# JDBC

**（Java DataBase Connectivty）**

麦田技术博客

**整理人：徐仕锋（Eric）**

**版本号：v2009-1-20**

# 一、 JDBC 概述

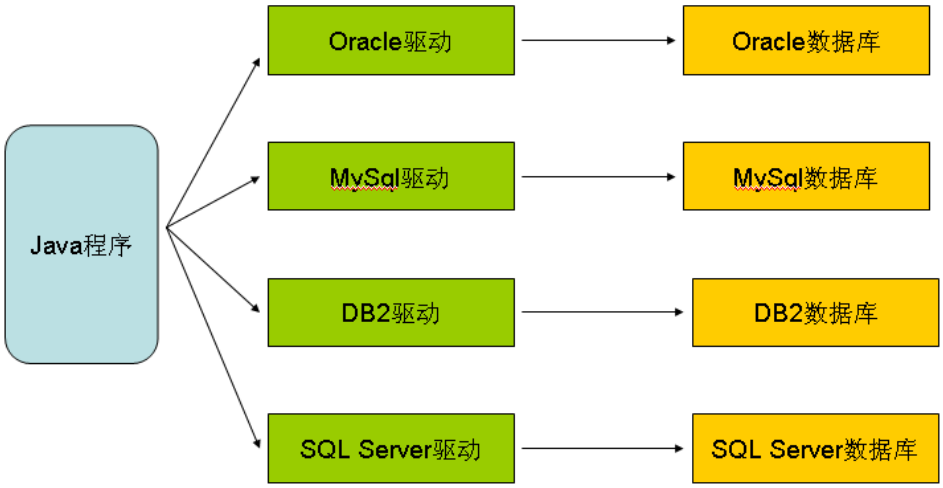
## 1 概述

JDBC 从物理结构上说就是 Java 语言访问数据库的一套接口集合。从本质上来说就是调用者（程序员）和实现者（数据库厂商）之间的协议。JDBC 的实现由数据库厂商以驱动程序的形式提供。JDBC API 使得开发人员可以使用纯 Java 的方式来连接数据库，并进行操作。

ODBC：基于 C 语言的数据库访问接口。

- JDBC 也就是 Java 版的 ODBC。
- JDBC 的特性：高度的一致性、简单性（常用的接口只有 4、5 个）。

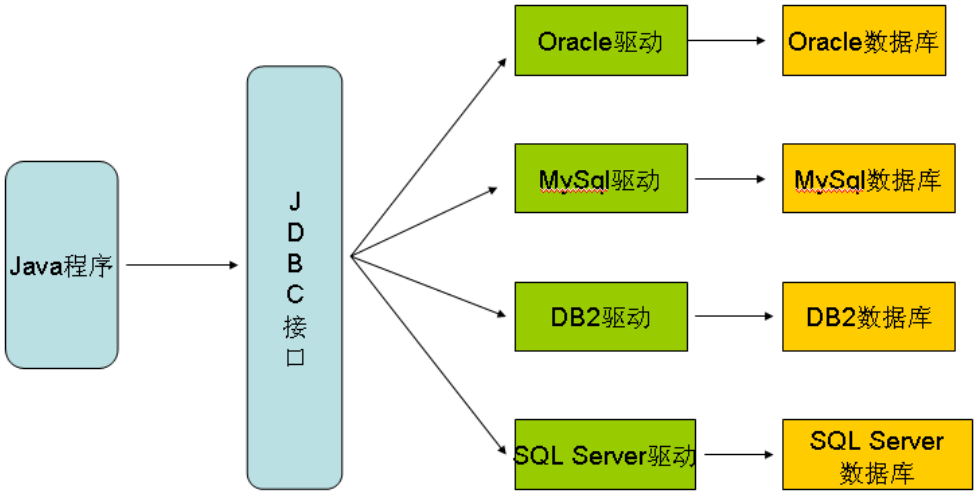
## 2 JDBC 的发展




没有 JDBC 之前 java 程序是这样连接各种数据库的。

缺点：1、要求程序员必须熟悉编写 java 程序连接各种数据库的驱动。

2、移植性很不好，更改数据库必须重新编写连接数据库的驱动程序。



用了 JDBC 以后 java 连接各种数据库方便多了！

# 3 JDBC 的 API 介绍

在 JDBC 中包括了两个包：java.sql 和 javax.sql。

① java.sql　基本功能。这个包中的类和接口主要针对基本的数据库编程服务，如生成连接、执行语句以及准备语句和运行批处理查询等。同时也有一些高级的处理，比如批处理更新、事务隔离和可滚动结果集等。

② javax.sql　扩展功能。它主要为数据库方面的高级操作提供了接口和类。如为连接管理、分布式事务和旧有的连接提供了更好的抽象，它引入了容器管理的连接池、分布式事务和行集（RowSet）等。

**主要对象和接口：**

注：除了标出的 Class,其它均为接口。

| API | 说明 |
|---|---|
| **java.sql.Connection** | 与特定数据库的连接（会话）。能够通过 getMetaData 方法获得数据库提供的信息、所支持的 SQL 语法、存储过程和此连接的功能等信息。代表了数据库。 |
| **java.sql.Driver** | 每个驱动程序类必需实现的接口，同时，每个数据库驱动程序都应该提供一个实现 Driver 接口的类。 |
| **java.sql.DriverManager (Class)** | 管理一组 JDBC 驱动程序的基本服务。作为初始化的一部分，此接口会尝试加载在"jdbc.drivers"系统属性中引用的驱动程序。只是一个辅助类，是工具。 |
| **java.sql.Statement** | 用于执行静态 SQL 语句并返回其生成结果的对象。 |
| **java.sql.PreparedStatement** | 继承 Statement 接口，表示预编译的 SQL 语句的对象，SQL 语句被预编译并且存储在 PreparedStatement 对象中。然后可以使用此对象高效地多次执行该语句。 |
| **java.sql.CallableStatement** | 用来访问数据库中的存储过程。它提供了一些方法来指定语句所使用的输入/输出参数。 |
| **java.sql.ResultSet** | 指的是查询返回的数据库结果集。 |
| **java.sql.ResultSetMetaData** | 可用于获取关于 ResultSet 对象中列的类型和属性信息的对象。 |

