# 第12章 JDBC

# JAVA数据库连接

JDBC（Java Data Base Connectivity,JAVA数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用Java语言编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使开发人员能够编写数据库应用程序。

程序类型

## 类型1:JDBC-ODBC桥

JDBC桥接器用于访问安装在每台客户机上的ODBC驱动程序。 使用ODBC需要在系统上配置表示目标数据库的数据源名称(DSN)。当Java第一次出现时，这是一个驱动程序，因为大多数数据库仅支持ODBC访问，但现在这种类型的驱动程序仅推荐用于实验性使用或没有其他替代方案时使用。

## 类型2:本地API驱动

JDBC API调用将转换为本地C/C++ API调用，这是数据库唯一的。 这些驱动程序通常由数据库供应商提供，并以与JDBC-ODBC桥接相同的方式使用。 必须在每个客户机上安装供应商特定的驱动程序。

如果要更改数据库，则必须更改原生API，因为它特定于数据库，并且现在大部分已经过时，但是使用类型2驱动程序实现了一些扩展功能的开发，它消除了ODBC的开销。

## 类型3:Net纯Java驱动

使用标准网络套接字与中间件应用程序服务器进行通信。套接字信息随后由中间件应用服务器转换成DBMS所需的调用格式，并转发到数据库服务器。

这种驱动程序是非常灵活的，因为它不需要在客户端上安装代码，一个驱动程序实际上可以提供多个数据库的访问。可以将应用程序服务器视为JDBC“代理”，它会调用客户端应用程序。 因此，我们需要了解应用程序服务器的配置，才能有效地使用此驱动程序类型。

## 类型4:100％纯Java驱动

基于纯Java的驱动程序通过套接字连接与供应商的数据库直接通信。这是数据库可用的最高性能驱动程序，通常由供应商自己提供。

这种驱动是非常灵活的，不需要在客户端或服务器上安装特殊的软件。MySQL Connector/J驱动程序是类型4驱动程序。

# 第一个示例

在写代码前，需要导入数据库驱动，即 mysql-connector-java.jar。

Class.forName("com.mysql.jdbc.Driver");

Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/jdbc", "root", "123456");

Statement st = conn.createStatement();

ResultSet rs = st.executeQuery("select name from user");

while (rs.next()) {

System.out.println(rs.getObject("name"));

}

rs.close();

st.close();

conn.close();

# 基本步骤

## 1-注册驱动（只做一次）

一：Class.forName(“com.mysql.jdbc.Driver”);

　　推荐这种方式，不会对具体的驱动类产生依赖。

二：DriverManager.registerDriver(new com.mysql.jdbc.Driver());　　 会造成DriverManager中产生两个一样的驱动，并会对具体的驱动类产生依赖。

## 2-建立连接（Connection）

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

URL用于标识数据库的位置，通过URL地址告诉JDBC程序连接哪个数据库，URL的写法为：

其他参数如：useUnicode=true&characterEncoding=utf8

## 3-创建执行SQL的语句（Statement）

Statement st = conn.createStatement();

st.executeQuery(“select \* from user”);

上面这种方式会存在SQL注入的危险。

String sql = “insert into user(name,pwd) values(?,?)”;

PreparedStatement ps = conn.preparedStatement(sql);

ps.setString(1, “col\_value”); //占位符顺序从1开始

ps.setString(2, “123456”); //也可以使用setObject

ps.executeQuery();

## 4-处理执行结果（ResultSet）

ResultSet rs = ps.executeQuery();

While(rs.next()){

rs.getString(“col\_name”);

rs.getInt(1);

//...

}

## 5-释放资源

数据库连接（Connection）非常耗资源，尽量晚创建，尽量早释放。

依次关闭对象及连接：ResultSet→Statement→Connection

分别调用各自的close方法进行关闭释放资源。

调用方法时需捕捉SQLException。

# API说明

JDBC API主要位于JDK中的java.sql包中（之后扩展的内容位于javax.sql包中），主要包括（接口需驱动程序提供者来具体实现）：

## DriverManager

负责加载各种不同驱动程序（Driver），并根据不同的请求，向调用者返回相应的数据库连接（Connection）。

Driver

由数据库厂家提供，作为JAVA开发人员，只需要使用Driver接口就可以了。在编程中要连接数据库，必须先装载特定厂商的数据库驱动程序，不同的数据库有不同的装载方法。

