯Java的，但是考虑到数据库中间件的平台移植性，当然性能也一般。



(3) Type 3驱动器

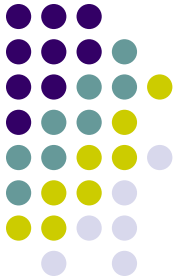
第三类驱动器是采用原始数据库API的非纯Java驱动器，这类驱动器将JDBC调用转换成对数据库客户端的调用，这类驱动器的性能是最好的，它的平台移植性也取决于数据库客户端部分，内驱动器的运行需要在系统中加载和使用数据库的专用客户端库文件，而这部分往往是同平台相关的。



(4) Type 4驱动器

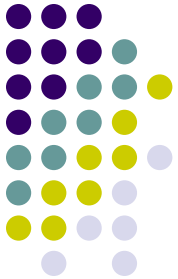
第四类驱动器是JDBC-ODBC桥，JDBC-ODBC桥通过ODBC实现JDBC的访问，实现了JDBC同ODBC的连接。由于ODBC在微软的Windows平台十分普及，可以连接Windows平台上绝大多数数据库，这样通过JDBC-ODBC桥，Java应用程序使用JDBC就是可以实现同Windows平台上绝大多数数据库的连接。

JDBC-ODBC对JDBC技术的推广很有意义，虽然JDBC-ODBC的性能不高，但是它方便地实现了广泛的连接，因此也是十分常用的。当然，在使用JDBC-ODBC桥时，系统中必须正确配置ODBC数据源和相应的数据库客户端驱动，平台移植性取决于ODBC部分。



JDBC在Web应用程序中的使用

JDBC既可以支持两层模型(客户/服务器模型)，也可以支持三层应用模型(浏览器/服务器模型)。当使用Java开发Web应用程序时，两种模型的JDBC应用都可能会遇到，但绝大多数情况下还是在三层应用模型下使用JDBC。两层模型的情况主要出现在使用applet等客户端构件直接访问数据库的情况。

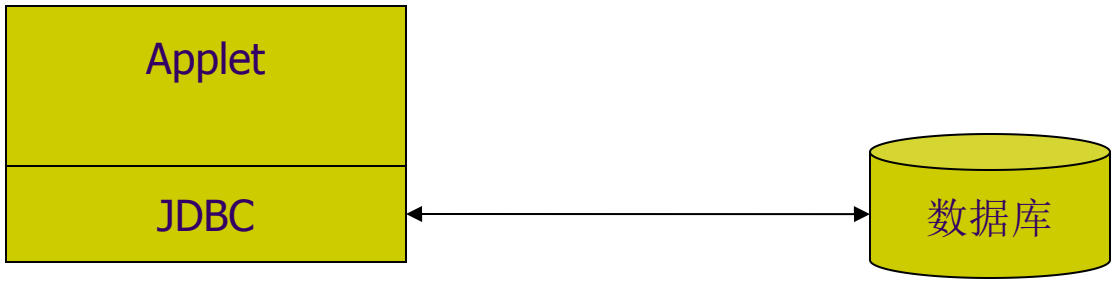


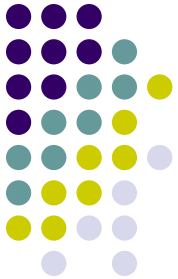
## 1、JDBC在两层模型中的使用

在两层模型中，客户端浏览器中运行的applet可以直接通过JDBC访问位于服务器上的数据库，在这种方式下一般只能采用Type1类驱动器，因为在浏览器端仅仅提供了Java Plug-in环境，只能运行Type1类驱动器，例如Oracle数据库就提供了专门的Java thin driver，applet可以使用这个JDBC驱动程序访问后台的Oracle数据库。

但是，除非必必需，否则一般不推荐这样使用。因为从applet中直接访问数据库不仅无法提供较高的性能，而且还会引发很多不必要的安全问题，因为对数据库的访问通路被暴露在外面了。

下图显示两层应用模型中JDBC的使用





## 2、JDBC在三层模型中的使用

在三层模型中，客户端的浏览器不直接访问位于服务器上的后台数据库，而是通过位于中间层的Web服务器或者支持Web的应用服务器来访问数据库。中间层程序主要是使用Servlet、JSP、JavaBean、EJB等方式实现，它们则是使用JDBC访问数据库。

