# Jdbc（1）

## Project(Jdbc01)

## JDBC入门

* 使用前准备
* 导jar包：驱动！
* 加载驱动类，Class.forname(“类名”);
* 给出url,username,password.其中url背下来;
* 使用DriverManager类来得到Connection对象.
* *\* jdbc四大配置参数  
  \* 1.driverClassName:**com.mysql.jdbc.Driver  
  \* 2.url:jdbc:mysql://localhost:3306/mydb3  
  \* 3.username:root  
  \* 4.password:1234*
* 原理

Class.forname(“com.mysql.jdbc.Driver”)，加载驱动类时，执行以下方法，自己注册：







此时不写Class.forname(“com.mysql.jdbc.Driver”)也可以连接成功！

* 代码（重要）

**public void** fun1() **throws** Exception{  
 */\*\*  
 \* 查询  
 \* 一、连接数据库，得到connection  
 \* 二、得到statememt,执行select语句  
 \* 三、处理并输出结果  
 \* \*/  
 /\*  
 \* 一、连接数据库，得到connection  
 \* \*/* String driverClassName = **"com.mysql.jdbc.Driver"**;  
 String url = **"jdbc:mysql://localhost:3306/mydb3"**;  
 String username = **"root"**;  
 String password = **"1234"**;  
 Class.*forName*(driverClassName);  
 *//得到连接对象* Connection connection = DriverManager.*getConnection*(url, username, password);  
 */\*  
 \* 二、得到statememt,执行select语句  
 \* 1.得到statememt对象：Connection的createStatememt()方法  
 \* \*/* Statement statement = connection.createStatement();  
 ResultSet rs = statement.executeQuery(**"SELECT** *\** **FROM stu"**);  
 */\*\*  
 \* 三、解析ResultSet  
 \* 1.把行光标移动到第一行，调用next()方法完成*

*\* 2.getObject(int index)和getString()方法最为通用  
 \* \*/* **while** (rs.next()){*//把光标向下移动一行，并判断下一行是否存在！* String number = rs.getString(**"number"**);  
 String name = rs.getString(**"name"**);  
 **int** age = rs.getInt(**"age"**);  
 String gender = rs.getString(**"gender"**);  
 System.***out***.println(**"[ "**+number+**" , "**+name+**" , "**+age+**" , "**+gender+**" ]"**);  
 }  
 */\*\*  
 \* 四、关闭资源（倒关）  
 \* \*/* rs.close();  
 statement.close();  
 connection.close();*//必须要关闭，在web项目中不关闭容易出问题*}

* **代码规范化（重要）**

*/\*  
 \* 代码规范化  
 \* \*/* @Test  
 **public void** fun2() **throws** Exception{  
 *//定义引用* Connection connection = **null**;  
 Statement statement = **null**;  
 ResultSet rs = **null**;  
 **try** {  
 */\*  
 \* 一、得到连接  
 \* \*/* String driverClassName = **"com.mysql.jdbc.Driver"**;  
 String url = **"jdbc:mysql://localhost:3306/mydb3"**;  
 String username = **"root"**;  
 String password = **"1234"**;  
  
 Class.*forName*(driverClassName);  
 connection = DriverManager.*getConnection*(url, username, password);*//实例化  
 /\*  
 \* 二、实例化statement  
 \* \*/* statement = connection.createStatement();  
 String sql = **"SELECT** *\** **FROM stu"**;  
 rs = statement.executeQuery(sql);  
 */\*  
 \* 三、输出  
 \* \*/* **while** (rs.next()){*//把光标向下移动一行，并判断下一行是否存在！  
// String number = rs.getString("number");* String name = rs.getString(**"name"**);  
 **int** age = rs.getInt(**"age"**);  
 String gender = rs.getString(**"gender"**);  
 System.***out***.println(**"[ "**+rs.getObject(1)+**" , "**+name+**" , "**+age+**" , "**+gender+**" ]"**);  
 }  
 } **catch** (Exception e) {  
 **throw new** RuntimeException(e);  
 } **finally** {  
 */\*  
 \* finally作为异常处理的一部分，它只能用在try/catch语句中，并且附带一个语句块，表示这段语句最终一定会被执行（不管有没有抛出异常），经常被用在需要释放资源的情况下。  
 \* \*/  
 //关闭* **if**(rs != **null**) rs.close();  
 **if**(statement != **null**) statement.close();;  
 **if**(connection != **null**) connection.close();  
 }  
 }

