

# SQL注入问题

## 1、小结

- **MVC架构: model view controller**

- model: 数据模型层(dao层)
- view: 视图层
- controller: 控制层

- **包:**

- org.lanqiao.entity
- org.lanqiao.dao
  - impl
- org.lanqiao.utils
- org.lanqiao.test

- **jdbcUtils: 工具类**

- 方法一般都是静态的: 使用类中方法时, 无需创建该类对象, 可直接通过: 类名. 的形式调用该方法
- 静态只能使用静态
- 与数据库连接的属性:都定义为静态的
  - driverClassName: 类加载器, 放在静态代码块中随着类的加载而加载, 无需调用
  - url
  - user
  - password
- 其中所用静态方法
  - getConnection(): 获取连接
  - releaseSource(ResultSet res, Statement statement, Connection connection): 释放连接
- jdbc代码重构: 为使数据库使用更加灵活, 便于切换数据库及实施, 将连接数据库的属性写在配置文件中: xxx.properties
  - 其配置都是以: 键值对的形式存在
  - 键名 = 键值(不需要: 分号、冒号)
- 对于属性文件的读取加载
  - 首先加载文件, 得到一个InputStream 输入流对象, 对于文件的读取, 都是以IO流进行操作

```
InputStream in = Utils.class.getClassLoad().getResourceAsStream("属性配置文件名");
```

- 创建 Properties对象

```
Properties properties = new Properties();
```

- 将IO流加载进来

```
properties.load(in);
```

- 通过get方法获取键所对应的值

```
properties.getProperties("属性文件中的键");
```

- 当当

- 当当

## • 当当

## 2、SQL注入问题及解决

- 使用Statement对象存在SQL注入问题，Test类中：

```
public class Test {  
    public static void main(String[] args) throws SQLException {  
        Scanner scanner = new Scanner(System.in);  
        System.out.println("请输入您的用户名: ");  
        String username = scanner.nextLine();  
        System.out.println("请输入您的密码: ");  
        String password = scanner.nextLine();  
        IUser iUser = new UserDaoImpl();  
        User user = iUser.getUserByUsernameAndPassword(username, password);  
        if (user != null){  
            System.out.println("恭喜您登陆成功");  
        }else {  
            System.out.println("您的账号或密码有误, 请重新登陆");  
        }  
        scanner.close();  
    }  
}
```

问题如图所示：

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
JDBC-demo2 src org lanqiao test Test
UserDaoImpl.java Test.java IUser.java
11 public static void main(String[] args) throws SQLException {
12     Scanner scanner = new Scanner(System.in);
13     System.out.println("请输入您的用户名:");
14     String username = scanner.nextLine();
15     System.out.println("请输入您的密码:");
16     String password = scanner.nextLine();
17     IUser iUser = new UserDaoImpl();
18     User user = iUser.getUserByUsernameAndPassword(username, password);
19     if (user != null) {
    Test > main()
Run: Test x
D:\eclipse\JDK\bin\java.exe ...
请输入您的用户名:
root 'or '1=1' SQL注入
请输入您的密码:
aaaaaaaaaaaaaaaaaaaaa
恭喜您登陆成功
Process finished with exit code 0
```

```
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
JDBC-demo2 src org lanqiao test Test
UserDaoImpl.java Test.java IUser.java
16 String password = scanner.nextLine();
17 IUser iUser = new UserDaoImpl();
18 User user = iUser.getUserByUsernameAndPassword(username, password);
19 if (user != null) {
20     System.out.println("恭喜您登陆成功");
21 } else {
22     System.out.println("您的账号或密码有误, 请重新登陆");
23 }
24 scanner.close();
    Test > main()
Run: Test x
D:\eclipse\JDK\bin\java.exe ...
请输入您的用户名:
root
请输入您的密码:
1234567890123456
恭喜您登陆成功
Process finished with exit code 0
```

- 使用PreparedStatement对象解决Sql注入

