# JDBC架构

JDBC，Java数据库连接(Java Database Connectivity)，是一个独立于特定数据库管理系统、通用的SQL数据库存取和操作的公共接口（一组API），定义了用来访问数据库的标准Java类库，使用这个类库可以以一种标准的方法、方便地访问数据库资源。

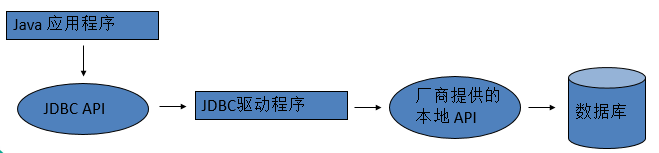
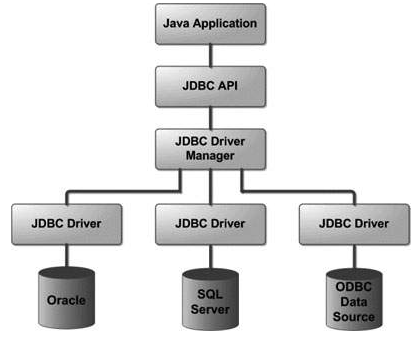
JDBC为访问不同的数据库提供了一种统一的途径，为开发者屏蔽了一些细节问题。

JDBC的目标是使Java程序员 使用JDBC可以连接任何提供了JDBC驱动程序的数据库系统，这样就使得程序员无需对特定的数据库系统的特点有过多的了解，从而大大简化和加快了开发过程。

JDBC API 是一系列的接口，在JDBC库中定义一些与数据库操作相关的API，如：

* 连接到数据库
* 创建SQL或MySQL语句
* 在数据库中执行SQL或MySQL查询
* 查看和修改结果记录
* 等等...

以下是架构图，它显示了驱动程序管理器相对于JDBC驱动程序和Java应用程序的位置



# 示例

**public class** JdbcConnectDBTest {  
 String **driver** = **"com.mysql.jdbc.Driver"**;  
 String **url** = **"jdbc:mysql://localhost:3306/test"**;  
 String **username** = **"root"**;  
 String **password** = **"123456"**;  
  
 @Test  
 **public void** testJDBC() **throws** SQLException, ClassNotFoundException {  
 *// 根据JDBC驱动名加载对应的数据库驱动* Class.*forName*(**driver**);  
 *// 加载完驱动后，就可以使用驱动管理器创建Connection对象了* Connection connection = DriverManager.getConnection(**url**, **username**, **password**);  
 *// 创建sql执行对象* Statement statement = connection.createStatement();  
  
 *// 将SQL语句提交到数据库,并且返回执行结果* ResultSet resultSet = statement.executeQuery(**"SELECT \* FROM USER"**);  
 **while** (resultSet.next()) {  
 System.***out***.println(**"name: "** + resultSet.getString(**"name"**));  
 }  
  
 *//关闭相关的对象* resultSet.close();  
 statement.close();  
 connection.close();  
  
 }  
  
}

# SQL语言的分类

SQL语言共分为四大类：数据查询语言DQL，数据操纵语言DML，数据定义语言DDL，数据控制语言DCL。

**1. 数据查询语言DQL**

数据查询语言DQL基本结构是由SELECT子句，FROM子句，WHERE子句组成的查询块：  
SELECT <字段名表> FROM <表或视图名> WHERE <查询条件>  
  
**2 .数据操纵语言DML**  
 数据操纵语言DML主要有三种形式：  
1) 插入：INSERT  
2) 更新：UPDATE  
3) 删除：DELETE  
  
**3. 数据定义语言DDL**  
 数据定义语言DDL用来创建数据库中的各种对象-----表、视图、索引、同义词、聚簇等如：CREATE TABLE/VIEW/INDEX/SYN/CLUSTER，分别表示：表/视图/索引/同义词/簇。DDL操作是隐性提交的！不能rollback   
  
**4. 数据控制语言DCL**  
 数据控制语言DCL用来授予或回收访问数据库的某种特权，并控制数据库操纵事务发生的时间及效果，对数据库实行监视等。如：  
1) GRANT：授权。  
2) ROLLBACK [WORK] TO [SAVEPOINT]：回退到某一点。  
 回滚---ROLLBACK  
 回滚命令使数据库状态回到上次最后提交的状态。其格式为：SQL>ROLLBACK;  
3) COMMIT [WORK]：提交。