如：

装载MySql驱动：Class.forName("com.mysql.jdbc.Driver");

装载Oracle驱动：Class.forName("oracle.jdbc.driver.OracleDriver");

## Connection

数据库连接，负责与进行数据库间通讯，SQL执行以及事务处理都是在某个特定Connection环境中进行的。可以产生用以执行SQL的Statement。

Connection与特定数据库的连接（会话），在连接上下文中执行sql语句并返回结果。DriverManager.getConnection(url, user, password)方法建立在JDBC URL中定义的数据库Connection连接上。

如：

连接MySql数据库：Connection conn = DriverManager.getConnection("jdbc:mysql://host:port/database", "user", "password");

连接Oracle数据库：Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@host:port:database", "user", "password");

### Connection（URL）常用参数

user 数据库用户名（用于连接数据库）

password 用户密码（用于连接数据库）

useUnicode 是否使用Unicode字符集

characterEncoding 当useUnicode=true时，指定字符编码

autoReconnect 当连接异常中断时，是否自动重新连接，默认false

autoReconnectForPools 是否使用针对连接池的重连策略，默认false

failOverReadOnly 自动重连成功后，连接是否设置为只读，默认true

maxReconnects当autoReconnect=true时，重试连接的次数

initialTimeout当autoReconnect=true时，两次重连间隔（秒）

### Connection常用方法

createStatement()：创建向数据库发送sql的statement对象。

prepareStatement(sql) ：创建向数据库发送预编译sql的PrepareSatement对象。

prepareCall(sql)：创建执行存储过程的callableStatement对象。

setAutoCommit(boolean autoCommit)：设置事务是否自动提交。

commit() ：在链接上提交事务。

rollback() ：在此链接上回滚事务。

## Statement

用以执行SQL查询和更新（针对静态SQL语句和单次执行）。

由createStatement创建，用于发送简单的SQL语句（不带参数）。

### Statement常用方法

execute(String sql):运行语句，返回是否有结果集

executeQuery(String sql)：运行select语句，返回ResultSet结果集。

executeUpdate(String sql)：运行insert/update/delete操作，返回更新的行数。

addBatch(String sql) ：把多条sql语句放到一个批处理中。

executeBatch()：向数据库发送一批sql语句执行。

### PreparedStatement

用以执行包含动态参数的SQL查询和更新（在服务器端编译，允许重复执行以提高效率）。

继承自Statement接口，由preparedStatement创建，用于发送含有一个或多个参数的SQL语句。PreparedStatement对象比Statement对象的效率更高，并且可以防止SQL注入，所以一般都使用PreparedStatement。

### CallableStatement

继承自PreparedStatement接口，由方法prepareCall创建，用于调用存储过程。

## ResultSet

结果集(ResultSet)是数据中查询结果返回的一种对象，可以说结果集是一个存储查询结果的对象，但是结果集并不仅仅具有存储的功能，他同时还具有操纵数据的功能，可能完成对数据的更新等。

结果集读取数据的方法主要是getXXX()，他的参数可以是整型表示第几列（是从1开始的），还可以是列名。所有的getXXX方法都是对当前行进行操作。

### ResultSet常用方法

提供检索不同类型字段的方法，常用方法有：

getString(int index)、getString(String columnName)：获得在数据库里是varchar、char等类型的数据对象。

getFloat(int index)、getFloat(String columnName)：获得在数据库里是Float类型的数据对象。

getDate(int index)、getDate(String columnName)：获得在数据库里是Date类型的数据。

getBoolean(int index)、getBoolean(String columnName)：获得在数据库里是Boolean类型的数据。

getObject(int index)、getObject(String columnName)：获取在数据库里任意类型的数据。

还提供了对结果集进行滚动的方法：

next()：移动到下一行

Previous()：移动到前一行

absolute(int row)：移动到指定行

beforeFirst()：移动resultSet的最前面。

afterLast() ：移动到resultSet的最后面。

### ResultSet分类

1、最基本的ResultSet

之所以说是最基本的ResultSet是因为这个ResultSet它起到的作用就是完成了查询结果的存储功能，而且只能读取一次，不能够来回的滚动读取。这种结果集的创建方式如下:

Statement st = conn.CreateStatement()

ResultSet rs = Statement.excuteQuery(sqlStr);