# 4 驱动程序工作分类

驱动程序按照工作方式分为四类：

1、 JDBC-ODBC bridge + ODBC 驱动

JDBC-ODBC bridge 桥驱动将 JDBC 调用翻译成 ODBC 调用，再由 ODBC 驱动翻译成访问数据库命令。

优点：可以利用现存的 ODBC 数据源来访问数据库。

缺点：从效率和安全性的角度来说的比较差。不适合用于实际项目。

2、 基于本地 API 的部分 Java 驱动

我们应用程序通过本地协议跟数据库打交道。然后将数据库执行的结果通过驱动程序中的 Java 部分返回给客户端程序。

优点：效率较高。

缺点：安全性较差。

3、 纯 Java 的网络驱动

（中间协议） （本地协议）

app → JDBC → 纯 Java ——————→ 中间服务器 ——————→ DB

缺点：两段通信，效率比较差

优点：安全信较好

4、 纯 Java 本地协议：通过本地协议用纯 Java 直接访问数据库。

特点：效率高，安全性好。

## 二、 JDBC 编程步骤

必须掌握！

## 1 注册一个 Driver

注册驱动程序有三种方式：

**方式一：** Class.forName("oracle.jdbc.driver.OracleDriver");

JAVA 规范中明确规定：所有的驱动程序必须在静态初始化代码块中将驱动注册到驱动程序管理器中。

**方式二：** Driver drv = new oracle.jdbc.driver.OracleDriver();

DriverManager.registerDriver(drv);

**方式三：** 通过设置系统属性 jdbc.drivers，编译时在虚拟机中加载驱动.

javac xxx.java ( 要确保驱动包在 classpath 里)

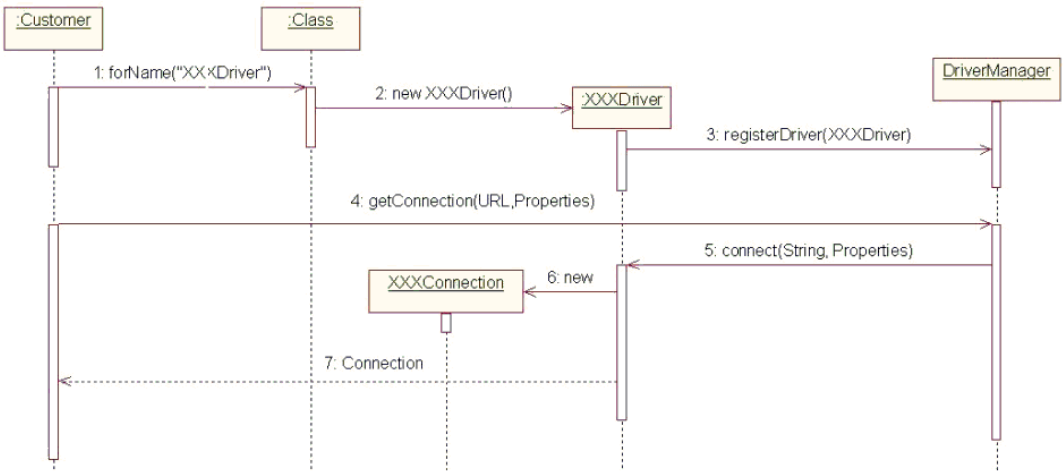
java –D jdbc.drivers=驱动全名 类名

使用系统属性名，加载驱动 -D 表示为系统属性赋值

附 : mysql 的 Driver 的全名 com.mysql.jdbc.Driver

SQLServer 的 Driver 的全名 com.microsoft.jdbc.sqlserver.SQLServerDriver

- **JDBC 中驱动加载的时序图**




以上是 JDBC 中驱动加载的时序图。时序图主要有以下 7 个动作：

1. 客户调用 Class.forName(“XXXDriver”)加载驱动。
2. 此时此驱动类首先在其静态语句块中初始化此驱动的实例。
3. 再向驱动管理器注册此驱动。
4. 客户向驱动管理器 DriverManager 调用 getConnection 方法，
5. DriverManager 调用注册到它上面的能够理解此 URL 的驱动建立一个连接，
6. 在该驱动中建立一个连接，一般会创建一个对应于数据库提供商的 XXXConnection 连接对象，
7. 驱动向客户返回此连接对象，不过在客户调用的 getConnection 方法中返回的为一个 java.sql.Connection 接口，而具体的驱动返回一个实现 java.sql.Connection 接口的具体类。

# 2 建立连接

conn=DriverManager.getConnection(“**jdbc:oracle:thin:@192.168.0.20:**1521**:tarena**”, ” User”,” Pasword”);

用户名，密码

IP 地址及端口号和数据库实例名

**Connection** 连接是通过 **DriverManager** 的静态方法 getConnection(.....)来得到的，这个方法的实质是把参数传到实际的 Driver 中的 connect()方法中来获得数据库连接的。

Oracle URL 的格式:

jdbc:oracle:thin:（协议）@XXX.XXX.X.XXX:XXXX（IP 地址及端口号）:XXXXXXX（所使用的库名）

MySql URL 的写法 例： jdbc:mysql://192.168.8.21:3306/test

# 3 获得一个 Statement 对象

Statement stmt = conn.createStatement();

# 4 通过 Statement 执行 Sql 语句

stmt.executeQuery(String sql); //返回一个查询结果集。

stmt.executeUpdate(String sql); //返回值为 int 型，表示影响记录的条数。

将 sql 语句通过连接发送到数据库中执行，以实现对数据库的操作。

# 5 处理结果集

使用 **Connection** 对象获得一个 **Statement**，Statement 中的 **executeQuery(String sql)** 方法可以使用 select 语句查询，并且返回一个结果集 ResultSet 通过遍历这个结果集，可以获得 select 语句的查寻结果，ResultSet 的 **next()**方法会操作一个游标从第一条记录的前面开始读取，直到最后一条记录。**executeUpdate(String sql)** 方法用于执行 DDL 和 DML 语句，比如可以 update，delete 操作。

只有执行 select 语句才有结果集返回。

例： **Statement str=con.createStatement();** **//创建 Statement**

```
String sql=”insert into test(id,name) values(1,”+””+”test”+””+”)”;
str. executeUpdate(sql);//执行 Sql 语句
String sql=”select * from test”;
ResultSet rs=str. executeQuery(String sql);//执行 Sql 语句，执行 select 语句后有结果集
//遍历处理结果集信息
    while(rs.next()){
      System.out.println(rs.getInt(“id”));
      System.out.println(rs.getString(“name”))
 }
```

next()如果有下一条记录返回 true,否则为 false;有，则游标向下一条记录.