```
@Override
    public User getUserByUsernameAndPassword(String name, String password) throws
        SQLException {
        Connection connection = JDBCUtils.getConnection();
        //Statement statement = connection.createStatement();
        //String sql = "select * from stu where name = '"+name+"' and password
        = '"+password+"'";
        String sql = "select * from stu where name = ? and password = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString(1, name);
        preparedStatement.setString(2, password);
        ResultSet resultSet = preparedStatement.executeQuery();
        User user = null;
        while (resultSet.next()){
            user = new User();
            user.setId(resultSet.getInt("id"));
            user.setName(resultSet.getString("name"));
            user.setAge(resultSet.getInt("age"));
            user.setPassword(resultSet.getString("password"));
            user.setSex(resultSet.getString("sex"));
        }
        return user;
    }
}
```

## • Statement和PreparedStatement的区别

- PreparedStatement代码的可读性和维护性高(Sql模板, 使用占位符表示参数).
  - PreparedStatement能保证安全性, 可放在SQL注入
  - 当当
- 当当

## 事务：对数据库的一次完整性操作

### 事务的特性：

- 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：如果事务执行之前数据库是一个完整性的状态,那么事务结束后,无论事务是否执行成功,数据库仍然是一个完整性状态. 数据库的完整性状态:当一个数据库中的所有的数据都符合数据库中所定义的所有的约束,此时可以称数据库是一个完整性状态.
- 隔离性：事务的隔离性是指多个用户并发访问数据库时，一个用户的事务不能被其它用户的事务所干扰，多个并发事务之间事务要隔离
- 持久性：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，接下来即使数据库发生故障也不应该对其有任何影响

## 1、事务管理操作

- MySql的事务

- 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。
- 开启事务：connection.setAutoCommit(false)，数据库中事务默认为自动提交，需要设置为手动提交 UserDaoImpl类

```
@Override
public void insertUser(String name ,String password) {
    Connection connection = JDBCUtils.getConnection();
    String sql = "insert into stu(name,password) values(?,?)";
    PreparedStatement preparedStatement = null;
    try {
        connection.setAutoCommit(false);//关闭数据库事务的自动提交
        preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString(1,name);
        preparedStatement.setString(2,password);
        System.out.println(1/0);
        preparedStatement.executeUpdate();
        connection.commit();//手动提交
    } catch (SQLException e) {
        e.printStackTrace();
        try {
            //当数据在更新操作中，一旦发生异常，则执行回滚
            connection.rollback();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    } finally {
        //释放资源
        JDBCUtils.releaseSource(null,preparedStatement,connection);
    }
}
```

Test类

```
public class Test {
    public static void main(String[] args) throws SQLException {
        IUser iUser = new UserDaoImpl();
        /*Scanner scanner = new Scanner(System.in);
        System.out.println("请输入您的用户名: ");
        String username = scanner.nextLine();
        System.out.println("请输入您的密码: ");
        String password = scanner.nextLine();
        User user = iUser.getUserByUsernameAndPassword(username,password);
        if (user != null){
            System.out.println("恭喜您登陆成功");
        }else {
            System.out.println("您的账号或密码有误，请重新登陆");
        }
        scanner.close();*/
        iUser.insertUser("小米","ddd");
    }
}
```

```
}  
}
```

- 结束事务
  - `connection.commit()`//提交
  - `connection.rollback()`//回滚
- 格式
  - try开始: 开启事务
  - try结束: 提交事务
  - catch中: 回滚事务
  - 代码

```
try{  
    con.setAutoCommit(false);//开始事务  
    ...  
    con.commit();//提交  
}catch(...){  
    con.rollback();//回滚事务  
}
```

- 三种并发读问题
  - 脏读: 读取到另一个事务未提交的数据
  - 不可重复读: 两次读取不一样
  - 幻读: 读取到另一个事务已提交数据
- 四种隔离级别
- 事务综述和处理: 如何在代码中去处理事务?
  - 在JDBC中, 事务是默认提交的, 必须先设置事务为手动提交。