三层模型的程序结构能够得到较高的数据库访问性能，因为位于中间层的服务器系统可以使用各种技术(如连接池等)集中优化数据库的访问，另外中间层较好的解决了安全问题，因为系统只有一个访问入口点，可以采用统一的安全策略，所以三层模型是我们开发具有数据库功能的Web应用的主要模型，并且可以使用任何一种驱动器。

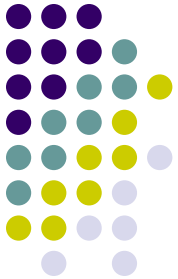
# JDBC程序设计

一般的JDBC程序都会涉及到以下主要的内容(JDBC程序的基本结构)：

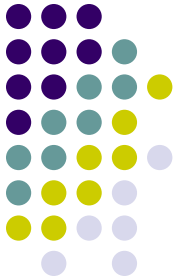
1、注册相应的驱动程序

要使用一个JDBC驱动器，首先需要有驱动器管理者对其进行注册。注册的语句是：Class.forName(DriverClassName);并且需要放在try/catch结构中。

其中DriverClassName就是数据库驱动程序的名字，如：sun.jdbc.odbc.JdbcOdbcDriver，这个驱动程序就是JDBC-ODBC桥驱动器。



它是JDK内置的，因此可以直接使用。而如果想要使用其它数据库厂商提供的专用驱动器，则需要首先将数据库驱动程序的类库文件包含到JDK的类目录中，例如：Oracle数据库提供的驱动器oracle.jdbc.driver.OracleDriver，如果要使用它首先要把它包含到类目录中来。



2、建立同数据库的连接

驱动程序加载成功后，就可以建立到数据库的连接，可以通过如下语句实现

```
Connection
    con=DriverManager.getConnection(URL,Username,Password);
```

其中URL是数据库连接串，称为JDBC URL，如：jdbc:odbc:soft，它说明通过ODBC的DSN soft建立同数据库的连接。

JDBC URL的标准格式为：

jdbc:<子协议>:<子名字>。

再如：jdbc:oracle:thin:@locatehost:1521:oradb

是JDBC连接到本地机器上的名字为oradb的

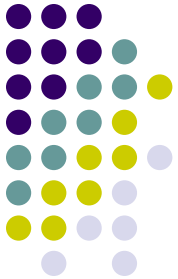
Oracle数据库上。

另外的两个参数Username和Password分别表示

数据库连接地用户名和口令，如果数据库不需要

这两个参数，就可以不写它们。

注意，数据库连接是一个很重要的资源，建立一个数据库连接的开销往往是很大的，因此在程序中应该尽量避免不必要的连接，并尽量较少连接的占用时间，而且我们可以使用很多机制来提高连接的效率，从而提高数据库操作的性能。



3、申请进行数据库操作的语句对象

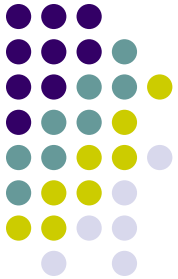
在建立了数据库连接之后，要进行实际的数据库操作之前，还需要获取操作数据库的语句对象(Statement),获取的语句如下：

Statement stmt=con.createStatement();

其中con是前面建立好的数据库连接。一旦创建好语句对象，它就可以执行SQL命令了。

4、进行数据库操作

有了语句对象之后，我们就可以进行数据库操作了。数据库的操作分为两类：一种是数据库查询操作，主要是通过SQL Select语句进行，它将返回查询的结果数据，称为结果集(ResultSet)；另外一种就是数据库更新操作，包括SQL Update、Insert、Delete等修改数据库内容的语句或者是数据库建立的SQL命令。这一类操作不返回结果集，只是返回一个有含义的整数罢了。



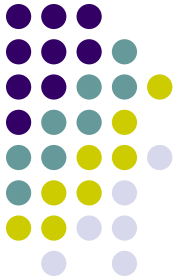
一般有4种执行SQL命令的方法：

(1) executeUpdate：趋向用于SQL Insert、Update或Delete语句，或者数据定义语句，返回已修改的行数。

(2) executeQuery：用于执行一个SQL Select语句并返回结果集。

(3) execute：可用于前面两种情况，但更趋向用于返回一个修改数量、多结果集或者量这结合的语句

(4) executeBatch：允许在一个批处理中执行多修改语句，修改数量返回到一个数组中。

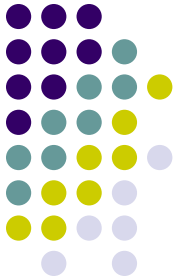


## 5、对数据库操作结果进行处理

对数据库操作结果的最主要的处理就是处理查询操作返回的结果集，结果集是一个Java对象，它定义类为ResultSet，我们就可以使用ResultSet类的方法处理结果集了。其中最常用的处理方法：