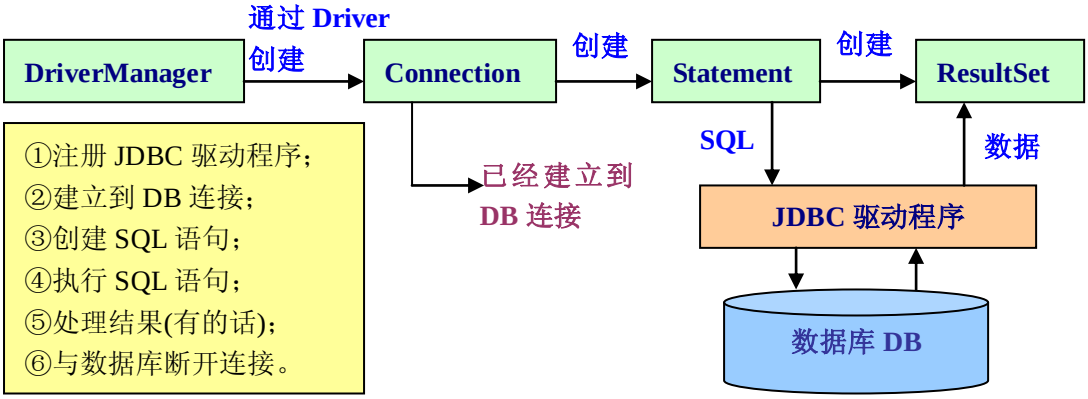
## 6 关闭数据库连接（释放资源）

**rs.close();** **stmt.close();** **con.close();**

**ResultSet Statement Connection 是依次依赖的。**

注意：要按先 ResultSet 结果集，后 Statement，最后 Connection 的顺序关闭资源，因为 Statement 和 ResultSet 是需要连接时才可以使用的，所以在使用结束之后有可能其它的 Statement 还需要连接，所以不能现关闭 Connection。

## 图形演绎编写 JDBC 程序的一般过程




## 三、 JDBC 中几个重要接口

# 1 Statement

**—— SQL 语句执行接口**

Statement 接口代表了一个数据库的状态，在向数据库发送相应的 SQL 语句时，都需要创建 Statement 接口或者 PreparedStatement 接口。在具体应用中，Statement 主要用于操作不带参数（可以直接运行）的 SQL 语句，比如删除语句、添加或更新。

扩展：如何进行模糊查询？

----关键点：如何根据条件拼接 SQL 语句。

# 2 PreparedStatement

**—— 预编译的 Statement**

**第一步：**通过连接获得 PreparedStatement 对象，用带占位符(?)的 sql 语句构造。

PreparedStatement pstm = con.preparedStatement(“select * from test where id=?”);

**第二步：**设置参数

pstm.setString(1，“ganbin”);

**第三步：**执行 sql 语句

Rs = pstm.excuteQuery();

statement 发送完整的 Sql 语句到数据库不是直接执行而是由数据库先编译，再运行。而 PreparedStatement 是先发送带参数的 Sql 语句，再发送一组参数值。如果是同构的 sql 语句，PreparedStatement 的效率要比 statement 高。而对于异构的 sql 则两者效率差不多。

同构：两个 Sql 语句可编译部分是相同的，只有参数值不同。

异构：整个 sql 语句的格式是不同的

注意点：1、使用预编译的 Statement 编译多条 Sql 语句一次执行

2、可以跨数据库使用，编写通用程序

3、能用预编译时尽量用预编译

# 3 ResultSet

**——结果集操作接口**

ResultSet 接口是查询结果集接口，它对返回的结果集进行处理。ResultSet 是程序员进行 JDBC 操作的必需接口。

# 4 ResultSetMetaData

**——元数据操作接口**

ResultSetMetaData 是对元数据进行操作的接口，可以实现很多高级功能。Hibernate 运行数据库的操作，大部分都是通过此接口。可以认为，此接口是 SQL 查询语言的一种反射机制。

ResultSetMetaData 接口可以通过数组的形式，遍历数据库的各个字段的属性，对于我们开发者来说，此机制的意义重大。

JDBC 通过元数据(MetaData)来获得具体的表的相关信息，例如，可以查询数据库中有哪些表，表有哪些字段，以及字段的属性等。MetaData 中通过一系列 getXXX 将这些信息返回给我们。

MetaData 包括：
- 数据库元数据 Database MetaData　　使用 connection.getMetaData()获得包含了关于数据库整体元数据信息。
- 结果集元数据 Result Set MetaData　　使用 resultSet.getMetaData()获得比较重要的是获得表的列名、列数等信息。

结果集元数据对象：ResultSetMetaData meta = rs.getMetaData();

- ✓ 字段个数：meta.getColomnCount();
- ✓ 字段名字：meta.getColumnName();
- ✓ 字段 JDBC 类型：meta.getColumnType();
- ✓ 字段数据库类型：meta.getColumnTypeName();

数据库元数据对象：DatabaseMetaData　dbmd = con.getMetaData();

数据库名＝dbmd.getDatabaseProductName();
数据库版本号＝dbmd.getDatabaseProductVersion（）;
数据库驱动名＝dbmd.getDriverName（）;
数据库驱动版本号＝dbmd.getDriverVersion（）;
数据库 Url＝dbmd.getURL（）;
该连接的登陆名＝dbmd.getUserName（）;