```
Connection对象.setAutoCommit(false)
```

- 手动提交事务: `connection对象.commit()`
  - 若出现异常必须回滚事务, 不回滚事务, 总余额毅然整却, 若不回滚事务, 不释放数据库资源
  - 当当
- 当当

## 2、批处理操作

## 3、大数据类型操作

- ☐ 通常值很长的字符类型文件, 内容由字符组成
- ☐ 表

字符	数据大小	Mysql	Oracle
	小	char\varchar	varchar
字节	大	text/lo/text	clob
	大		

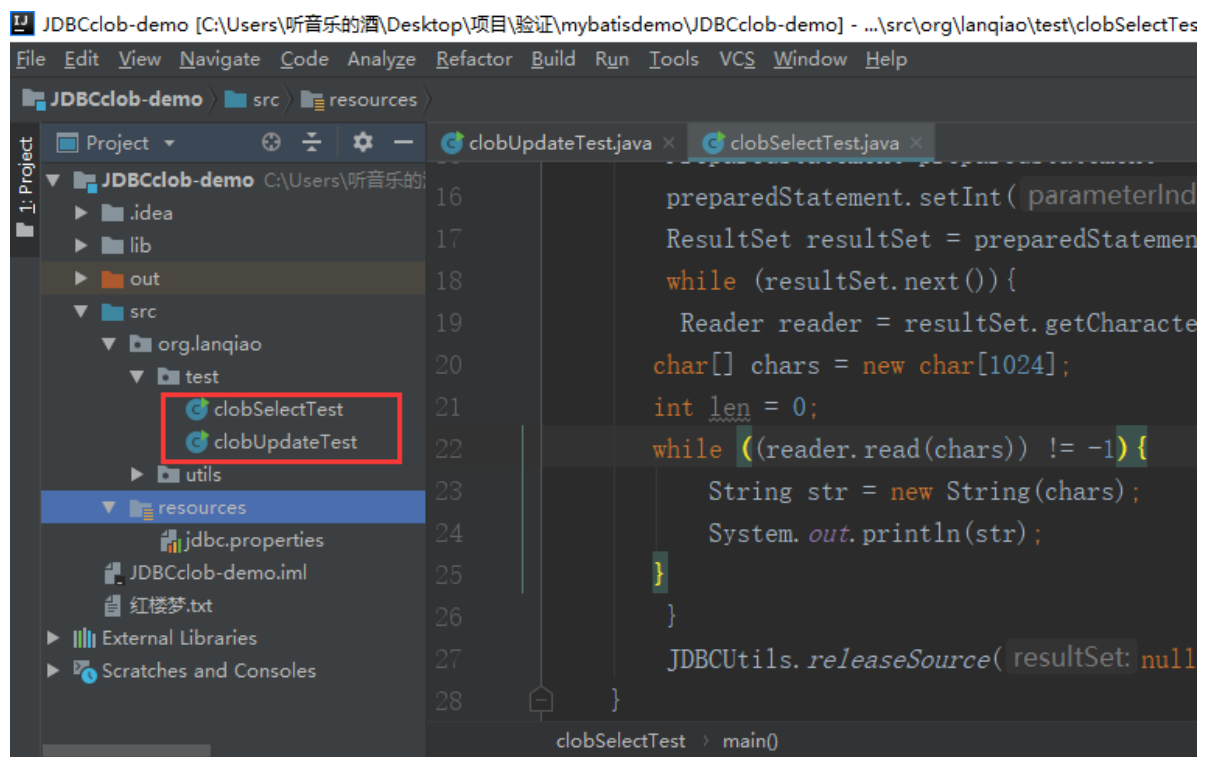
## ☐ 数据库设计

表代码:

## ☐ 数据操作

### ☐ Clob操作

项目结构图:



### ☐ clobSelectTest类

```
public class clobSelectTest {
    public static void main(String[] args) throws SQLException, IOException {
        Connection connection = JDBCUtils.getConnection();
        String sql = "select * from t_clob where id = ?";
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setInt(1, 1); // 传一个参数
        ResultSet resultSet = preparedStatement.executeQuery(); // 执行查询, 得到一个结果集
        while (resultSet.next()) {
            Reader reader = resultSet.getCharacterStream("resume"); // 获取Read对象
            char[] chars = new char[1024];
            int len = 0;
```

```

while ((reader.read(chars)) != -1){
    String str = new String(chars);
    System.out.println(str);
}
}
JDBCUtils.releaseSource(null,preparedStatement,connection);
}
}

```

☐ clobUpdateTest类