# 常见的JDBC组件

**JDBC接口（API）包括两个层次：**

面向数据库的API：Java Driver API，供开发商开发数据库驱动程序用。

面向应用程序的API：Java API，抽象接口，供应用程序开发人员使用（连接数据库，执行SQL语句，获得结果）。

## DriverManager类

DriverManager是驱动程序管理器类，负责管理驱动程序，DriverManager使用通信子协议，将来自java应用程序的连接请求与适当的数据库驱动程序进行匹配。在JDBC下识别某个子协议的第一个驱动程序将用于建立数据库连接。

通常有两种方式去注册数据库驱动程序，而我们一般不显式调用 DriverManager 类的registerDriver() 方法来注册驱动程序类的实例，因为 Driver 接口的驱动程序类都包含了静态代码块，在这个静态代码块中，会调用 DriverManager.registerDriver() 方法来注册自身的一个实例。

#### DriverManager注册驱动的两种方式

在使用程序之前，必须先注册该驱动程序。 注册驱动程序是将相应的数据库驱动程序的类文件加载到内存中的过程，因此可以将其用作JDBC接口的实现。只需在程序中一次注册就可以。通常有两种方式去注册数据库驱动程序，如：

DriverManager.*registerDriver*(oracle.jdbc.driver.OracleDriver());  
Class.*forName*(**"com.mysql.jdbc.Driver"**);

**方法I - Class.forName()**

注册驱动程序最常见的方法是使用Java的Class.forName()方法，将驱动程序的类文件动态加载到内存中，并将其自动注册。这个方法是推荐使用的方法，因为它使驱动程序注册可配置和便携。以下示例使用Class.forName()注册Oracle驱动程序：

Class.*forName*(**"com.mysql.jdbc.Driver"**);  
Connection conn = **null**;  
conn = DriverManager.*getConnection*(**"jdbc:mysql://hostname:port/db\_name"**,**"db\_username"**, **"db\_password"**);  
conn.close();

**方法II - DriverManager.registerDriver()**

第二种方法是使用静态DriverManager.registerDriver()方法来注册驱动程序。如果使用的是非JDK兼容的JVM(如Microsoft提供的)，则应使用registerDriver()方法。以下示例使用registerDriver()注册Oracle驱动程序：

**try** {  
 Driver myDriver = **new** oracle.jdbc.driver.OracleDriver();  
 DriverManager.registerDriver(myDriver);  
}**catch**(ClassNotFoundException ex) {  
 System.***out***.println(**"Error: unable to load driver class!"**);  
 System.*exit*(1);  
}

## Driver 接口：JDBC驱动程序接口

java.sql.Driver 是JDBC API中定义的接口，所有 JDBC 驱动程序需要实现该接口。Java中使用这个接口对不同的数据库厂商进行规范，这个接口是提供给数据库厂商使用的，不同数据库厂商提供不同的实现，此接口用于与数据库服务器进行交互，我们很少会直接与Driver对象进行交互，即在应用程序中我们不需要直接去访问实现了 Driver 接口的类，这些类是由驱动管理器类（DriverManager）进行管理的，通常我们使用 Class.forName(driverName) 这段代码，向DriverManager注册相应的数据库驱动器，然后通过该DriverManager去调用这些Driver实现。

例如，使用JDBC驱动程序，可以通过发送SQL或数据库命令，然后使用Java接收结果来打开数据库连接并与数据库进行交互。JDK附带的Java.sql包中包含各种类，其类的行为被定义，实现在第三方驱动程序中完成。 第三方供应商在其数据库驱动程序中实现java.sql.Driver接口。

### JDBC驱动程序类型

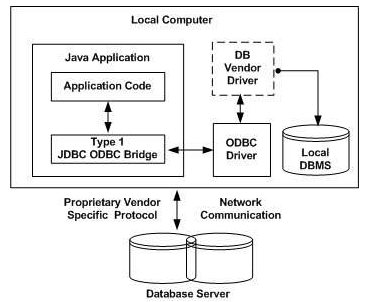
JDBC驱动程序是各个数据库厂商根据JDBC的规范制作的 JDBC 实现类的类库，它因Java运行的各种操作系统和硬件平台而异，Sun将实现类型分为四种类型：

* 第一类：JDBC-ODBC桥驱动程序。
* 第二类：部分本地API部分Java的驱动程序。
* 第三类：JDBC网络纯Java驱动程序。
* 第四类：本地协议的纯 Java 驱动程序。

