JDBC的全称为:Java DataBase Connectivity(java数据库连接)

- 搭建开发环境
- 编写程序，在程序中加载数据库驱动
- 建立连接
- 创建用于向数据库发送SQL的Statement对象
- 从代表结果集的ResultSet中取出数据
- 断开与数据库的连接，并释放相关资源

JDBC的API

- `DriverManager`:驱动管理类
  - 主要作用:
    - 注册驱动
      - 实际开发中注册驱动会使用如下的方式
      - ```
        Class.forName("com.mysql.jdbc.Driver");
        ```
    - 获得连接
      - ```
        Connection conn = DriverManager.getConnection(String url,String username,String password);
        ```
- `Connection`:连接对象
  - 主要作用
    - 创建执行SQL语句的对象
      - ```
        Statement createStatement()   //执行SQL语句,有SQL注入的漏洞存在
        ```
      - ```
        PreparedStatement prepareStatement(String sql)   //预编译SQL语句，解决SQL注入的漏洞
        ```
      - ```
        CallableStatement prepareCall(String sql)          //执行SQL中存储过程
        ```
    - 进行事务的管理
      - ```
        setAutoCommit(boolean autoCommit)          //设置事务是否自动提交
        ```

```
commit()                //事务提交
```

```
rollback()              //事务回滚
```

- `Statement`:执行SQL
  - 主要作用
    - 执行SQL语句

```
boolean execute(String sql)          //执行SQL
```

```
ResultSet executeQuery(String sql)     //执行SQL中的select语句
```

```
int executeUpdate(String sql)          //执行SQL中的insert/update/delete语句
```

    - 执行批处理操作

```
addBatch(String sql)               //添加到批处理
```

```
executeBatch()                     //执行批处理
```

```
clearBatch()                    //清空批处理
```

- `ResultSet`:结果集
  - 结果集：其实就是查询语句(select)结果的封装
  - 结果集获取查询到的结果
  - 主要作用：

```
next()
```

    - 针对不同类型的数据，可以使用getXX()获取数据，通用的获取数据方法`getObject()`

JDBC的资源释放

- JDBC程序运行完后，切记要释放程序在运行过程中，创建的那些与数据库进行交互的对象，这些对象通常是ResultSet,Statement和Connection对象
- 特别是Connection对象，它是非常稀有的资源，用完后必须马上释放，如果Connection不能及时、正确的关闭，极易导致系统宕(dang)机，Connection的使用原则是尽量晚创建，早释放

JDBC的CRUD操作

- 向数据库中保存记录

```
    String sql = "insert into goods(name,price,desp) values('耳机',200.0,'蓝牙耳机')";
              int i = stmt.executeUpdate(sql);
              if (i >0 ){
                  System.out.println("插入成功");
              }
```

- 修改数据库中的记录

```
String sql = "update goods set desp='冷暖空调' where id =4";
           int i = stmt.executeUpdate(sql);
           if (i >0 ){
               System.out.println("修改成功");
           }
```

- 删除数据库中的记录

```
String sql = "delete from user where uid =4 ";
          int i = stmt.executeUpdate(sql);
          if (i>0){
              System.out.println("删除成功");
          }
```

- 查询数据库中的记录

JDBC的工具类的抽取

- 为了简化JDBC的开发，可以将一些重复的代码进行抽取
  - JDBCUtils
  - jdbc.properties

JDBC的SQL注入漏洞

SQL注入漏洞的解决

- PreparedStatement是Statement的子接口，它的实例对象可以通过调用Connection.preparedStatement(sql)方法获得，相对于Statement对象而言：
  - PreperedStatement可以避免SQL注入的问题。
  - Statement会使数据库频繁编译SQL，可能造成数据库缓冲区溢出。PreparedStatement 可对SQL进行预编译，从而提高数据库的执行效率。
  - 并且PreperedStatement对于sql中的参数，允许使用占位符的形式进行替换，简化sql语句的编写。

```java
/**
    * 避免SQL漏洞注入的方法
    */
   public static boolean login2(String username, String password) {
       Connection conn = null;
       PreparedStatement pstmt = null;
       ResultSet rs = null;
       boolean flag = false;
       try {
           conn = JDBCUtils.getConnection();
           String sql = "select * from user where username = ? and password= ?";
           //预处理SQL
           pstmt = conn.prepareStatement(sql);
           //设置参数
           pstmt.setString(1, username);
           pstmt.setString(2, password);
           //执行sql
           rs = pstmt.executeQuery();
           if (rs.next()) {
               flag = true;
           } else {
               flag = false;
           }
       } catch (Exception e) {
           e.printStackTrace();
       } finally {
           JDBCUtils.release(rs, pstmt, conn);
       }
       return flag;
   }
```