# 四、 JDBC 异常处理：

JDBC 中，和异常相关的两个类是 **SQLException** 和 **SQLWarning**。

1. SQLException 类：用来处理较为严重的异常情况。

比如：① 传输的 SQL 语句语法的错误；
② JDBC 程序连接断开；
③ SQL 语句中使用了错误的函数。

SQLException 提供以下方法：

getNextException() —— 用来返回异常栈中的下一个相关异常；
getErrorCode() —— 用来返回代表异常的整数代码 (error code)；
getMessage() —— 用来返回异常的描述信息 (error message)。

2. SQLWarning 类：用来处理不太严重的异常情况，也就是一些警告性的异常。其提供的方法和使用与 SQLException 基本相似。

结合异常的两种处理方式，明确何时采用哪种。
A. throws　　处理不了，或者要让调用者知道，就 throws;
B. try … catch　　能自行处理，就进行异常处理。

# 五、 JDBC 中使用 Transaction 编程

# 1 事务的四大特性

**事务是具备以下特征(ACID)的工作单元：**

(1) **原子性**

事务的原子性指的是，事务中包含的程序作为数据库的逻辑工作单位，它所做的对数据修改操作要么全部执行，要么完全不执行。

原子操作，也就是不可分割的操作，必须一起成功一起失败。

(2) **一致性**

事务的一致性指的是在一个事务执行之前和执行之后数据库都必须处于一致性状态。这种特性称为事务的一致性。假如数据库的状态满足所有的完整性约束，就说该数据库是一致的。

(3) **分离性**

分离性指并发的事务是相互隔离的。即一个事务内部的操作及正在操作的数据必须封锁起来，不被其它企图进行修改的事务看到。

(4) **持久性**

持久性意味着当系统或介质发生故障时，确保已提交事务的更新不能丢失。即一旦一个事务提交，DBMS 保证它对数据库中数据的改变应该是永久性的，耐得住任何系统故障。持久性通过数据库备份和恢复来保证。

# 2 事务处理三步曲

**① connection.setAutoCommit(false);　　//把自动提交关闭**
**② 正常的 DB 操作　　//若有一条 SQL 语句失败了，自动回滚**
**③ connection.commit()　　//主动提交**
**或 connection.rollback()　　//主动回滚**

**完整的代码片段：**

```
try{
    con.setAutoCommit(false);     //step① 把自动提交关闭
    Statement stm = con.createStatement();
    stm.executeUpdate("insert into person(id, name, age) values(520, 'X-Man', 18)");
    stm.executeUpdate("insert into Person(id, name, age) values(521, 'Super', 19)");
    //step② 正常的 DB 操作
    con.commit();                          //step③ 成功主动提交
} catch(SQLException e){
    try{
        con.rollback();
    } catch(Exception e){ e.printStackTrace(); }      //step③ 失败则主动回滚
}
```

# 3 JDBC 事务及事务隔离级别

**JDBC 事务并发产生的问题:**

① **脏读（Dirty Reads）** 一个事务读取了另一个并行事务还未提交的数据。

② **不可重复读（UnRepeatable Read）** 一个事务再次读取之前的数据时，得到的数据不一致，被另一个已提交的事务修改。

③ **幻读（Phantom Read）** 一个事务重新执行一个查询，返回的记录中包含了因为其它最近提交的事务而产生的新记录。

为了避免以上三种情况的出现，则采用

**事务隔离级别:**

| | |
|---|---|
| **TRANSACTION_NONE** | 不使用事务 |
| **TRANSACTION_READ_UNCOMMITTED** | 可以读取未提交数据 |
| **TRANSACTION_READ_COMMITTED** | 可以避免脏读，不能够读取没提交的数据，最常用的隔离级别　大部分数据库的默认隔离级别 |
| **TRANSACTION_REPEATABLE_READ** | 可以避免脏读，不可以重复读取 |
| **TRANSACTION_SERIALIZABLE** | 可以避免脏读，不可重复读取和幻读，（事务串行化）会降低数据库效率 |

以上的五个事务隔离级别都是在 **Connection** 类中定义的静态常量，使用 **setTransactionIsolation(int level)** 方法可以设置事务隔离级别。

**比如：con.setTransactionIsolation(Connection.**TRANSACTION_READ_UNCOMMITTED**);**

# 六、 JavaBean 的定义

1　是一个普通的 Java 类
2　在结构上没有预先的规定，不需要容器，不需要继承类或实现接口
3　要求必须放在包中，要求实现 Serializable 接口
4　要求有一个无参的构造方法.
5　属性的类型必须保持唯一，get 方法返回值必须和 set 方法参数类型一致
6　对每个属性要有对应的 get 和 set 方法。注：隐藏属性可以没有
7　可以有外观作为显示控制，事件机制。

# 七、 JDBC2.0 新特性

## 1 Scrollability 结果集可滚动

滚动：可双向支持绝对与相对滚动，对结果集可进行多次迭代。

```
Con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);
```

- TYPE_FORWARD_ONLY：

该常量指示指针只能向前移动的 ResultSet 对象的类型。

- TYPE_SCROLL_SENSITIVE：

该常量指示可滚动并且通常受其他的更改影响的 ResultSet 对象的类型。

- CONCUR_UPDATABLE：

该常量指示可以更新的 ResultSet 对象的并发模式。

绝对定位：boolean absolute(int row)将游标移动到指定位置。
相对定位：void afterLast（）将游标向后移动一位。
void beforeFirst（）。将游标向前移动一位。
boolean first（）将游标移动到结果集最前
boolean last（）将游标移动到结果集末尾。

## 2 Updatability 结果集可更新

更新：

```
// 定位到要更新的记录
  rs.absolute(int row);
//更新该行记录的信息
rs.updateString(“name”,”Tony”);
rs.updateInt(1,”122323”);
//更新底层数据
rs.updateRow();
```

插入：

```
//把指针移动到可插入的行
rs.moveToInsertRow();
// 更新该行记录的信息
rs.updateInt(1,113);
rs.updateString(2, "test2");
rs.updateDate(3,Date.valueOf("1999-9-9"));
rs.updateFloat(4,80.5f);
 // 插入数据
rs.insertRow();
rs.moveToCurrentRow();
```