其中，三四类都是纯Java的驱动程序，因此，对于Java开发者来说，它们在性能、可移植性、功能等方面都有优势。

#### 类型1：JDBC-ODBC桥驱动程序

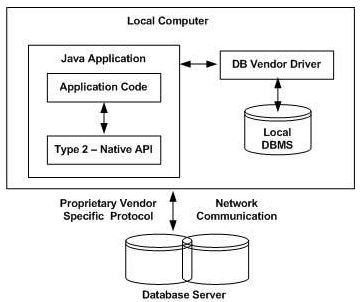
在类型1驱动程序中，JDBC桥接器用于访问安装在每台客户机上的ODBC驱动程序。 使用ODBC需要在系统上配置表示目标数据库的数据源名称(DSN)。当Java第一次出现时，这是一个驱动程序，因为大多数数据库仅支持ODBC访问，但现在这种类型的驱动程序仅推荐用于实验性使用或没有其他替代方案时使用。



JDK 1.2附带的JDBC-ODBC桥接是这种驱动程序的一个很好的例子。

#### 类型2：JDBC本地API

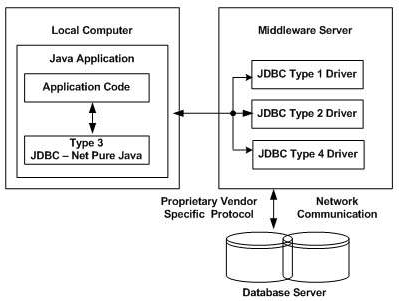
在类型2驱动程序中，JDBC API调用将转换为本地C/C++ API调用，这是数据库唯一的。 这些驱动程序通常由数据库供应商提供，并以与JDBC-ODBC桥接相同的方式使用。 必须在每个客户机上安装供应商特定的驱动程序。如果要更改数据库，则必须更改原生API，因为它特定于数据库，并且现在大部分已经过时，但是使用类型2驱动程序实现了一些扩展功能的开发，它消除了ODBC的开销。



Oracle调用接口(OCI)驱动程序是类型2驱动程序的示例。

#### 类型3：JDBC-Net纯Java

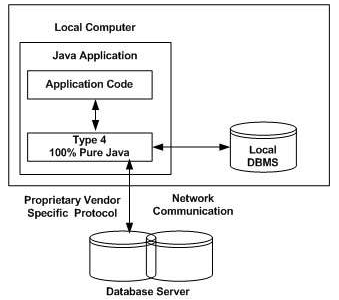
在类型3驱动程序中，使用三层方法访问数据库。 JDBC客户端使用标准网络套接字与中间件应用程序服务器进行通信。 套接字信息随后由中间件应用服务器转换成DBMS所需的调用格式，并转发到数据库服务器。这种驱动程序是非常灵活的，因为它不需要在客户端上安装代码，一个驱动程序实际上可以提供多个数据库的访问。



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。应用程序服务器可能会使用类型1,2或4驱动程序与数据库通信，了解细微差别对理解JDBC是有帮助的。

#### 类型4：100％纯Java

在类型4驱动程序中，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。 此外，这些驱动程序可以动态下载。



MySQL Connector/J驱动程序是类型4驱动程序。 由于其网络协议的专有性质，数据库供应商通常提供类型4驱动程序。

#### 应该使用哪个驱动程序？

如果您正在访问一种类型的数据库，例如Oracle，Sybase或IBM DB2，则首选类型4的驱动程序；

如果Java应用程序同时访问多种类型的数据库，则类型3是首选驱动程序；

类型2驱动程序是在数据库不可用的类型3或类型4驱动程序的情况下使用；

类型1驱动程序不被视为部署级驱动程序，通常仅用于开发和测试目的。

## Connection接口

连接对象(Connection)表示通信上下文，就是与数据库的所有通信仅通过连接对象。加载驱动程序后，可以使用DriverManager.getConnection()方法建立连接。为了方便参考，这里列出三个重载的DriverManager.getConnection()方法：

* getConnection(String url)
* getConnection(String url, Properties prop)
* getConnection(String url, String user, String password)

这里每个方法都需要一个数据库URL，数据库URL是指向数据库的地址。

### 创建连接对象

上面列出了三种形式的DriverManager.getConnection()方法来创建一个连接对象。

**使用具有用户名和密码的数据库URL**

String **url** = **"jdbc:mysql://localhost:3306/test"**;  
String **username** = **"root"**;  
String **password** = **"123456"**;

Connection connection = DriverManager.*getConnection*(**url**,**username**,**password**);

**仅使用数据库URL**

Connection connection = DriverManager.*getConnection*(**"jdbc:mysql://localhost:3306/test?user=root&password=123456"**);

**使用数据库URL和Properties对象**

Properties对象包含一组键-值对。 在调用getConnection()方法时，它用于将驱动程序属性传递给驱动程序。要进行与上述示例相同的连接，请使用以下代码：

