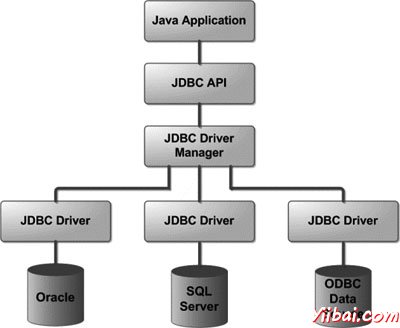
JDBC

# JDBC简介

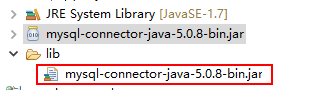
JDBC（Java Data Base Connectivity,java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用[Java语言](http://baike.baidu.com/item/Java%E8%AF%AD%E8%A8%80" \t "http://baike.baidu.com/_blank)编写的类和接口组成。JDBC提供了一种基准，据此可以构建更高级的工具和接口，使数据库开发人员能够编写数据库应用程序

* Java DataBase Connectivity（java数据库连接）
* 组成包：java.sql.\*;javax.sql.\*;这两个包都包含在了JDK中。
* 还需要数据库的驱动，这些驱动就相当于对JDBC规范的实现



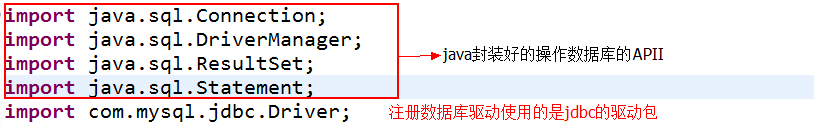
# 开发步骤

1. 创建一个java项目
2. 导入mysql的数据库驱动jar包



1. **注册驱动**
2. **获取与数据库的链接**
3. **得到代表发送和执行SQL语句的对象 Statement**
4. **执行语句**
5. **如果执行的是查询语句，就会有结果集，处理**
6. **释放占用的资源**

|  |
| --- |
| // 注册数据库驱动  DriverManager.*deregisterDriver*(**new** Driver());  // 获取数据库连接  Connection conn = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/ssm", "root", "hang");  // 获取执行sql语句的statem0ent  Statement st = conn.createStatement();  String sql = "select \* from user ";  // 执行sql语句增删改查  //st.executeUpdate("insert into user values(12,'杨过','123456')");  //st.executeUpdate("update user set namer='杨康' where id='12'");  //st.executeUpdate("delete from user where id='12'");  ResultSet set = st.executeQuery(sql);  **while** (set.next()) {  // 获取指定位置的数据，角标从1 开始  String com1 = set.getString(1);  String com2 = set.getString(2);  String com3 = set.getString(3);  String com4 = set.getString(4);  System.***out***.println(com2 + ":" + com3 + ":" + com4 + ":" + com1);  }  // 释放数据库资源  set.close();  st.close();  conn.close(); |



# 主要接口或类

## **DriverManger**

**作用**

1. 注册驱动b、
2. 获取与数据库的链接

**改进注册驱动：**

DriverManager.registerDriver(new com.mysql.jdbc.Driver());

缺点：严重依赖具体的驱动类；会导致驱动被注册2次。

替代方案：Class.forName("com.mysql.jdbc.Driver");

**获取与数据库的链接**

DriverManager.getConnection("jdbc:mysql://localhost:3306/ssm", "root", "hang"); //ssm为数据库名

|  |
| --- |
| Class.forName("com.mysql.jdbc.Driver");  // 获取数据库连接  Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/ssm", "root", "hang"); |