删除：

```
// 定位到要删除的记录
Rs.absolute( int row);
//删除该行记录
rs.deleteRow();
```

注：只有在必要的时候（如桌面应用）才用结果集更新数据库，因为使用结果集更新数据库效率低下。可更新结果集还要看数据库驱动程序是否支持，如 Oracle 就支持 MySql 不支持。并且只能针对一张表做结果集更新。而且不能有 join 操作。必须有主健，必须把非空没有默认值的字段查出。处理可更新结果集时不能用 select *来执行查询语句，必须指出具体要查询的字段。

## 3 Batch updates 可批量更新

将一组对数据库的更新操作发送到数据库统一执行(数据库支持并发执行操作)，以提高效率。主要是通过减少数据（Sql 语句或参数）在网络上传输的次数来节省时间。

（1）对于 Statement 的批量更新处理：

```
stm.addBatch(Sql);
    stm.addBatch(Sql);
int[] results=stm.executeBatch();
```

（2）对于 PreparedStatement 的批量跟新处理

```
pstm.setInt(1,11);pstm.setString(2,”haha”);……..
pstm.addBatch()
pstm.setInt(1,12);pstm.setString(2,”gaga”);……..
pstm.addBatch()
int[] results=stm.executeBatch();
```

注：int[] 中每一个数表示该 Sql 语句影响到的记录条数。

PreparedStatement 的更新操作比 Statement 的更新操作多了一个设置参数的过程。

## 八、 SQL3.0 规范中的新类型

**Array** 数组类型，主要用于保存一些类似于数组结构的数据。

**Sturct** 结构

**Blob**，大的二进制数据文件，最多存储 2G。

**Clob**，大文本文件对象，最多存储 2G。

在使用上述大对象的时候，在使用 JDBC 插入记录时要先插入一个空的占位对象，然后使用

**select blobdata from t_blob where id = " + id + " for update** 这样的语法来对获得的大对象，进行实际的写入操作 **Blod** 通过 **getBinaryOutputStream()**方法获取流进行写入。**getBinaryStream()**方法获得流来获取 **Blob** 中存储的数据。

**Clob** 的操作也和、**Blob** 相同。**getAsciiStream()** 方法用于读取存储的文本对象，**getAsciiOutputStream()**方法之获得流用来向文件对象写入的。

**BLOB 与 CLOB 的异同点：**

① 都可以存储大量超长的数据；

② BLOB (Binary Large Object) 以二进制格式保存于数据库中，特别适合保存图片、视频文件、音频文件、程序文件等；

③ CLOB (Character Large Object) 以 Character 格式保存于数据库中，适合保存比较长的文本文件。

# 九、 SQL 数据类型及其相应的 Java 数据类型

| SQL 数据类型 | Java 数据类型 | 说明 |
|---|---|---|
| INTEGER 或者 INT | int | 通常是个 32 位整数 |
| SMALLINT | short | 通常是个 16 位整数 |
| NUMBER(m,n) | Java.sql.Numeric | 合计位数是 m 的定点十进制数，小数后面有 n 位数 |
| DECIMAL(m,n) | 同上 | |
| DEC(m,n) | Java.sql.Numeric | 合计位数是 m 的定点十进制数，小数后面有 n 位数 |
| FLOAT(n) | double | 运算精度为 n 位二进制数的浮点数 |
| REAL | float | 通常是 32 位浮点数 |
| DOUBLE | double | 通常是 64 位浮点数 |
| CHAR(n) | String | 长度为 n 的固定长度字符串 |
| CHARACTER(n) | 同上 | |
| VARCHAR(n) | String | 最大长度为 n 的可变长度字符串 |
| BOOLEAN | boolean | 布尔值 |
| DATE | Java.sql.Date | 根据具体设备而实现的日历日期 |
| TIME | Java.sql.Time | 根据具体设备而实现的时戳 |
| TIMESTAMP | Java.sql.Timestamp | 根据具体设备而实现的当日日期和时间 |
| BLOB | Java.sql.Blob | 二进制大型对象 |
| CLOB | Java.sql.Clob | 字符大型对象 |
| ARRAY | Java.sql.Array | |

# 十、 面向对象的数据库设计

类的关联，继承在数据库中的体现：

类定义－－－>表定义

类属性－－－>表字段

类关系－－－>表关系

对　象－－－>表记录

注：　Oid（对象 id）－－－>业务无关

在数据库中每一条记录都对应一个唯一的 id；

**Id** 通常是用来表示记录的唯一性的，通常会使用业务无关的数字类型

字段的个数不会影响数据库的性能，表则越多性能越低。

## （一）类继承关系对应表，

1、为每一个类建一张表。通过父类的 Oid 来体现继承关系。

特点：在子类表中引用父类表的主建作为自己的外建。

优点：方便查询。属性没有冗余。支持多态。

缺点：表多，读写效率低。生成报表比较麻烦。

2、为每一个具体实现类建一个表

特点：父类的属性被分配到每一个子类表中。

优点：报表比较容易

缺点：如果父类发生改变会引起所有子了表随之更改。

并且不支持多态。数据有冗余。
3、所有的类在一张表中体现，加一个类型辨别字段
特点：效率高，查询不方便，用于重复字段不多时。
优点：支持多态，生成报表很简单。
缺点：如果任何一个类发生变化，必须改表。字段多，难以维护。