```

public class clobUpdateTest {
    public static void main(String[] args) throws SQLException, FileNotFoundException {
        Connection connection = JDBCUtils.getConnection();
        String sql = "insert into t_clob(resume) values(?)";
        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        File file = new File("红楼梦.txt");//获取文件
        InputStream inputStream = new FileInputStream(file);//获取输入流
        preparedStatement.setAsciiStream(1,inputStream,file.length());
        preparedStatement.executeUpdate();
        JDBCUtils.releaseSource(null,preparedStatement,connection);
    }
}

```

☐ Blob操作

☐ 当当

## 4、获取自动生成的主键

## 5、连接池：DataSource

### • 连接池思想

连接池思想

为什么必须使用数据库连接池：

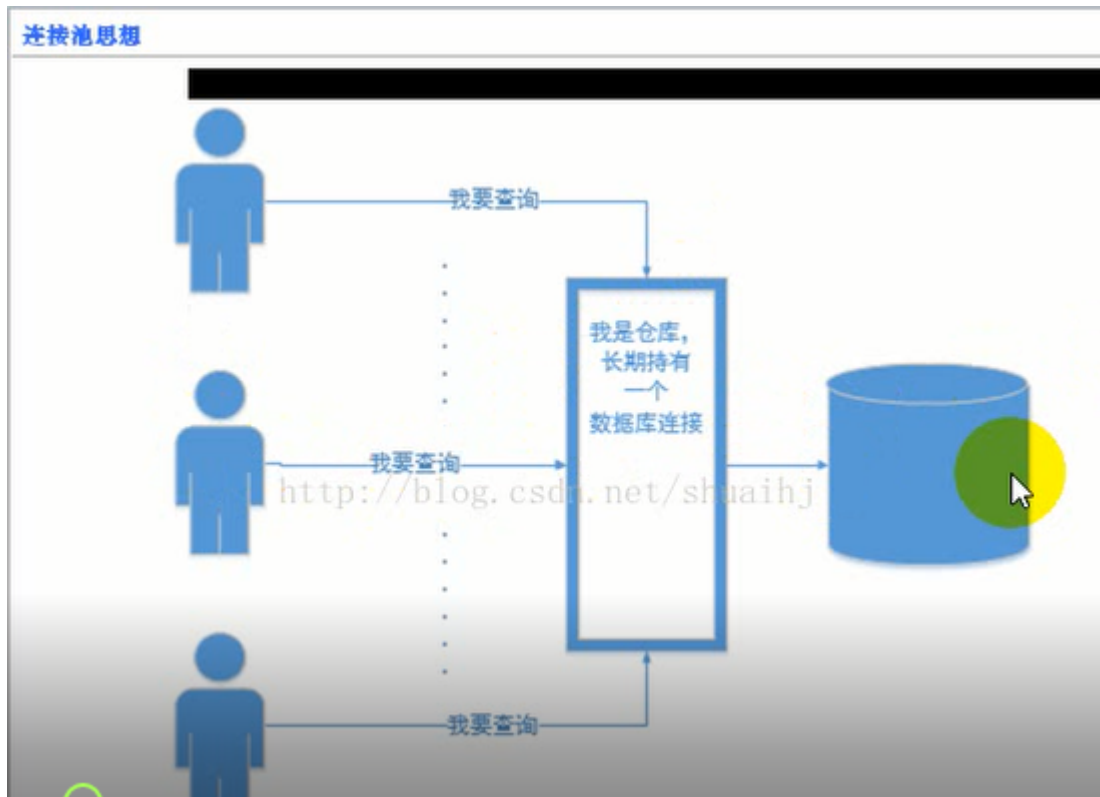
普通的JDBC数据库连接使用 DriverManager 来获取，每次向数据库建立连接的时候都要将 Connection 加载到内存中，再验证用户名和密码(得花费0.05s~1s的时间)。需要数据库连接的时候，就向数据库要求一个，执行完成后断开连接。这样的方式将会消耗大量的资源和时间。数据库的连接资源并没有得到很好的重复利用。若同时有几百人甚至几千人在线，频繁的进行数据库连接操作将占用很多的系统资源，严重的甚至会造成服务器的崩溃。

对于每一次数据库连接，使用完后都得断开。否则，如果程序出现异常而未能关闭，将会导致数据库系统中的内存泄漏，最终将导致重启数据库。

这种开发不能控制被创建的连接对象数，系统资源会被毫无顾及的分配出去，如连接过多.也可能导致内存泄漏，服务器崩溃



数据库连接”是一种稀缺的资源，为了保障网站的正常使用，应该对其进行妥善管理。其实我们查询完数据库后，如果不关闭连接，而是暂时存放起来，当别人使用时，把这个连接给他们使用。就避免了一次建立数据库连接和断开的操作时间消耗。原理如下：



#### 数据库连接池作用

连接池思想

由于数据库连接得到重用，避免了频繁创建、释放连接引起的大量性能开销。在减少系统消耗的基础上，增进了系统环境的平稳性（减少内存碎片以及数据库临时进程、线程的数量）

②更快的系统响应速度

数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于池内备用。此时连接池的初始化操作均已完成。对于业务请求处理而言，直接利用现有可用连接，避免了数据库连接初始化和释放过程的时间开销，从而缩减了系统整体响应时间。

③新的资源分配手段

对于多应用共享同一数据库的系统而言，可在应用层通过数据库连接的配置，实现数据库连接技术。

数据库连接池

#### ④统一的连接管理，避免数据库连接泄露

在较为完备的数据库连接池实现中，可根据预先的连接占用超时设定，强制收回被占用的连接，从而避免了常规数据库连接操作中可能出现的资源泄露

java提供的连接池接口：`javax.sql.DataSource`，连接池厂商的连接池类需要实现这一接口。

#### • 连接池概述

在Java中, 连接池使用`javax.sql.DataSource`接口来表示连接池.

注意:`DataSource`仅仅只是一个接口, 由各大服务器厂商来实现(Tomcat, JBoss).

常用的`DataSource`的实现:

DBCP: Spring推荐的

C3P0: Hibernate推荐的

`DataSource` (数据源) 和连接池 (Connection Pool) 是同一个.

=====

使用连接池和不使用连接池的区别在哪里?

从代码上:

不使用连接池: `Conenction`对象由`DriverManager`获取.