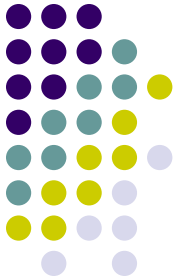
(1) next()：取下一个结果纪录

(2) getXXX(int i)：按照XXX所指的数据类型取出第i个列的值



5、最后关闭数据库操作的对象，释放相应资源

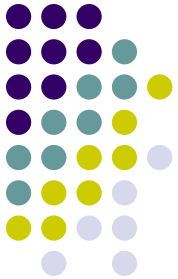
数据库操作结束，应该将数据库操作结果集、申请的语句对象、建立的数据库连接关闭，这些对象都具有一个close()方法,用于完成关闭操作，虽然程序推出后，Java虚拟机也会释放这些资源，但是主动释放不用的资源是很好的编程风格。



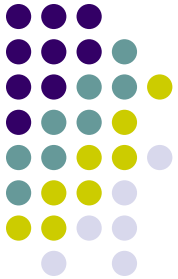
## 6、异常处理

因为数据库操作往往是一个应用程序的复杂且关键的部分，其中影响其运行的因素很多，难免会出现各种异常，所以对数据库操作的代码一定要进行保护，捕获其抛出的异常。

数据库操作一般都抛出SQLException异常，程序中可以进行捕获并进行相应的处理。



对于Servlet、JavaBean等Java程序，可以直接使用try/catch结构捕获异常，然后进行相应的处理，在Servlet程序中还可以在处理请求的方法(doGet、doPost方法)的定义中使用throw语句把异常向外抛出，由其他程序去截获。JSP程序中我们也可以不使用try/catch结构而指定异常处理页面来捕获异常，进行相应的处理。



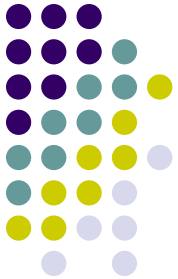
## 7、使用元数据

JDBC提供大量的元数据—数据的数据，即对于数据库连接和结果集的集合。

关于JDBC连接的信息可以使用Connection.getMetaData()获得，此方法返回java.sql.DatabaseMetaData的实例，它是一个比java.sql或javax.sql包中任何其他接口或类中包含的方法都要多的一个接口。这些方法描述了数据库支持的特性，其中包含表格以及表格中的列，使用元数据可以最小化SQL语言和数据库系统功能的差别。

除了使用DatabaseMetaData来获取连接的数据库的元数据以外，我们还可以使用ResultSetMetaData来获取官易一个返回结果集列的信息。一般的情况下，我们可以通过程序调用ResultSet.getMetaData()方法来得到。

# JDBC程序示例



1、示例1:simpleJdbcServlet.java

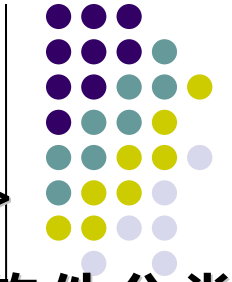
```
Import java.util.*;
Import java.io.*;
Import java.sql.*;
Import javax.Servlet.*;
Import javax.Servlet.http.*;
Public class simpleJdbcServlet extends HttpServlet{
    //设置数据库连接字符串
    String strCon=“jdbc:odbc:soft”;
```

```
    //Servlet初始化
    public void init(ServletConfig conf) throws ServletException {
    super.init(conf);
   try{
       //装载JDBC驱动程序
     Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
    }catch(Exception e){
          System.err.println(“没有找到JDBC-ODBC驱动器”)；
    }
}
```