由于这种结果集不支持滚动的读取功能,所以如果获得这样一个结果集，只能使用它里面的next()方法，逐个的读取数据。

2、可滚动的ResultSet

这个类型支持前后滚动取得纪录next()、previous()，回到第一行first()，同时还支持要取的ResultSet中的第几行 absolute(int n)，以及移动到相对当前行的第几行relative(int n)，要实现这样的ResultSet在创建Statement时用如下的方法。

Statement st =conn.createStatement(int resultSetType, int resultSetConcurrency);

ResultSet rs = st.executeQuery(sqlStr);

其中两个参数的意义是：

resultSetType是设置ResultSet对象的类型标示可滚动，或者是不可滚动。取值如下:

ResultSet.TYPE\_FORWARD\_ONLY只能向前滚动

ResultSet.TYPE\_SCROLL\_INSENSITIVE和Result.TYPE\_SCROLL\_SENSITIVE这两个方法都能够实现任意的前后滚动,使用各种移动的ResultSet指针的方法.二者的区别在于前者对于修改不敏感,而后者对于修改敏感。

resultSetConcurency是设置ResultSet对象能够修改的,取值如下:

ResultSet.CONCUR\_READ\_ONLY 设置为只读类型的参数.

ResultSet.CONCUR\_UPDATABLE 设置为可修改类型的参数.

所以如果只是想要可以滚动的类型的Result只要把Statement如下赋值就行了。

Statement st = conn.createStatement(Result.TYPE\_SCROLL\_INSENITIVE,ResultSet.CONCUR\_READ\_ONLY);

ResultSet rs = st.excuteQuery(sqlStr);

用这个Statement执行的查询语句得到的就是可滚动的ResultSet。

3、可更新的ResultSet

这样的ResultSet对象可以完成对数据库中表的修改，但是我知道ResultSet只是相当于数据库中表的视图，所以并不是所有的ResultSet只要设置了可更新就能够完成更新的，能够完成更新的ResultSet的SQL语句必须要具备如下的属性：

a、只引用了单个表。

b、不含有join或者group by子句。

c、那些列中要包含主关键字。

具有上述条件的，可更新的ResultSet可以完成对数据的修改，可更新的结果集的创建方法是：

Statement st =createstatement(Result.TYPE\_SCROLL\_INSENSITIVE,Result.CONCUR\_UPDATABLE)

这样的Statement的执行结果得到的就是可更新的结果集。更新的方法是，把ResultSet的游标移动到你要更新的行，然后调用 updateXXX()，这个方法XXX的含义和getXXX()是相同的。updateXXX（）方法有两个参数，第一个是要更新的列，可以是列名或者 序号。第二个是要更新的数据，这个数据类型要和XXX相同。每完成对一行的update要调用updateRow()完成对数据库的写入，而且是在 ResultSet的游标没有离开该修改行之前，否则修改将不会被提交。

使用updateXXX方法还可以完成插入操作。但是首先要介绍两个方法：

moveToInsertRow()是把ResultSet移动到插入行，这个插入行是表中特殊的一行，不需要指定具体那一行，只要调用这个方法系统会自动移动到那一行的。

moveToCurrentRow()这是把ResultSet移动到记忆中的某个行，通常当前行。如果没有使用insert操作，这个方法没有什么效果，如果使用了insert操作，这个方法用于返回到insert操作之前的那一行，离开插入行，当然也可以通过next(),previous()等方法离开插入行。

要完成对 数据库的插入，首先调用moveToInsertRow()移动到插入行，然后调用updateXXX的方法完成对各列数据的更新，完成更新后和更新操作一样，要写到数据库，不过这里使用的是insertRow()，也要保证在该方法执行之前ResultSet没有离开插入列，否则插入不被执行，并且对插入行的更新将丢失。

4、可保持的ResultSet

正常情况下如果使用Statement执行完一个查询，又去执行另一个查询时这时候第一个查询的结果集就会被关闭，也就是说，所有的Statement的查询对应的结果集是一个，如果调用Connection的commit()方法 也会关闭结果集。可保持性就是指当ResultSet的结果被提交时，是被关闭还是不被关闭。要完成这样的ResultSet的对象的创建，要使用的Statement的创建要具有三个参数，这个Statement的创建方式也就是，所说的Statement的第三种创建方式。

Statement st = conn.createStatement(int resultsetscrollable,int resultsetupdateable,intresultsetSetHoldability)

ResultSet rs = st.excuteQuery(sqlStr);

前两个参数和createStatement方法中的参数是完全相同的，这里只介绍第三个参数:

resultSetHoldability表示在结果集提交后结果集是否打开，取值有:

ResultSet.HOLD\_CURSORS\_OVER\_COMMIT:表示修改提交时,不关闭数据库.