## Connection

所有的与数据库的交互都是基于链接的基础之上的，想要对数据库进行操作，首先要获取此对象，从连接对象中获取执行数据库的statement对象

Statement stmt = conn.createStatement()：创建向数据库发送sql的statement对象

## Statement

**作用：**

向数据库发送并执行具体的SQL语句

**常用方法：**

常用方法：

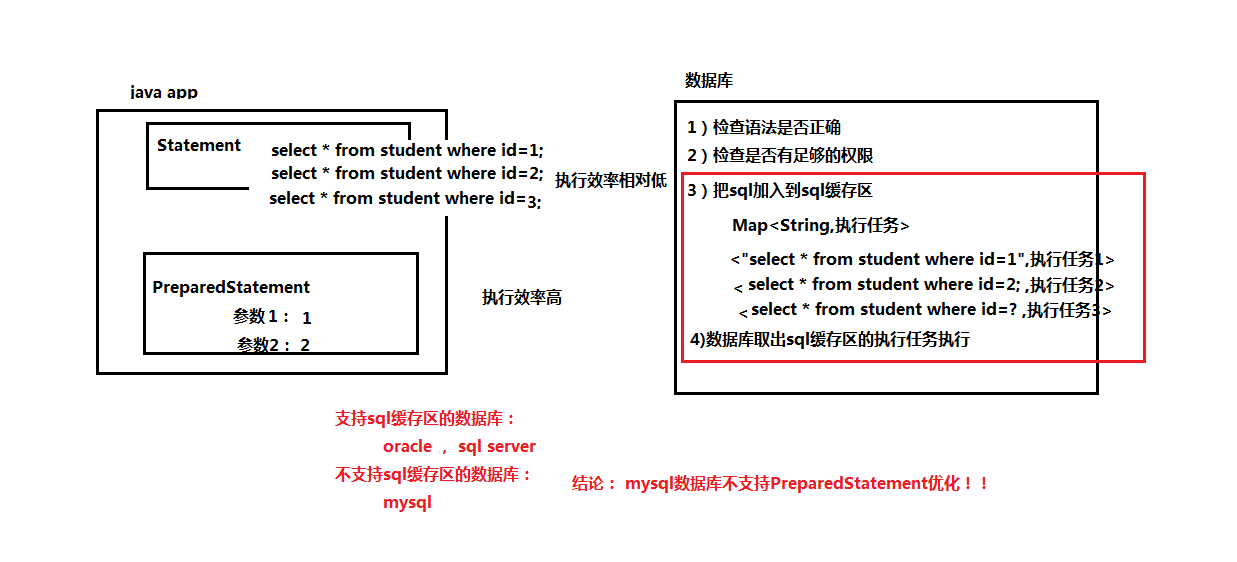
* ResultSet executeQuery(String sql):只适合查询，返回的是查询的结果集
* int executeUpdate(String sql):适合DML，或者没有返回结果集的DDL,返回的是影响的记录行数
* boolean execute(String sql):执行任何的SQL语句，返回的不是成功与否。有结果集的返回true，没有返回false。

|  |
| --- |
| //有结果集的返回true，没有返回false  **boolean** b = st.execute(sql);  //返回的是影响的记录行数  **int** update = st.executeUpdate(sql);  //返回结果集  ResultSet set = st.executeQuery(sql); |

## PreparedStatement

**PreparedStatement实现Statement**

* PreparedStatement需要预编译以及需要参数
* 由于PreparedStatement有缓存区，所以效率更高
* 由于PreparedStatement有缓存区，所以更安全，防止了注入（1=1）
* 语句中的参数可以使用占位符(?)



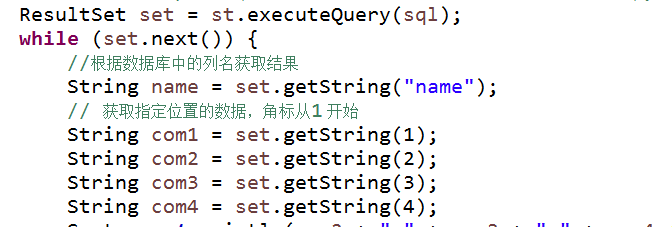
|  |
| --- |
| // 2：准备sql预编译语句  // ?占用一个参数位  String sql = "INSERT INTO user (NAME,sex,age) VALUES (?,?,?);";  // 3：执行sql预编译语句（检查语法）  stsm=conn.prepareStatement(sql);  // 4：设置传递的参数  stsm.setString(1,"张三");  stsm.setString(2,"男");  stsm.setInt(3,20);  // 5：发送参数，执行sql  // 注意：这里的方法后面没有参数  **int** result = stsm.executeUpdate(); |

**能用PreparedStatement就不要使用Statement**

## ResultSet

**作用：**

代表者查询语句的查询结果集



## 释放占用的资源

**按照打开的顺序，以相反的方向进行释放。**

**Connection要尽量晚的创建和尽量早的释放。**

为了保障资源能被释放，应该放到finally代码块中

|  |
| --- |
| finally{  //6.关闭资源 (有顺序)  if(rs!=null){  try {  rs.close();  rs = null; //垃圾回收器立即回收  } catch (SQLException e) {  e.printStackTrace();  }  }  if(st!=null){  try {  st.close();  st = null;  } catch (SQLException e) {  e.printStackTrace();  }  }  if(con!=null){  try {  con.close();  con = null;  } catch (SQLException e) {  e.printStackTrace();  }  }  } |