String URL = **"jdbc:mysql://localhost:3306/test"**;  
Properties info = **new** Properties( );  
info.put( **"user"**, **"root"** );  
info.put( **"password"**, **"123456"** );  
Connection connection = DriverManager.*getConnection*(URL, info);

### 关闭JDBC连接

在JDBC程序结束之后，显式地需要关闭与数据库的所有连接以结束每个数据库会话。 但是，如果在编写程序中忘记了关闭也没有关系，Java的垃圾收集器将在清除过时的对象时也会关闭这些连接。依靠垃圾收集，特别是数据库编程，是一个非常差的编程实践。所以应该要使用与连接对象关联的close()方法关闭连接。要确保连接已关闭，可以将关闭连接的代码中编写在“finally”块中。 一个finally块总是会被执行，不管是否发生异常。

### 关于JDBC URL

JDBC URL 用于标识一个被注册的驱动程序，驱动程序管理器通过这个 URL 选择正确的驱动程序，从而建立到数据库的连接。JDBC URL的标准由三部分组成，各部分间用冒号分隔，如：**jdbc:<子协议>:<子名称>**

协议：JDBC URL中的协议总是jdbc

子协议：子协议用于标识一个数据库驱动程序

子名称：一种标识数据库的方法。子名称可以依不同的子协议而变化，用子名称的目的是为了定位数据库提供足够的信息

**几种常用的JDBC URL：**

对于 Oracle 数据库连接，采用如下形式：jdbc:oracle:thin:@localhost:1521:sid  
 对于 SQLServer 数据库连接，采用如下形式：jdbc:microsoft:sqlserver//localhost:1433; DatabaseName=sid  
 对于 MYSQL 数据库连接，采用如下形式：jdbc:mysql://localhost:3306/sid

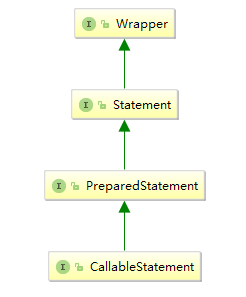
## Statement接口

通过调用 Connection 对象 的 createStatement 方法可以创建一个Statement对象，Statement 对象用于将 SQL 语句发送到数据库中。实际上有三种 Statement 对象，它们都作为在给定连接上执行 SQL语句的包装器： Statement、PreparedStatement和CallableStatement。

Statement 对象用于执行不带参数的简单 SQL 语句，它提供了执行语句和获取结果的基本方法；

PreparedStatement 对象用于执行带或不带 IN参数的预编译 SQL 语句；

Statement和PreparedStatement对象用于发送特定类型的SQL语句，而CallableStatement对象用于执行数据库中已存在存储过程的调用。



### **创建SQL执行对象**

Statement、PreparedStatement和CallableStatement对象，通过调用Connection相关方法来创建，如下列代码段中所示：

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(sql);  
  
PreparedStatement preparedStatement = connection.prepareStatement(sql);

ResultSet resultSet = preparedStatement.executeQuery();

CallableStatement callableStatement = connection.prepareCall(sql);

callableStatement.executeQuery();

### 使用 Statement 对象执行语句

Statement 接口提供了三种执行 SQL 语句的方法：executeQuery、executeUpdate 和execute。具体使用哪一个方法由 SQL 语句所产生的结果而决定：

* executeQuery 方法用于产生单个结果集的语句，例如 SELECT 语句。
* excuteUpdate()方法用来执行DML（插入、更新和删除）与DDL（创建数据库对象）语句，这个方法返回一个整数值，表示数据库中有多少行受到了影响。对于CREATE TABLE 或 DROP TABLE 等不操作行的语句，executeUpdate 的返回值总为零。
* 方法 execute用于执行返回多个结果集、多个更新计数或二者组合的语句。

有些 DBMS将已存储过程中的每条语句视为独立的语句，而另外一些则将整个过程视为一个复合语句。在启用自动提交时，这种差别就变得非常重要，因为它影响什么时候调用commit 方法。在前一种情况中，每条语句单独提交，在后一种情况中，所有语句同时提交。

### 关闭 Statement 对象

Statement 对象将由 [Java](http://lib.csdn.net/base/java" \o "Java 知识库" \t "http://blog.csdn.net/yejin191258966/article/details/_blank) 垃圾收集程序自动关闭。而作为一种好的编程风格，应在不需要 Statement对象时显式地关闭它们。这将立即释放 DBMS 资源，有助于避免潜在的内存问题。

### PreparedStatement的优点

PreparedStatement的预编译可以使你可以通过设置不同的参数来查询不同的目标，在数据库端，只会保存一段预编译语句，但是如果你使用Statement来发送语句，每发送一条，数据库中就会存一条，这可能会造成占用大量内存。