ResultSet.CLOSE\_CURSORS\_AT\_COMMIT:表示修改提交时ResultSet关闭.

## SQLException

代表在数据库连接的建立和关闭和SQL语句的执行过程中发生了例外情况（即异常）。

# 批处理Batch

简单实例1

conn.setAutoCommit(false); //设为手动提交

stmt = conn.createStatement();

for (int i = 0; i < 20000; i++) {

stmt.addBatch("insert into table1(name,pwd)values('name" + i + "',123456)");

}

stmt.executeBatch();

conn.commit(); //提交事务

简单实例2

conn.setAutoCommit(false); //JDBC中默认是true，自动提交事务

ps1 = conn.prepareStatement("insert into t\_table1(name,pwd)values(?,?)"); //事务开始

ps1.setObject(1, "小高");

ps1.setObject(2, "123");

ps1.execute();

conn.commit();

# 元数据

## DataBaseMetaData

元数据：在改元数据中定义了一些关于数据库、表、列的信息。

获取对象

DatabaseMetaData cmd = conn.getMetaData();

相关方法

DataBaseMetaData对象

getURL()：返回一个String类对象，代表数据库的URL。

getUserName()：返回连接当前数据库管理系统的用户名。

getDatabaseProductName()：返回数据库的产品名称。

getDatabaseProductVersion()：返回数据库的版本号。

getDriverName()：返回驱动驱动程序的名称。

getDriverVersion()：返回驱动程序的版本号。

isReadOnly()：返回boolean值，指示数据库是否只允许读操作。

## ParameterMetaData

该元数据中主要是包含了一些参数的信息。

获取对象

PreparedStatement pst = conn.prepareStatement(“”);

ParameterMetaData pmd = pst.getParameterMetaData();

相关方法

ParameterMetaData对象

getParameterCount()：获得指定参数的个数

getParameterType(int param)：获得指定参数的sql类型

## ResultSetMetaData

该元数据中保存了一些关于数据查询结果的相关信息。

获取对象

ResultSet rs = pst.executeQuery();

ResultSetMetaData rmd = rs.getMetaData();

相关方法

ResultSetMetaData对象

getColumnCount():返回resultset对象的列数

getColumnName(int column):获得指定列的名称

getColumnTypeName(int column):获得指定列的类型

# CLOB文本大对象操作

简单示例

ps = conn.prepareStatement("insert into table1(name,info)values(?,?)");

ps.setString(1, "小高");

//将文本文件内容直接输入到数据库中

//ps.setClob(2, new FileReader(new File("D:/a.txt")));

//将程序中的字符串输入到数据库中的CLOB字段中

ps.setClob(2, new BufferedReader(new InputStreamReader(new ByteArrayInputStream("aaaa".getBytes()))));

ps.executeUpdate();

ps2 = conn.prepareStatement("select \* from table1 where id=?");

ps2.setObject(1, 223021);

rs = ps2.executeQuery();

while (rs.next()) {

Clob c = rs.getClob("info");

r = c.getCharacterStream();

int temp = 0;

while ((temp=r.read())!=-1) {

System.out.print((char)temp);

}

}

# BLOB二进制大对象的使用

简单示例

ps = conn.prepareStatement("insert into table1(name,image)values(?,?)");

ps.setString(1, "小高");

ps.setBlob(2, new FileInputStream("G:/d.jpg"));

ps.execute();

ps2 = conn.prepareStatement("select \* from table1 where id=?");

ps2.setObject(1, 223024);

rs = ps2.executeQuery();

while (rs.next()) {

Blob b = rs.getBlob("image");

is = b.getBinaryStream();

os = new FileOutputStream("G:/h.jpg");

int temp = 0;

while ((temp=is.read())!=-1) {

os.write(temp);

}

}

# 资源文件properties

db.properties

mysqlDriver=com.mysql.jdbc.Driver

mysqlURL=jdbc:mysql://localhost:3306/testjdbc

mysqlUser=root

mysqlPwd=mysql

# 简单封装JDBCUtil.java

Properties pros = new Properties();

pros.load(Thread.currentThread().getContextClassLoader().getResourceAsStream("db.properties"));