优化代码如下：最开始的例子中存在，数去库驱动两次注册，数据未封装，关闭资源不规范，异常未处理等问题 。

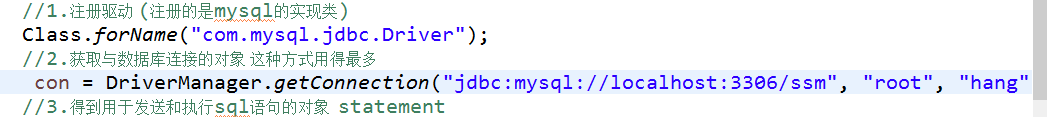
|  |
| --- |
| **public** **static** **void** main(String[] args) {  Connection con = **null**;  Statement st = **null**;  ResultSet rs = **null**;  **try** {  //1.注册驱动 (注册的是mysql的实现类)  Class.*forName*("com.mysql.jdbc.Driver");  //2.获取与数据库连接的对象 这种方式用得最多  con = DriverManager.*getConnection*("jdbc:mysql://localhost:3306/ssm", "root", "hang");  //3.得到用于发送和执行sql语句的对象 statement  st = con.createStatement();  //4.执行语句  rs = st.executeQuery("select \* from book ");//用于查询的  //5.如果是查询，就有结果集，就要处理结果集  List<Book> list = **new** ArrayList<Book>();  **while**(rs.next()){  Book book = **new** Book();  book.setBid(rs.getString("bid"));  book.setTitle(rs.getString("title"));  book.setAuthor(rs.getString("author"));  list.add(book);  }    //展示数据  **for**(Book book :list){  System.***out***.println(book);  }    } **catch** (Exception e) {  e.printStackTrace();  }**finally**{  //6.关闭资源 (有顺序)  **if**(rs!=**null**){  **try** {  rs.close();  rs = **null**; //垃圾回收器立即回收  } **catch** (SQLException e) {  e.printStackTrace();  }  }    **if**(st!=**null**){  **try** {  st.close();  st = **null**;  } **catch** (SQLException e) {  e.printStackTrace();  }  }    **if**(con!=**null**){  **try** {  con.close();  con = **null**;  } **catch** (SQLException e) {  e.printStackTrace();  }  }    }  } |

# 工具类的提取

问题一 ；

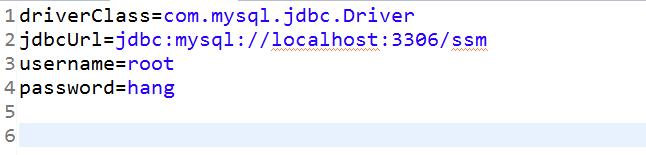
根据上面的例子咳咳发现，获取数据库连接，释放数据库资源的代码都一样，可以提取出来一个方法，当获取连接，释放资源是直接调用方法，减少代码的冗余度。

问题二 ：加载数据库驱动，获取连接时存在硬编码，当数据库改变，数据库类型改变。密码等信息改变时，代码不利于维护 。

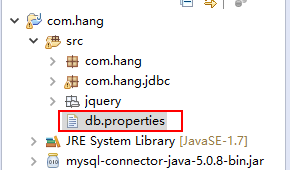


## 配置文件properties

配置数据库连接的信息，数据都是以key/value的形式保存



在src跟目录下创建db.properties 文件 。



获取配置文件中的value

|  |
| --- |
| //通过类加载器，获取db.propertis文件的输入流  InputStream in = GetProperties.**class**.getClassLoader().getResourceAsStream("db.properties");  Properties p = **new** Properties();  **try** {  p.load(in);//加载文件内容  //根据key的名称获取文件的value值  String driverClass = p.getProperty("driverClass");  String jdbcUrl = p.getProperty("jdbcUrl");  String username = p.getProperty("username");  String password = p.getProperty("password");  } **catch** (IOException e) {  // **TODO** Auto-generated catch block  e.printStackTrace();  } |

## 封装工具类，提取获取连接 ，关闭资源方法

|  |
| --- |
| public static String DRIVER ;  public static String URL ;  public static String USERNAME ;  public static String PASSWORD ;    static{  //读取配置文件的方式  InputStream is = JdbcUtil.class.getClassLoader().getResourceAsStream("db.properties");  Properties p = new Properties();  try {  p.load(is);  DRIVER = p.getProperty("driverClass");  URL = p.getProperty("jdbcUrl");  USERNAME = p.getProperty("username");  PASSWORD = p.getProperty("password");    Class.forName(DRIVER);//加载驱动  } catch (Exception e) {  e.printStackTrace();  }  }  /\*\*  \* 获取数据库连接  \* @return  \* @throws Exception  \*/  public static Connection getConnection() throws Exception{  return DriverManager.getConnection(URL, USERNAME, PASSWORD);  }    /\*\*  \* 关闭资源的方法  \* @param rs  \* @param st  \* @param con  \*/  public static void release(ResultSet rs ,Statement st,Connection con){  //6.关闭资源 (有顺序)  if(rs!=null){  try {  rs.close();  rs = null; //垃圾回收器立即回收  } catch (SQLException e) {  e.printStackTrace();  }  }    if(st!=null){  try {  st.close();  st = null;  } catch (SQLException e) {  e.printStackTrace();  }  }    if(con!=null){  try {  con.close();  con = null;  } catch (SQLException e) {  e.printStackTrace();  }  }    } |