PreparedStatement可以写参数化查询，比Statement能获得更好的性能。  
 对于PreparedStatement来说，数据库可以使用已经编译过及定义好的执行计划，这种预处理语句查询比普通的查询运行速度更快。  
 PreparedStatement可以阻止常见的SQL注入式攻击。  
 PreparedStatement可以写动态查询语句  
 PreparedStatement与java.sql.Connection对象是关联的，一旦你关闭了connection，PreparedStatement也没法使用了。  
 PreparedStatement查询默认返回FORWARD\_ONLY的ResultSet，你只能往一个方向移动结果集的游标。当然你还可以设定为其他类型的值如：”CONCUR\_READ\_ONLY”。  
 不支持预编译SQL查询的JDBC驱动，在调用connection.prepareStatement(sql)的时候，它不会把SQL查询语句发送给数据库做预处理，而是等到执行查询动作的时候（调用executeQuery()方法时）才把查询语句发送个数据库，这种情况和使用Statement是一样的。  
 占位符的索引位置从1开始而不是0，如果填入0会导致\*java.sql.SQLException invalid column index\*异常。所以如果PreparedStatement有两个占位符，那么第一个参数的索引时1，第二个参数的索引是2.

以上就是为什么要使用PreparedStatement的全部理由，不过你仍然可以使用Statement对象用来做做测试。但是在生产环境下你一定要考虑使用 PreparedStatement 。

### Statement与PreparedStatement的区别

1.语法不同

Statement只支持静态编译，SQL语句是写死的。

PreparedStatement支持预编译，用？号来占位。

2.效率不同

Statement每次都要发送一条SQL语句，不支持缓存，执行效率低。

PreparedStatement支持预编译，缓存在数据库，只需发送参数，执行效率快。

3.安全性不同

Statement容易被注入。注入：狡猾的分子可以编写特殊的SQL语句来入侵数据库。

例如：要查询某个用户的信息

一般情况：SELECT \* FROM user\_list where username=xxx and password=xxx;(这里的xxx本应为用户填写自己的用户名和密码)

注入情况：SELECT \* FROM user\_list where username='abc' or 1=1 -- password=xxx;

这样1=1恒等，而且在password前加上了“--”号，后面的内容成为了注释不被执行。也就是说，这样就能不用密码地查询所有的用户信息。PreparedStatement，因为规定了SQL语句中的参数，所以可以防止注入。

结论：建议使用PreparedStatement，因为它更快更安全。

执行语句的所有方法都将关闭所调用的 Statement 对象的当前打开结果集（如果存在）。这意味着在重新执行 Statement对象之前，需要完成对当前 ResultSet 对象的处理。  
 应注意，继承了 Statement 接口中所有方法的 PreparedStatement 接口都有自己的executeQuery、executeUpdate 和 execute 方法。Statement 对象本身不包含 SQL语句，因而必须给 Statement.execute 方法提供 SQL 语句作为参数。PreparedStatement 对象并不将SQL 语句作为参数提供给这些方法，因为它们已经包含预编译 SQL 语句。CallableStatement 对象继承这些方法的PreparedStatement 形式。对于这些方法的 PreparedStatement 或 CallableStatement版本，使用查询参数将抛出 SQLException。  
  
 当连接处于自动提交模式时，其中所执行的语句在完成时将自动提交或还原。语句在已执行且所有结果返回时，即认为已完成。对于返回一个结果集的executeQuery 方法，在检索完 ResultSet 对象的所有行时该语句完成。对于方法executeUpdate，当它执行时语句即完成。但在少数调用方法 execute的情况中，在检索所有结果集或它生成的更新计数之后语句才完成。

## ResultSet接口

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。

ResultSet 对象以逻辑表格的形式封装了执行数据库操作的结果集，ResultSet 接口由数据库厂商实现。ResultSet 对象中维护了一个指向当前数据行的游标，初始的时候，游标在第一行之前，可以通过 ResultSet 对象的 next() 方法移动到下一行。

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。返回的是对应的XXX类型的值。如果对应那列是空值，XXX是对象的话返回XXX型的空值，如果XXX是数字类型，如Float等则返回0,boolean返回false.使用getString()可以返回所有的列的值，不过返回的都是字符串类型的。

XXX可以代表的类型有: 基本的数据类型如整型(int)，布尔型(Boolean)，浮点型(Float,Double)等，比特型(byte)，还包括一些特殊的类型，如：日期类型（java.sql.Date），时间类型(java.sql.Time)，时间戳类型(java.sql.Timestamp)，大数型 (BigDecimal和BigInteger）等等。

还可以使用getArray(intcolindex/String columnname)，通过这个方法获得当前行中，colindex所在列的元素组成的对象的数组。使用 getAsciiStream(intcolindex/String colname)可以获得该列对应的当前行的ascii流。也就是说所有的getXXX方法都是对当前行进行操作。

### 四种类型的ResultSet

结果集从其使用的特点上可以分为四类，这四类结果集所具备的特点都是和Statement语句的创建有关，因为结果集是通过Statement语句执行后产生的，所以可以说，结果集具备何种特点，完全决定于Statement。

另外，下面的代码中用到的变量conn代表的就是Connection对应的对象，SqlStr代表的是响应的SQL语句.