Class.forName(pros.getProperty("mysqlDriver"));

return DriverManager.getConnection(pros.getProperty("mysqlURL"),pros.getProperty("mysqlUser"),pros.getProperty("mysqlPwd"));

# 数据库连接池

## 基本概念

用池来管理Connection，可以重复使用Connection。有了池就不用自己来创建Connection，而是通过池来获取Connection对象。当使用完Connection后，调用Connection的close()方法也不会真的关闭Connection，而是把Connection“归还”给池。池就可以再利用这个Connection对象了。

## 主要优点

（1） 缩短了连接创建时间

创建新的JDBC连接会导致联网操作和一定的JDBC驱动开销，如果这类连接是“循环”使用的，使用该方式，可避免这类不利因素。

（2）简化的编程模型

使用连接池技术时，每个单独线程能够像创建了自己的JDBC连接那样进行操作，从而允许使用直接的JDBC编程技术。

（3）受控的资源使用

如果不使用连接池技术，而是在每次需要时为线程创建新的连接，那么应用程序的资源使用将十分浪费，而且在负载较重的情况下会导致无法预期的结果。

池基本参数

（所有池参数都有默认值）：

初始大小：10个

最小空闲连接数：3个

增量：一次创建的最小单位（5个）

最大空闲连接数：12个

最大连接数：20个

最大的等待时间：1000毫秒

## 四大连接参数

连接池也是使用四大连接参数来完成创建连接对象。

分别是：驱动类，连接地址，用户名，密码。

## 实现接口

连接池必须实现：javax.sql.DataSource接口。各厂商可以让自己的连接池实现这个接口。这样应用程序可以方便的切换不同厂商的连接池.

连接池返回的Connection对象，它的close()方法与众不同，调用它的close()不是关闭，而是把连接归还给池，代表该链接不被占用了。

## 常见连接池

### DBCP

DataBase connection pool数据库连接池:是Apache上的java连接池项目，也是tomcat使用的连接池组件。需要3个依赖：common-dbcp.jar,common-pool.jar,common-collections.jar由于建立数据库连接是一个非常耗时耗资源的行为，所以通过连接池预先同数据库建立一些连接，放在内存中，应用程序需要建立数据库连接时直接到连接池中申请一个就行，用完后再放回去。dbcp没有自动的去回收空闲连接的功能。

DBCP常用方法

BasicDataSource dataSouce = new BasicDataSource();//连接池核心类

dataSouce.setUrl("jdbc:mysql://localhost:3306/mydb1");

dataSouce.setDriverClassName("com.mysql.jdbc.Driver");

dataSouce.setUsername("root");

dataSouce.setPassword("root");

dataSouce.setInitialSize(3); // 初始化连接

dataSouce.setMaxActive(6); // 最大连接

Connection con = dataSouce.getConnection();

### C3P0

C3P0是一个开源的JDBC连接池，它实现了数据源和JNDI绑定，支持JDBC3规范和JDBC2的标准扩展。c3p0是异步操作的，缓慢的JDBC操作通过帮助进程完成。扩展这些操作可以有效的提升性能。目前使用它的开源项目有Hibernate，Spring等。c3p0有自动回收空闲连接功能。

依赖：c3p0-0.9.5.2.jar

C3P0常用方法

ComboPooledDataSource ds = new ComboPooledDataSource();

ds.setJdbcUrl("jdbc:mysql://localhost:3306/mydb1");

ds.setUser("root");

ds.setPassword("123");

ds.setDriverClass("com.mysql.jdbc.Driver");

ds.setPassword("admin");//设置连接数据库的密码

ds.setMaxPoolSize(40);//设置连接池的最大连接数

ds.setMinPoolSize(10);//设置连接池的最小连接数

ds.setInitialPoolSize(10);//设置连接池的初始连接数

ds.setMaxStatements(100);//设置连接池的缓存Statement的最大数 //当连接池中的连接用完时，C3P0一次性创建新连接的数目2

ds.setAcquireIncrement(2);

//acquireRetryAttempts：定义在从数据库获取新连接失败后重复尝试获取的次数，默认为30秒

ds.setAcquireRetryAttempts(30);

//两次连接中间隔时间，单位毫秒，默认为1000;

ds.setAcquireRetryDelay(3000);