PreparedStatement的使用

- 保存数据

```
@Test
   /***
    * 保存数据
    */
   public void demo1(){
       Connection conn = null;
       PreparedStatement pstmt = null;
       try{

           conn =JDBCUtils.getConnection();
           String sql = "insert into user values(null,?,?,?)";

           pstmt =conn.prepareStatement(sql);
           pstmt.setString(1,"qqq");
           pstmt.setString(2,"123");
           pstmt.setString(3,"田七");

           int i = pstmt.executeUpdate();
           if (i>0){
               System.out.println("保存成功");
           }else {
               System.out.println("保存失败");
           }
       }catch(Exception e){
           e.printStackTrace();
       }finally {
           JDBCUtils.release(pstmt,conn);
       }
   }
```

```

```

- 修改数据

```
@Test
  /**
   * 修改数据
   * */
  public void demo2(){
      Connection conn =null;
      PreparedStatement pstmt =null;
      try{
          conn= JDBCUtils.getConnection();

          String sql = "update user set username= ? ,password =? ,name=?  where
uid=?";
          pstmt=conn.prepareStatement(sql);

          pstmt.setString(1,"iii");
```

```
            pstmt.setString(2,"123456");
            pstmt.setString(3,"王八");
            pstmt.setInt(4,7);

            int i = pstmt.executeUpdate();
            if (i>0){
                System.out.println("修改成功");
            }else {
                System.out.println("修改失败");
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally {
            JDBCUtils.release(pstmt,conn);
        }
    }
```

```
1
```

- 删除记录
  - ```
    @Test
    /**
     * 删除记录
     * */
    public void demo3(){
        Connection conn = null;
        PreparedStatement pstmt = null;

        try{
            conn = JDBCUtils.getConnection();

            String sql = "delete from user where uid =?";

            pstmt =conn.prepareStatement(sql);
            pstmt.setInt(1,7);
            int i = pstmt.executeUpdate();
            if (i>0){
                System.out.println("删除成功");
            }else {
                System.out.println("删除失败");
            }
        }catch(Exception e){
            e.printStackTrace();
        }finally {
            JDBCUtils.release(pstmt,conn);
        }
    }

    ```

- 查询数据
    ```

```java
@Test
  /**
   * 查询所有数据
   * */
  public void demo4(){
      Connection conn = null;
      PreparedStatement pstmt= null;
      ResultSet rs = null;
      try{
          conn = JDBCUtils.getConnection();
          String sql = "select * from user";

          pstmt = conn.prepareStatement(sql);

          rs= pstmt.executeQuery();
          while (rs.next()){
              System.out.println(rs.getInt("uid")+ "   "+
rs.getString("username")+"   "+rs.getString("password")+"
"+rs.getString("name"));

          }
      }catch(Exception e){
          e.printStackTrace();
      }finally {
          JDBCUtils.release(rs,pstmt,conn);
      }
  }

  @Test
  /**
   * 查询单个记录
   * */
  public void demo5(){
      Connection conn = null;
      PreparedStatement pstmt =null;
      ResultSet rs =null;
      try{
          conn =JDBCUtils.getConnection();

          String sql ="select * from user where uid = ?";

          pstmt =conn.prepareStatement(sql);
          pstmt.setInt(1,3);
          rs = pstmt.executeQuery();
          while (rs.next()){
              System.out.println(rs.getInt("uid")+"   "+rs.getString("username")+"
  "+rs.getString("password")+"   "+rs.getString("name"));
          }
      }catch(Exception e){
          e.printStackTrace();
      }finally {
          JDBCUtils.release(rs,pstmt,conn);
      }
```

```
51  }
```

```
1
```

数据库连接池

连接池是创建和管理一个连接的缓冲池的技术，这些连接准备好被任何需要它们的线程使用

应用程序直接获取连接的缺点：用户每次请求都需要向数据库获得连接，而数据库创建连接通常需要消耗相对较大的资源，创建时间也较长。假设网站一天10万访问量，数据库服务器就需要创建10万次连接，极大地浪费数据库的资源，并且极易**造成数据库服务器内存溢出。**

- 连接池架包
- 连接池工具类
- 连接池配置文件
- 连接池使用