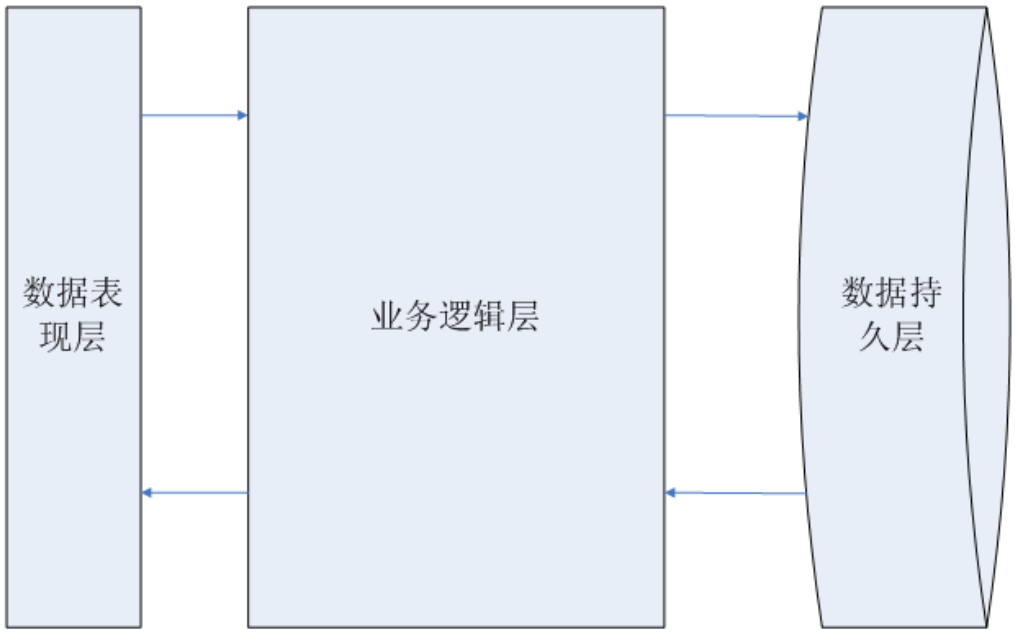
### （二）类关联关系对应表

1，一对一关联，类关系对应成表时有两种做法：
一是引用主键，也就是一方引用另一方的主键既作为外键有作为自身的主键。
二是外键引用，一方引用另一方的主键作为自身的外键，并且自己拥有主键。
2，一对多关联，也就是多端引用一端的主键当作外键，多端自身拥有主键。
3，多对多关系，多对多关系是通过中间表来实现的，中间表引用两表的主键当作联合主键，就可以实现多对多关联。