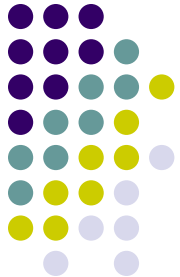
```
//处理GET请求
public void doGet(HttpServletRequest
request,HttpServletResponse response) throws
ServletException,IOException{
response.setContentType(“text/html;charset=gb2312”);
PrintWriter out=response.getWriter();
out.println(“<html>”);
out.println(“<body>”);

Connection sqlCon;
Statement sqlStmt;
ResultSet sqlRst;
```

```
String strSQL;
try{
        //连接数据库
sqlCon=java.sql.DriverManager.getConnection(strCon);
        //创建一个SQL语句
sqlStmt=sqlConn.createStatement();
strSQl=“select * from CATALOG”
sqlRst=sqlStmt.executeQuery(strSQL);
out.println(“<table border=\” 1\” width=\” 100\>”)
out.println(“<tr>”);
```

```
  out.println("<dt width=\" 100\"  height=\" 25\" >
  <font  face=\ "宋体\"   size=\" 3\" ><strong>软件分类
  </strong></font></dt>");
 out.println("</tr>");
  while(sqlRst.next()){
    out.println("<tr>");
    out.print("<td width=\" 100\"  height=\" 18\>")
    out.print(sqlRst.getString(2));
   out.println("</td></tr>");
}
out.println("</table>");
```



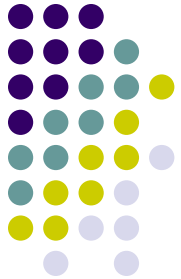
```
    sqlRst.close();
    sqlStmt.close();
   sqlCon.close();
   out.println(“</table>”);
   }catch(Exception e){
   out.println(“”<p> </p><p
   align=\”center\”><font size=\”2\”
   color=\”#FF0000\”>数据库错误。</font></p>”);
   System.err.println(“数据库错误”);
   }
   out.println(“</body>”);
   out.println(“</html>”);
  }
}
```



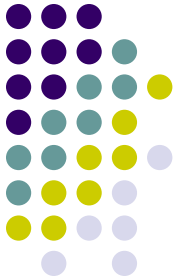
## 2、简单示例2

假设某公司拥有内部职工数据库，它包含两个关系表departments和employees。它们的DDL语句分别是：

```
Create table departments(
    deptno          char(2),
    deptname        char(40);
)
```

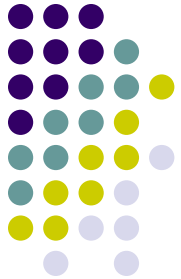


```
Create table employees(
    deptno      char(2),
    empno       char(4),
    name        char(20),
    hiredate    date,
   ismgr        bit,
    email       char(32),
   phone        char(20)
)
```



要在JSP页面上显示部门名称、以及该部门经理名字、Email以及电话。那么此SQL查询语句应为：

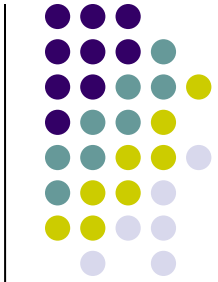
```
Select D.deptname, E.name, E.email, E.phone
From departments D, employees E
Where D.deptno=E.deptno and ismgr=1
Order By D.detptname
```



显示内容的JSP页面代码为：

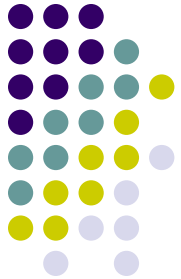
```
<%@ page session=“false” %>
<%@ page import=“java.sql.*” %>
<%@ page import=“java.util.*” %>
<html>
<head><title>Department Managers</title></head>
<body>
<img src=“images/lyric_note.png” border=0><p>
<hr color=“#000000” >
<h2>Department Managers</h2>
```

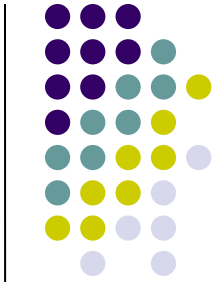
```
<% String driver= “org.enhydra.instantdb.jdbc.idbDriver” ;
  String          url=          “jdbc:idb:d:/lyricnote/WEN-
   INF/database/internal/db.prp” ;
  Class.forName(driver);
  Connection con=null;
   try{
         con=DriverManager.getConnection(URL);
       sql= “select D.deptname, E.name, E.email,     E.phone from
   departments D, employees E
         where D.deptno=E.deptno and ismgr=1
         order By D.detptname” ;
```



```
    Statement stmt=con.createStatement();
   ResultSet rs=stmt.executeQuery(sql); %>
<dl>
<%
    while(rs.next()){
     String dept=rs.getString(1);
     String name=rs.getString(2);
     String email=rs.getString(3);
     String phone=rs.getString(4);
%>
<dt><b><%=dept%></b></dt>
<dd><%=name%>,<%=email%>,<%=phone%>
</dd>
```