## JDBC对象介绍

### DriverManager

### Connection

获取Statement

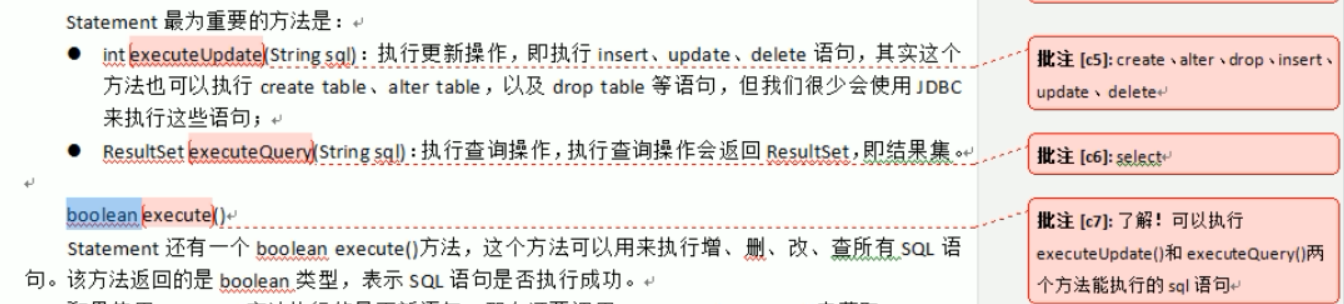
* 无参数（最常用）

Statement statement = Connection.createStatement();

* 有参数（与ResultSet相关）

Statement statement = Connection.createStatement(int,int);

### Statememt



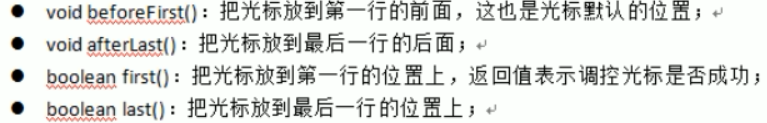
* Boolean execute()方法
* 可以执行任意的sql；
* 返回结果为boolean类型，表示sql语句是否有结果集；
* 若执行更新语句，需要调用int getUpdateCount()来获取insert,update,delete语句所影响的行数；
* 若执行查询语句，需要调用ResultSet getesult()来获取select语句的查询结果。
* 常用的是前两种方法；