#### 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，逐个的读去数据.

#### 可滚动的ResultSet

Statement st =conn.createStatement(**resultSetType**, **resultSetConcurrency**)  
 ResultSet rs = st.executeQuery(sqlStr)

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。

**resultSetType参数：**设置ResultSet对象游标的滚动类型，取值如下:

* ResultSet.TYPE\_FORWARD\_ONLY：只能向前滚动(这是默认值)
* ResultSet.TYPE\_SCROLL\_INSENSITIVE：结果集的游标可以上下移动，当数据库变化时，当前结果集不变
* Result.TYPE\_SCROLL\_SENSITIVE：返回可滚动的结果集，当数据库变化时，当前结果集同步改变

**resultSetConcurency参数：**设置ResultSet对象能否修改的，取值如下：

* ResultSet.CONCUR\_READ\_ONLY：不能用结果集更新数据库中的表
* ResultSet.CONCUR\_UPDATABLE：能用结果集更新数据库中的表

#### 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、返回的列需包含主键

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。更新时，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()方法完成插入操作：**

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()：**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()：**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.

#### 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。JDBC2.0和1.0提供的都是提交后ResultSet 就会被关闭。不过在JDBC3.0中，我们可以设置ResultSet是否关闭。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，我所说的Statement的第三种创建方式，如下：

Statementst=createStatement(resultsetscrollable , resultsetupdateable , resultsetSetHoldability)  
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参数：**表示在结果集提交后结果集是否打开，取值有两个：

* ResultSet.HOLD\_CURSORS\_OVER\_COMMIT：表示修改提交时ResultSet不关闭
* ResultSet.CLOSE\_CURSORS\_AT\_COMMIT：表示修改提交时ResultSet关闭

注意：这种功能只是在JDBC3.0的驱动下才能成立。

## SQLException类

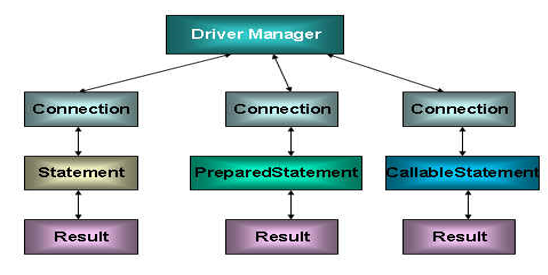
异常处理允许我们以受控的方式处理异常情况，而不是直接退出程序，例如程序定义的错误。发生异常时可以抛出异常。术语“异常”表示当前的程序执行停止，并且被重定向到最近的适用的catch子句。如果没有适用的catch子句存在，则程序的执行结束。JDBC异常处理与Java异常处理非常相似，但对于JDBC，要处理的最常见异常是java.sql.SQLException。

**SQLException的相关方法**

驱动程序和数据库中都会发生SQLException。 发生这种异常时，SQLException类型的对象将被传递给catch子句。传递的SQLException对象具有以下可用于检索有关异常信息的方法：

| 方法 | 描述 |
| --- | --- |
| getErrorCode( ) | 获取与异常关联的错误代码。 |
| getMessage( ) | 获取驱动程序处理的错误的JDBC驱动程序的错误消息，或获取数据库错误的Oracle错误代码和消息。 |
| getSQLState( ) | 获取XOPEN SQLstate字符串。 对于JDBC驱动程序错误，不会从此方法返回有用的信息。 对于数据库错误，返回五位数的XOPEN SQLstate代码。 此方法可以返回null。 |
| getNextException( ) | 获取异常链中的下一个Exception对象。 |
| printStackTrace( ) | 打印当前异常或可抛出的异常，并将其追溯到标准错误流。 |
| printStackTrace(PrintStream s) | 将此throwable及其回溯打印到指定的打印流。 |
| printStackTrace(PrintWriter w) | 打印这个throwable，它是回溯到指定的打印器(PrintWriter)。 |