//连接关闭时默认将所有未提交的操作回滚。默认为false；

ds.setAutoCommitOnClose(false);

//连接池数据源

//ds.setConnectionPoolDataSource(ConnectionPoolDataSource 对象);

//idleConnectionTestPeriod：隔多少秒检查所有连接池中的空闲连接，默认为0表示不检查

ds.setIdleConnectionTestPeriod(10);

//初始化时创建的连接数，应在minPoolSize与maxPoolSize之间取值。默认为3；

ds.setInitialPoolSize(5);

//最大空闲时间，超过空闲时间的连接将被丢弃。为0或负数则永不丢弃。默认为0；

ds.setMaxIdleTime(3);

//连接池中保留的最大连接数。默认为15；

ds.setMaxPoolSize(100);

Connection con = ds.getConnection();

c3p0用法（XML）

c3p0-config.xml

<?xml version="1.0" encoding="UTF-8"?>

<c3p0-config>

<default-config>

<property name="jdbcUrl">jdbc:mysql://localhost:3306/db</property>

<property name="driverClass">com.mysql.jdbc.Driver</property>

<property name="user">root</property>

<property name="password">123</property>

<property name="acquireIncrement">3</property>

<property name="initialPoolSize">10</property>

<property name="minPoolSize">2</property>

<property name="maxPoolSize">10</property>

</default-config>

<named-config name="oracle">

<property name="jdbcUrl">jdbc:mysql://localhost:3306/mydb1</property>

<property name="driverClass">com.mysql.jdbc.Driver</property>

<property name="user">root</property>

<property name="password">123</property>

<property name="acquireIncrement">3</property>

<property name="initialPoolSize">10</property>

<property name="minPoolSize">2</property>

<property name="maxPoolSize">10</property>

</named-config>

</c3p0-config>

//ComboPooledDataSource ds = new ComboPooledDataSource();

ComboPooledDataSource ds = new ComboPooledDataSource("orcale");

Connection con = ds.getConnection();

# 三层架构

## 三层架构简介

三层架构(3-tier architecture) 通常意义上的三层架构就是将整个业务应用划分为：

界面层（User Interface layer）

业务逻辑层（Business Logic Layer）

数据访问层（Data access layer）

区分层次的目的即为了“高内聚低耦合”的思想。在软件体系架构设计中，分层式结构是最常见，也是最重要的一种结构。

原理

三个层次中，系统主要功能和业务逻辑都在业务逻辑层进行处理。

所谓三层体系结构，是在客户端与数据库之间加入了一个“中间层”，也叫组件层。这里所说的三层体系，不是指物理上的三层，不是简单地放置三台机器就是三层体系结构，也不仅仅有B/S应用才是三层体系结构，三层是指逻辑上的三层，即把这三个层放置到一台机器上。

三层体系的应用程序将业务规则、数据访问、合法性校验等工作放到了中间层进行处理。通常情况下，客户端不直接与数据库进行交互，而是通过与中间层建立连接，再经由中间层与数据库进行交互。

## 各层作用

1：数据访问层：主要是对非原始数据（数据库或者文本文件等存放数据的形式）的操作层，而不是指原始数据，也就是说，是对数据库的操作，而不是数据，具体为业务逻辑层或表示层提供数据服务．

2：业务逻辑层：主要是针对具体的问题的操作，也可以理解成对数据层的操作，对数据业务逻辑处理，如果说数据层是积木，那逻辑层就是对这些积木的搭建。

3：界面层：主要是WEB方式（如jsp），如果逻辑层相当强大和完善，无论表现层如何定义和更改，逻辑层都能完善地提供服务。

## 优缺点

优点

1、开发人员可以只关注整个结构中的其中某一层；

2、可以很容易的用新的实现来替换原有层次的实现；

3、可以降低层与层之间的依赖；

4、有利于标准化；

5、利于各层逻辑的复用。

6、结构更加的明确

7、在后期维护的时候，极大地降低了维护成本和维护时间

缺点

1、降低了系统的性能。这是不言而喻的。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。

2、有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。

3、增加了开发成本。

# 思途心得

1. JDBC程序类型、基本实现步骤、常见API、四大参数、批处理等。
2. PreparedStatement和Statement区别。
3. 常用开源连接池DBCP和C3P0的基本使用。
4. 三层架构设计思想。

# 作业

将学生信息管理系统使用JDBC来实现，完成基本的增删改查操作。