# 十一、 JDBC 应用的分层

## 1 项目分层开发

项目三层结构




**数据表现层：**

**数据的录入**
**数据的显示**

**业务逻辑层：**

**执行业务逻辑**

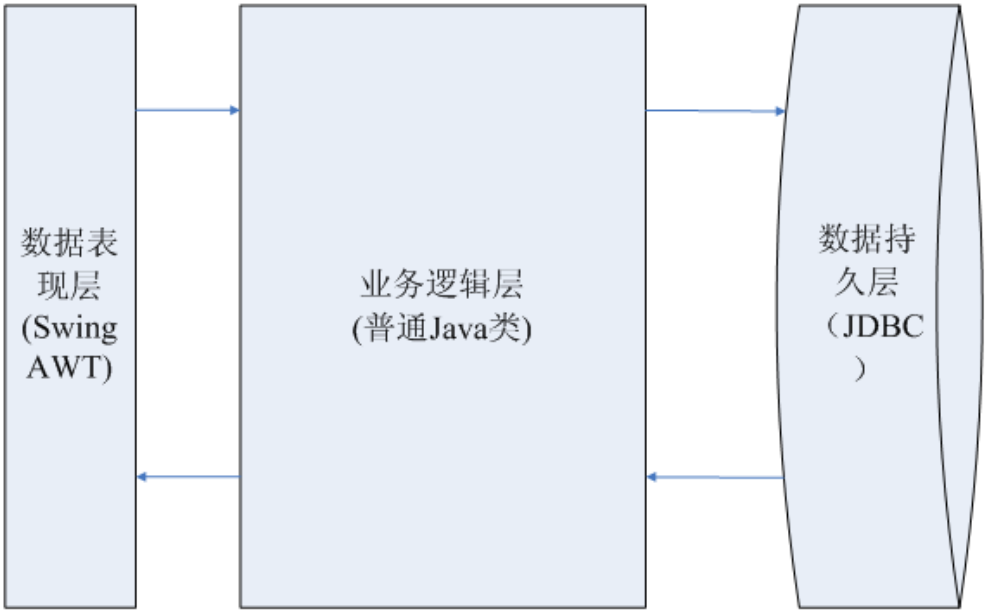
数据持久层：

系统中的数据与数据库中的数据进行交互

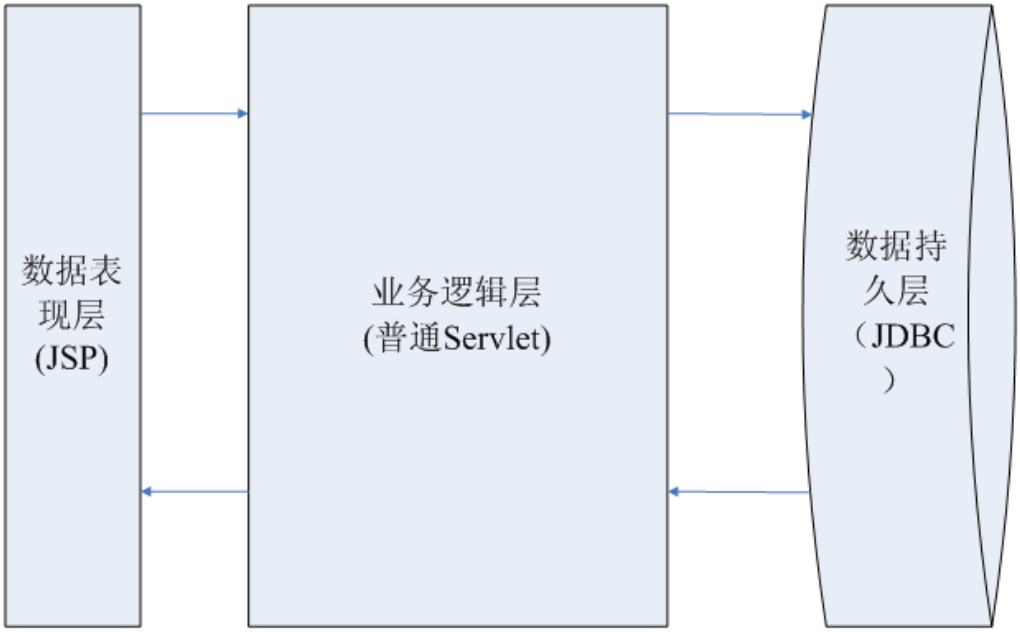
对数据库进行增删改查

# 2 Java 分层解决方案

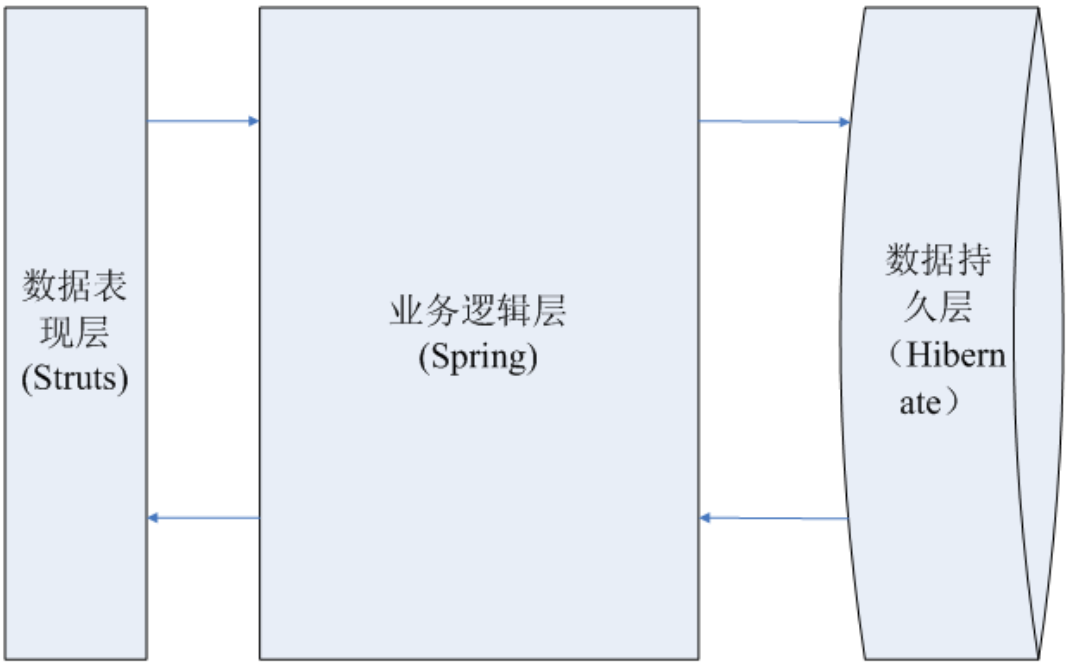
桌面应用三层架构的解决方案



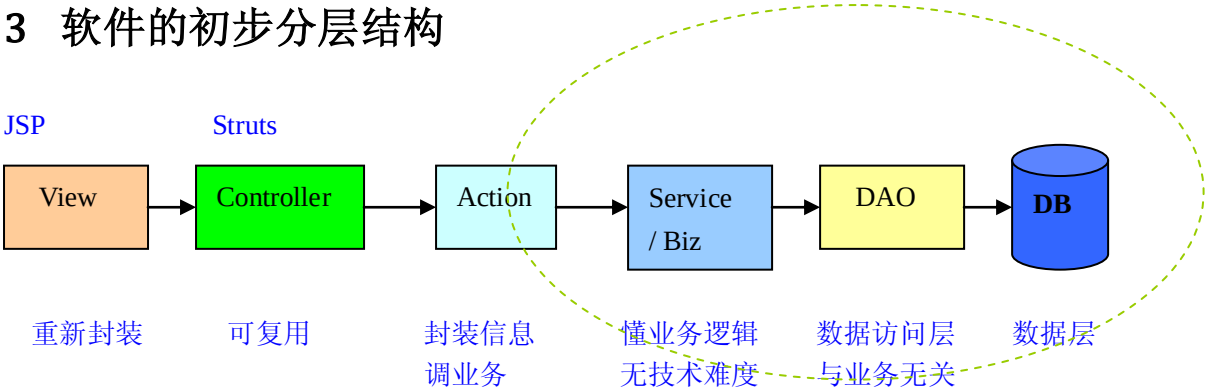

Web 应用三层架构的解决方案之 Sun 公司提供的基本应用解决方案



## Web 应用三层架构的解决方案之国内流行的开源解决方案




# 3 软件的初步分层结构




谁依赖谁就看谁调用谁。

软件的分层设计，便于任务的划分、降低层间的耦合。

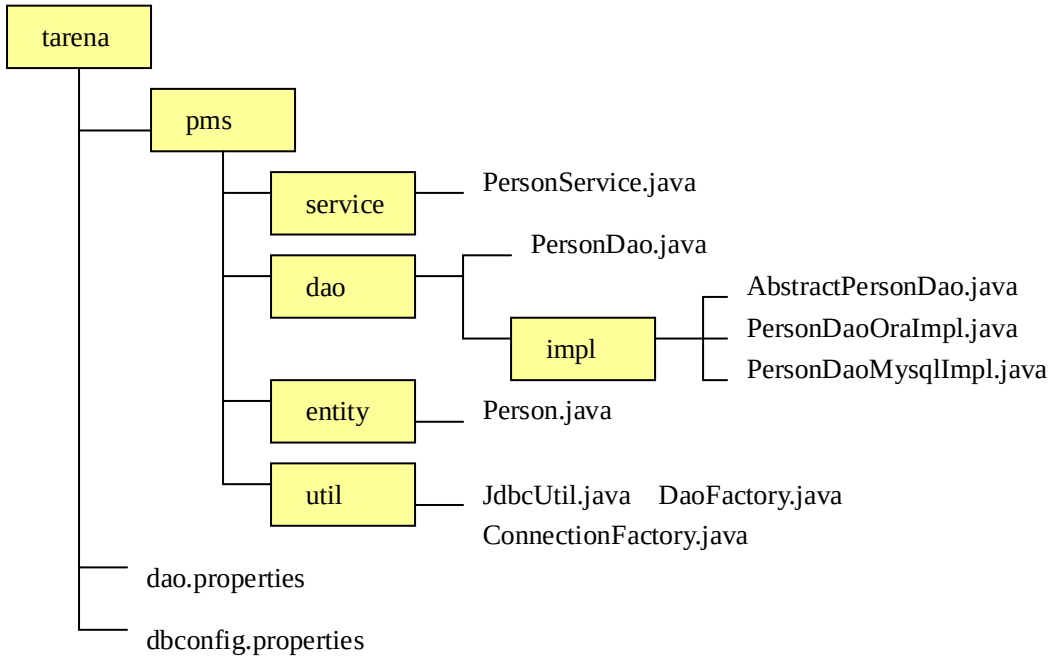
结合 PMS 的设计方法，思考这样分层的好处。

并且，使代码尽量减少重复，可复用性好，扩展余地加大，而且尽量减少硬编码。

**Model 部分**

**需求：实现对 Person 类的数据库持久化基本操作（CRUD）。**

包结构：




```
对时间的操作：
ps.setDate(1, Data.valueOf("2007-5-1"));
ps.setTimestamp(2, new Timestamp(System.currentTimeMillis()));          //对系统时间的截取
```

BS 架构和 CS 架构：
C－S 架构：两层体系结构，主要应用于局域网中。
B－S 架构：三层体系结构，表现层＋业务逻辑层＋数据存储层
注：层面越多，软件越复杂，但更灵活。分层是必须的但是要有个度。
层次一旦确定，数据必须按层访问，不能跨层访问。
层与层之间最好时单向依赖（单向调用）。

纵向划分：按功能划分。
横向划分：按抽象划分。

# 十二、 JDBC2.0 扩展

## 1 DataSourse（数据源）

定义:
1、包含了连接数据库所需的信息，可以通过数据源获得数据库连接，有时由于某些连接数据库的信息会变更，所以经常使用包含数据库连接信息的数据源。
2、一个标准的数据库连接工厂，作为 DriverManager 的替代项，保存与数据库相关的信息，可以将数据库的连接信息放在一个共享的空间进行提取，不用在本地安装。支持 JNDI 的绑定，支持连接池，支持分布式服务，用 getConnection 方法可获得与数据库的连接。数据源应该由管理员创建(目的是为了保证数据库的安全)。所以数据源对象一般放在 JNDI

服务器中。

通过 JNDI 获得绑定的资源

```
 public static Object lookup(String context) throws NamingException
    {
        Properties pro = new Properties();
    //Weblogic 的 JNDI 服务器参数
pro.put(InitialContext.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
        pro.put(InitialContext.PROVIDER_URL, "t3://localhost:7001");

        Context ctx = new InitialContext(pro);