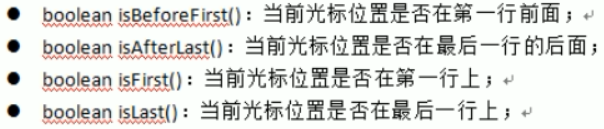
### ResultSet之滚动结果集

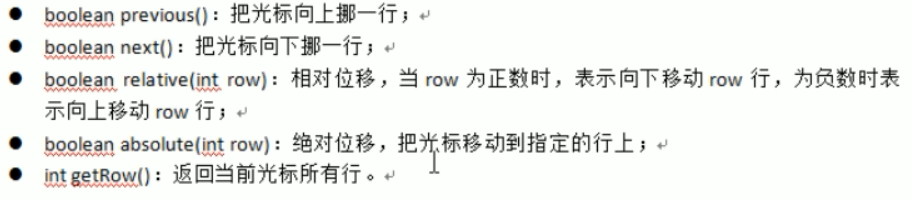
* 最常用：

Next()方法，将光标移动到下一行，并判断下一行是否存在。

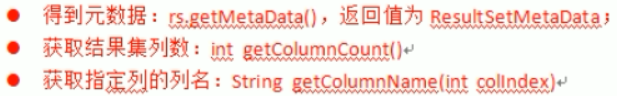
* 移动光标的4种方法：



* 判断光标位置：  
  
* 其他移动光标的方法

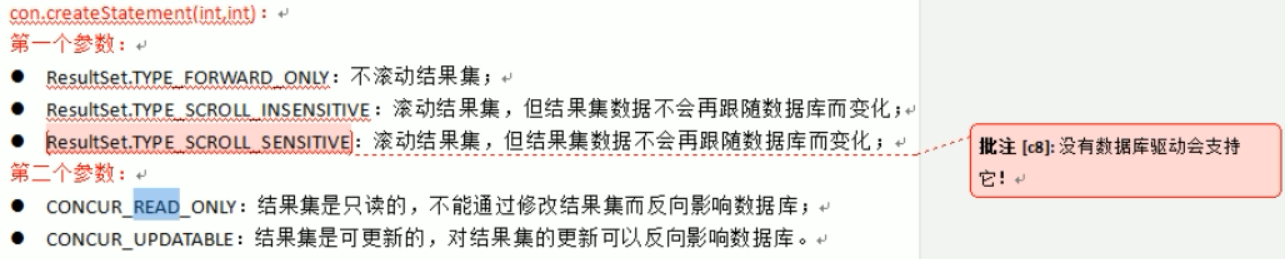


* 获取列数及列名



### 结果集的特性（面试问题）

当使用Connection的createStatement时，已经确定了Statement生成的结果集的特性。



## PreparedStatement

* Statement接口的子接口
* 强大之处；
* 防sql攻击；
* 提高代码的可读性、可维护性
* 提高效率！
* 用法

*/\*  
\* 一、得到 PreparedStatement  
\* 1.给出sql模板，所有参数使用?来代替  
\* 2.调用connection()方法，得到PreparedStatement对象  
\* \*/*String sql = **"SELECT** *\** **FROM t\_user WHERE username=? AND password=?"**;  
 PreparedStatement preparedStatement = connection.prepareStatement(sql);  
 */\*  
 \* 二、为参数赋值  
 \* \*/* preparedStatement.setString(1,username);*//给第一个参数赋值，值为username* preparedStatement.setString(2,password);*//给第二个参数赋值，值为password* ResultSet resultSet = preparedStatement.executeQuery();*//调用查询方法，向数据库发送查询语句（方法没有参数）* **return** resultSet.next();*//返回结果*

* 预处理的原理
* 数据库必须支持预处理；
* 每一个preparedStatement都与一个sql模板绑定在一起，先把sql模板给数据库，数据库先进行校验，再进行编译。执行时只是把参数传递过去而已。
* 若第二次执行时，就不用再次校验语法，也不用再次编译，直接执行！

## JdbcUtils工具类

**import** org.omg.PortableServer.THREAD\_POLICY\_ID;  
  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.security.PrivateKey;  
**import** java.sql.Connection;  
**import** java.sql.DriverManager;  
**import** java.time.Period;  
**import** java.util.Properties;  
  
*/\*  
\* v1.0  
\* \*/***public class** JdbcUtils {  
 **private static** Properties *properties* = **null**;  
 *//只在JdbcUtils类被加载时执行一次！* **static** {  
 */\*  
 \* 得到properties  
 \* \*/* **try**{  
 *//加载配置文件* InputStream resourceAsStream = JdbcUtils.**class**.getClassLoader().getResourceAsStream(**"dbconfig.properties"**);  
 *properties* = **new** Properties();  
 *properties*.load(resourceAsStream);  
 }**catch** (IOException e){  
 **throw new** RuntimeException(e);  
 }  
 *//加载驱动类* **try** {  
 Class.*forName*(*properties*.getProperty(**"driverClassName"**));  
 } **catch** (ClassNotFoundException e) {  
 **throw new** RuntimeException(e);  
 }  
  