## 小结（扩展）



java.sql.DriverManager用来装载驱动程序，获取数据库连接  
java.sql.Connection完成对某一指定数据库的连接  
java.sql.Statement在一个给定的连接中作为SQL执行声明的容器，他包含了两个重要的子类型：  
 Java.sql.PreparedSatement 用于执行预编译的sql声明  
 Java.sql.CallableStatement用于执行数据库中存储过程的调用  
java.sql.ResultSet存储查询结果的对象

**简单的JDBC API**

Connection :    与数据库连接的接口

　　　　　　|- Statement createStatement()   --创建Statement对象，用于发送sql语句

　　　　　　|- PreparedStatement prepareStatement(String sql)  -创建PreparedStatement对象，用于发送预编译的sql语句

　　　　　　|-CallableStatement prepareCall(String sql)  --创建CallableStatement对象，用于调用存储过程。

　　　　　　|-Statement： 用于执行静态sql语句

　　　　　　　　　　|-int executeUpdate(String sql)  --执行更新操作（DDL+DML）

　　　　　　　　　　|-ResultSet executeQuery(String sql)  --执行查询操作(DQL)

　　　　　　|- PreparedStatement： 用于执行预编译的sql语句

　　　　　　　　　　|- int executeUpdate() -- 执行更新操作

　　　　　　　　　　|- ResultSet executeQuery()    -- 执行查询操作

　　　　　　|- CallableStatement： 用于执行存储过程的sql

　　　　　　　　　　|- ResultSet executeQuery()  --调用存储过程

　　　　　　　　　　|- ResultSet： 结果集。用于封装数据库的查询后的数据

　　　　　　　　　　　　　　|- boolean next() --将记录光标移动到下一行

　　　　　　　　　　　　　　|- Object getObject(int columnIndex) -- 得到字段上的值

### １、SQL注入攻击

SQL 注入是利用某些系统没有对用户输入的数据进行充分的检查，而在用户输入数据中注入非法的 SQL 语句段或命令，从而利用系统的 SQL 引擎完成恶意行为的做法

对于 Java 而言，要防范 SQL 注入，只要用 PreparedStatement 取代 Statement 就可以了。

### 2、PreparedStatement

可以通过调用 Connection 对象的 preparedStatement() 方法获取 PreparedStatement 对象。PreparedStatement 接口是 Statement 的子接口，它表示一条预编译过的 SQL 语句。PreparedStatement 对象所代表的 SQL 语句中的参数用问号(?)来表示，调用 PreparedStatement 对象的 setXXX() 方法来设置这些参数. setXXX() 方法有两个参数，第一个参数是要设置的 SQL 语句中的参数的索引(从 1 开始)，第二个是设置的 SQL 语句中的参数的值

**PreparedStatement 与Statement：**

PreparedStatement 能最大可能提高性能。DBServer会对预编译语句提供性能优化。因为预编译语句有可能被重复调用，所以语句在被DBServer的编译器编译后的执行代码被缓存下来，那么下次调用时只要是相同的预编译语句就不需要编译，只要将参数直接传入编译过的语句执行代码中就会得到执行。

在statement语句中,即使是相同操作但因为数据内容不一样,所以整个语句本身不能匹配,没有缓存语句的意义.事实是没有数据库会对普通语句编译后的执行代码缓存.这样每执行一次都要对传入的语句编译一次。  
(语法检查，语义检查，翻译成二进制命令，缓存)

PreparedStatement 可以防止 SQL 注入 。

### 3、使用JDBC驱动程序处理元数据

Java 通过JDBC获得连接以后，得到一个Connection 对象，可以从这个对象获得有关数据库管理系统的各种信息，包括数据库中的各个表，表中的各个列，数据类型，触发器，存储过程等各方面的信息。根据这些信息，JDBC可以访问一个事先并不了解的数据库。

获取这些信息的方法都是在DatabaseMetaData类的对象上实现的，而DataBaseMetaData对象是在Connection对象上获得的。

### 4、DateBaseMetaDate

DatabaseMetaData 类中提供了许多方法用于获得数据源的各种信息，通过这些方法可以非常详细的了解数据库的信息：

* 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。
* isReadOnly()：返回一个boolean值，指示数据库是否只允许读操作。
* 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。

### 5、ResultMetaDate

可用于获取关于 ResultSet 对象中列的类型和属性信息的对象：

* getColumnName(int column)：获取指定列的名称
* getColumnCount()：返回当前 ResultSet 对象中的列数。
* getColumnTypeName(int column)：检索指定列的数据库特定的类型名称。
* getColumnDisplaySize(int column)：指示指定列的最大标准宽度，以字符为单位。
* isNullable(int column)：指示指定列中的值是否可以为 null。
* isAutoIncrement(int column)：指示是否自动为指定列进行编号，这样这些列仍然是只读的。