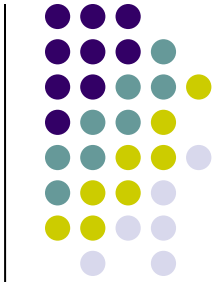
```
<%  }
        rs.close();
        rs=null;
        stmt.close();
        stmt=null;
    }
     finally{
            if(con!=null){
            con.close();
            }
     } %>
</dl>
</body></html>
```





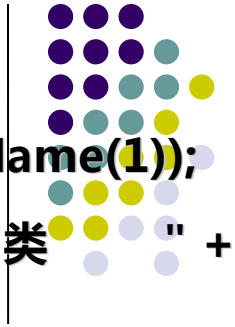
3、简单示例3：获得元数据

```
import java.sql.*;
public class DBAccess {
 public static java.sql.Connection conn = null;
private String sqlStr = "";
public DBAccess()
 {
try {
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
conn = DriverManager.getConnection("jdbc:odbc:TestDB", "admin","");
  }
catch (ClassNotFoundException ex) {
 System.out.println(ex.toString()); }
catch(SQLException sqlEx){
System.out.println(sqlEx.toString());}
}
```



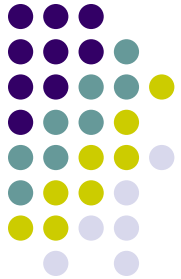
```
public ResultSet Search()
{
 ResultSet rset = null;
sqlStr = "SELECT * FROM STUDENTINFO";
 Statement smt = null;
 try {
 smt = conn.createStatement();
 rset = smt.executeQuery(sqlStr);
}
 catch (SQLException ex) {
 System.out.println("Exception:"+ex.toString());
}
 return rset;
 }
```

```
public void getResultSetMetaData()
{
 ResultSet rs = null;
 try {
 String[] tp = {"TABLE"};
rs = this.Search();
 ResultSetMetaData rsmd = rs.getMetaData();
/*
获得ResultSetMeataData对象。所有方法的参数都是列的索引号，即第几列，从1开始
*/
System.out.println("下面这些方法是ResultSetMetaData中方法");
```



```
System.out.println("获得1列所在的Catalog名字 : " + rsmd.getCatalogName(1));
System.out.println(" 获 得 1 列 对 应 数 据 类 型 的 类 " +
    rsmd.getColumnClassName(1));
System.out.println("获得该ResultSet所有列的数目 " + rsmd.getColumnCount());
System.out.println("1 列 在 数 据 库 中 类 型 的 最 大 字 符 个 数 " +
    rsmd.getColumnDisplaySize(1));
System.out.println(" 1列的默认的列的标题" + rsmd.getColumnLabel(1));
System.out.println( “1列的模式” + rsmd.GetSchemaName(1));
System.out.println("1 列 的 类 型 , 返 回 SqlType 中 的 编 号 " +
    rsmd.getColumnType(1));
```

```
System.out.println("1 列 在 数 据 库 中 的 类 型 ， 返 回 类 型 全 名 " +
    rsmd.getColumnTypeName(1));
System.out.println("1列类型的精确度(类型的长度): " + rsmd.getPrecision(1));
System.out.println("1列小数点后的位数 " + rsmd.getScale(1));
System.out.println("1 列 对 应 的 模 式 的 名 称 （ 应 该 用 于 Oracle ） " +
    rsmd.getSchemaName(1));
System.out.println("1列对应的表名 " + rsmd.getTableName(1));
System.out.println( “1列是否自动递增” + rsmd.isAutoIncrement(1));
System.out.println( “1列在数据库中是否为货币型” + rsmd.isCurrency(1));
System.out.println( “1列是否为空” + rsmd.isNullable(1));
System.out.println( “1列是否为只读” + rsmd.isReadOnly(1));
System.out.println( “1列能否出现在where中” + rsmd.isSearchable(1));
```



```
}
catch (SQLException ex) {
 ex.printStackTrace();
}
}
public static void main(String args[])
{
  DBAccess dbAccess = new DBAccess();
  dbAccess.getResultSetMetaData();
}
}
```