        return ctx.lookup(context);//通过指定的字符串获得先前绑定的资源。
```

## 2 JNDI（命名目录服务器）

**定义:**是 Java 的命名目录服务器。而 JDBC 是 Java 的数据库访问接口。
跟 JDBC 是平级的关系，是两个独立的 JNDI；JDBC 存储的数据都是以二维表的接口来大规模存储数据。而 JNDI 存储的是差异性比较大的 Java 对象。JDBC 取数据时用 Sql 语言访问数据。JDBC API 依赖于驱动程序，而 JNDI 依赖于服务提供者。JDBC 一般把数据存储到关系型数据库，而 JNDI 一般把数据存储到小型数据库、文件、甚至是注册表中。
JNDI 相当于一个电话本。允许程序将一个对象和一个命名绑定到目录树上。
(JNDI 的方法是在 javax.naming 包下，
接口是 Context，实现类是 InitialContext)

**bind(String name, Object obj)** 将名称绑定到对象资源，建立指定的字符串和对象资源的关联
lookup(String name) ，通过指定的字符串获得先前绑定的资源
以下是将资源和 JNDI 命名绑定的方法

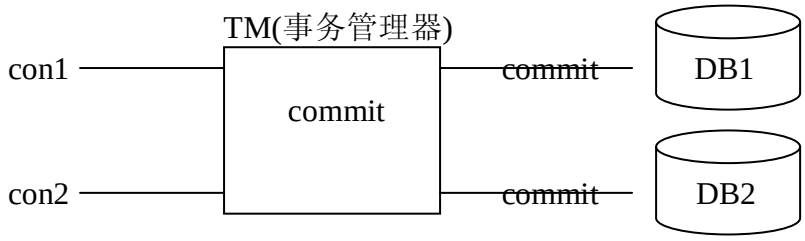
```
  public static void bind(String context, Object obj) throws NamingException
    {
        Properties pro = new Properties();
    //Weblogic 的 JNDI 服务器参数
pro.put(InitialContext.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
        pro.put(InitialContext.PROVIDER_URL, "t3://localhost:7001");

        Context ctx = new InitialContext(pro);
        ctx.bind(context, obj);//建立指定的字符串和对象资源的关联
    }
```

# 3 连接池

在内存中用来保存一个个数据库连接的对象。
访问数据库时，建立连接和拆连接需要花费较长时间，通过以连接池直连的方式获取连接，不需要注册驱动程序，可以大量的节省销毁和创建连接的资源消耗提高访问数据库的效率。
注：通过连接池获得的 Connection，当执行 con.close()时，不是关闭连接，而是表示将连接释放回连接池。连接池是一个很复杂的软件，所以是由服务器厂商实现。

# 4 分布式的事务管理器 JTA

分布式事务是通过多个异地数据库执行一组相关的操作，要保证原子操作的不可分，也不用再自己写 **commit**，和 **rollback**，全部都交给中间服务器(TM)来处理。（两阶段提交），也 就是在中间服务器发送 sql 语句等待数据库回应，都回应操作成功才提交，否则同时回滚。




1、regester
2、TM→execute()
3、commit→TM
4、TM→commit→DB

# 5 RowSet

行集，这是一个 JavaBean（事件机制），它增强了 ResultSet 的功能，包装了 Connection、Statement、ResultSet、DriverManage。通过 RowSet 可以获得数据源，设置隔离级别，也可以发送查寻语句，也实现了离线的操作遍历，RowSet 也支持预编译的 Statement。
RowSet 中的方法大致上和 ResultSet 相同，当需要使用时请查阅 JAVA API 参考文档。