用例：获取数据库自动生成的主键\*

Connection conn = JdbcUtil.getConnection();   
String sql = **"insert into user(name,password,email,birthday) values('abc','123','abc@sina.com','1978-08-08')"**;   
PreparedStatement st = conn.prepareStatement(sql,Statement.***RETURN\_GENERATED\_KEYS*** );   
st.executeUpdate();   
ResultSet rs = st.getGeneratedKeys(); *//得到插入行的主键***if**(rs.next()){   
 System.***out***.println(rs.getObject(1));   
}

### 6、数据库事务

在数据库中,所谓事务是指一组逻辑操作单元,使数据从一种状态变换到另一种状态。

为确保数据库中数据的一致性,数据的操纵应当是离散的成组的逻辑单元:当它全部完成时,数据的一致性可以保持,而当这个单元中的一部分操作失败,整个事务应全部视为错误,所有从起始点以后的操作应全部回退到开始状态。

事务的操作:先定义开始一个事务,然后对数据作修改操作,这时如果提交(COMMIT),这些修改就永久地保存下来,如果回退(ROLLBACK),数据库管理系统将放弃所作的所有修改而回到开始事务时的状态。

事务的ACID(acid)属性

* 原子性（Atomicity）：原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
* 一致性（Consistency）：事务必须使数据库从一个一致性状态变换到另外一个一致性状态。
* 隔离性（Isolation）：事务的隔离性是指一个事务的执行不能被其他事务干扰，即一个事务内部的操作及使用的数据对并发的其他事务是隔离的，并发执行的各个事务之间不能互相干扰。
* 持久性（Durability）：持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来的其他操作和数据库故障不应该对其有任何影响。

### 7、JDBC事务处理

事务：指构成单个逻辑工作单元的操作集合

事务处理：保证所有事务都作为一个工作单元来执行，即使出现了故障，都不能改变这种执行方式。当在一个事务中执行多个操作时，要么所有的事务都被提交(commit)，要么整个事务回滚(rollback)到最初状态

当一个连接对象被创建时，默认情况下是自动提交事务：每次执行一个 SQL 语句时，如果执行成功，就会向数据库自动提交，而不能回滚。为了让多个 SQL 语句作为一个事务执行，调用 Connection 对象的 setAutoCommit(false); 以取消自动提交事务在所有的 SQL 语句都成功执行后，调用 commit(); 方法提交事务在出现异常时，调用 rollback(); 方法回滚事务若此时 Connection 没有被关闭, 则需要恢复其自动提交状态

### 8、数据库的隔离级别

对于同时运行的多个事务, 当这些事务访问数据库中相同的数据时, 如果没有采取必要的隔离机制, 就会导致各种并发问题:

脏读: 对于两个事物 T1, T2, T1 读取了已经被 T2 更新但还没有被提交的字段. 之后, 若 T2 回滚, T1读取的内容就是临时且无效的.

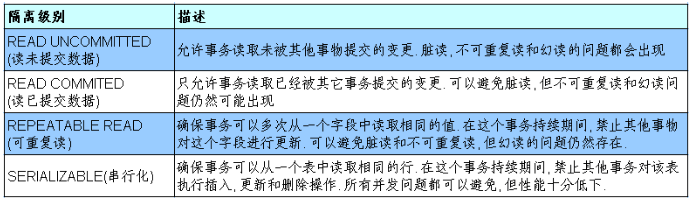
不可重复读: 对于两个事物 T1, T2, T1 读取了一个字段, 然后 T2\*\* 更新\*\*了该字段. 之后, T1再次读取同一个字段, 值就不同了.

幻读: 对于两个事物 T1, T2, T1 从一个表中读取了一个字段, 然后 T2 在该表中插入了一些新的行. 之后, 如果 T1 再次读取同一个表, 就会多出几行.

数据库事务的隔离性: 数据库系统必须具有隔离并发运行各个事务的能力, 使它们不会相互影响, 避免各种并发问题.

一个事务与其他事务隔离的程度称为隔离级别. 数据库规定了多种事务隔离级别, 不同隔离级别对应不同的干扰程度, 隔离级别越高, 数据一致性就越好, 但并发性越弱

**数据库的隔离级别**



Oracle 支持的 2 种事务隔离级别：READ COMMITED, SERIALIZABLE. Oracle 默认的事务隔离级别为: READ COMMITED

Mysql 支持 4 中事务隔离级别. Mysql 默认的事务隔离级别为: REPEATABLE READ