 }  
 **public static** Connection getConnetion() **throws** Exception {  
  
 */\*\*  
 \* 1.加载配置文件  
 \* 2.加载驱动类  
 \* 3.调用DriverManager.getConnection()  
 \* \*/  
 //得到connection* **return** DriverManager.*getConnection*(*properties*.getProperty(**"url"**),  
 *properties*.getProperty(**"username"**),  
 *properties*.getProperty(**"password"**));  
 }  
}

## UserDao

### 修改项目

* 创建UserDao接口，创建类UserDaoImpl实现UserDao接口；
* 创建DaoFactory,提供getUserDao()；
* 在UserService中对UserDao进行实例化:

Private UserDao userdao = DaoFactory.getUserDao();

### Dao模式

## 时间类型

### 数据库类型与java中类型的对应关系；

* DATE 🡪 java.sql.Date
* TIME 🡪 java.sql.Time
* TIMESTAMP 🡪 java.sql.Timestamp
* 邻域对象（domain）中的所有属性不能出现java.sql包下的东西！即不能使用java.sql.Date!
* resultSet#getDate() 和

preparedStatement#setDate(int,Date)中的Date都是java.sql.Date类型。

### 时间类型的转化

* java.utils.Date 🡪 java.sql.Date,Time,Timestamp
* 把java.utils.Date转化为毫秒值；
* 使用毫秒值创建java.sql.Date,Time,Timestamp；
* 例如：

java.utils.Date date = new java.utils.Date();

long l = date.getTime();

java.sql.Date sqldate = new java.sql.Date(l);

* java.sql.Date,Time,Timestamp 🡪 java.utils.Date
* 因为java.utils.Date是其他三者的父类，所以不需要处理：

java.utils.Date date = new java.sql.Date

## 大型（字符、字节）数据

* Mysql中的大型数据



* 更改允许的最大文件大小
* 先显示文件资源管理器的隐藏文件，再C盘中发现名为ProgramDate的文件夹；
* 找到mysql的配置文件my.ini；
* 更改max\_allowed\_packet = 100M；
* 向数据库中写入

*/\*  
\* 需要Blob类的对象  
\* 1.我们有的是文件，目标是Blob;  
\* 2.先把文件变为byte[];  
\* 3.再使用byte[]创建blob;  
\* \*/  
//把文件转化为byte[]***byte**[] bytes = IOUtils.*toByteArray*(

**new** FileInputStream(**"C:\\Users\\JOE\\Music\\Super Brothers.mp3"**)

);  
*//使用byte[]创建blob*Blob blob = **new** SerialBlob(bytes);  
*//设置参数*preparedStatement.setBlob(3,blob);  
*//执行语句*preparedStatement.executeUpdate();

* 从数据库中读取：

*/\*  
\* 把Blob变为硬盘上的文件  
\* \*/  
/\*  
\* 1.通过Blob得到输入流对象  
\* 2.自己创建输出流对象  
\* 3.把输入流的数据写到输出流中  
\* \*/*InputStream in = data.getBinaryStream();  
FileOutputStream out = **new** FileOutputStream(**"C:\\Users\\JOE\\Desktop\\110.mp3"**);  
IOUtils.*copy*(in,out);

## PreparedStatememt(SQL模版)的批处理

### 原理

逐条向服务器发送SQL语句，执行效率太慢，批处理即一次向服务器发送多条SQL语句，进行一次性处理，只针对增、删、改语句，不包括查询。

### 添加批处理支持

* 在Mysql数据库的url之后添加参数，开启批处理

**url**=**jdbc:mysql://localhost:3306/mydb3?rewriteBatchedStatements=true**

之后，时间大大缩减。