## 工具类的使用

**获取连接时调用工具类的获取连接方法，释放资源时，调用释放资源的方法**

|  |
| --- |
| Connection con = null;  Statement st = null;  ResultSet rs = null;  try {  con = JdbcUtil.getConnection();  st = con.createStatement();  rs = st.executeQuery("");  while(rs.next()){  //封装数据  }  } catch (Exception e) {  e.printStackTrace();  }finally{  JdbcUtil.release(null, st, con);  } |

# 数据库连接池

## 什么是数据库连接池

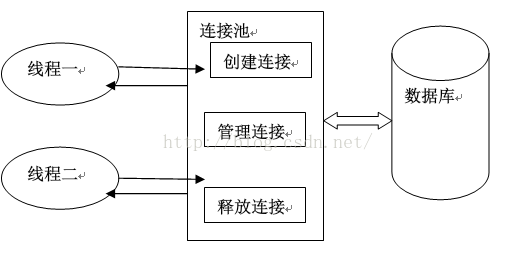
**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/wenwen091100304/article/details/_blank)**连接池（Connection pooling）是程序启动时建立足够的数据库连接，并将这些连接组成一个连接池，由程序动态地对池中的连接进行申请，使用，释放。

**优点**

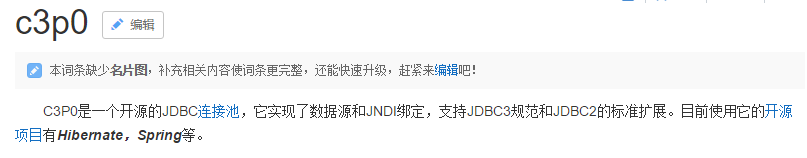
* 节省创建连接与释放连接 性能消耗
* 连接池中连接起到复用的作用 ，提高程序性能

## 数据库连接池的运行机制

（1）  程序初始化时创建连接池  
（2） 使用时向连接池申请可用连接  
（3） 使用完毕，将连接返还给连接池  
（4） 程序退出时，断开所有连接，并释放资源

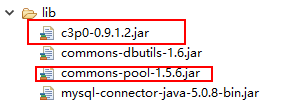


## C3p0连接池 dbcp连接池



## C3p0 的使用

### 添加两个jar包



### 数据库的源配置文件

db.properties 前面的可以任意，但是需要自己加载

|  |
| --- |
| driverClass=com.mysql.jdbc.Driver  jdbcUrl=jdbc:mysql://localhost:3306/ssm  username=root  password=hang |

以下两种配置方式，c3p0连接池会自动读取配置文件

c3p0.properties 文件名不能改， 必须放在src下 ，配置文件中的key名称不能变

|  |
| --- |
| c3p0.driverClass=com.mysql.jdbc.Driver  c3p0.jdbcUrl=jdbc:mysql://127.0.0.1:3306/ssm?characterEncoding=UTF-8  c3p0.user=root  c3p0.password=hang |

c3p0-config.xml 可以配置多个数据库连接并且可以指定名字去加载

ComboPooledDataSource dataSource = new ComboPooledDataSource("my");

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <c3p0-config>  <!-- 默认的配置，默认读取该配置 -->  <default-config>  <property name=*"driverClass"*>com.mysql.jdbc.Driver</property>  <property name=*"jdbcUrl"*>jdbc:mysql:///ssm</property>  <property name=*"user"*>root</property>  <property name=*"password"*>hang</property>  </default-config>    <!-- 指定名称的配置 -->  <named-config name=*"my"*>  <property name=*"driverClass"*>com.mysql.jdbc.Driver</property>  <property name=*"jdbcUrl"*>jdbc:mysql:///ssm</property>  <property name=*"user"*>root</property>  <property name=*"password"*>hang</property>  </named-config>  </c3p0-config> |

如果使用上面的两种颜色的配置文件配置，且放在src目录下 c3p0数据库连接池会自动读取配置文件，

|  |
| --- |
| // 测试c3p0  ComboPooledDataSource dataSource = **new** ComboPooledDataSource();  QueryRunner runner = **new** QueryRunner(dataSource); |

query(sql, **new** BeanListHandler<封装的实体类>(类名.**class**), 参数可以多个);

可以直接执行sql语句

c3p0有自动回收空闲连接功能 自动连接