```
Connection conn = DriverManager.getConnection(url, username, password);
```

不使用连接池: `Conenction`对象由`DriverManager`获取.

```
Connection conn = DriverManager.getConnection(url, username, password);
```

使用连接池:

如何创建`DataSource`对象, 如何在`DataSource`中设置url, 账号, 密码.

```
Connection conn = DataSource对象.getConnection();
```

-----

使用连接池的时候:

释放资源: `Connection`对象.`close()`:

是把`Connection`放回给连接池, 而不是和数据库断开.

## • dbcp连接池

- 硬编码版本

- 图示

## dbcp连接池

准备:

拷贝jar:

- commons-dbcp-1.3.jar 连接池的实现
- commons-pool-1.5.6.jar .连接池实现的依赖库

查阅文档:commons-dbcp-1.3-src\doc\BasicDataSourceExample.java

---

解决DBCP的硬编码问题:

应该把连接信息放到配置文件中去: 配置文件的名词可以任意.

但是配置文件中的key, 必须是BasicDataSource对象的属性(setXxx方法决定的属性);

---

## dbcp连接池

dbcp.properties

#连接字符串

url=jdbc:mysql://localhost:3306/jdbcdemo

#用户名

username=root

#密码

password=admin

#驱动的路径

driverClassName=com.mysql.jdbc.Driver

#连接池启动时的初始值

initialSize=1

#连接池的最大值

数据库连接池



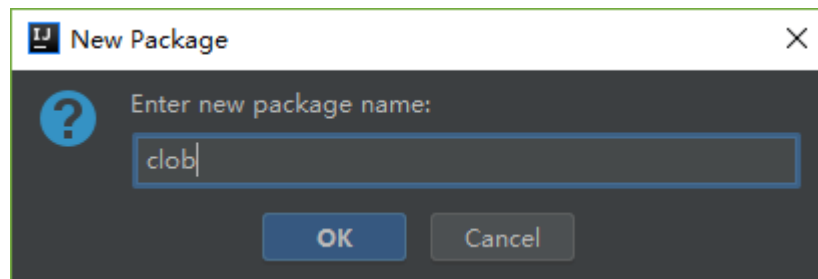
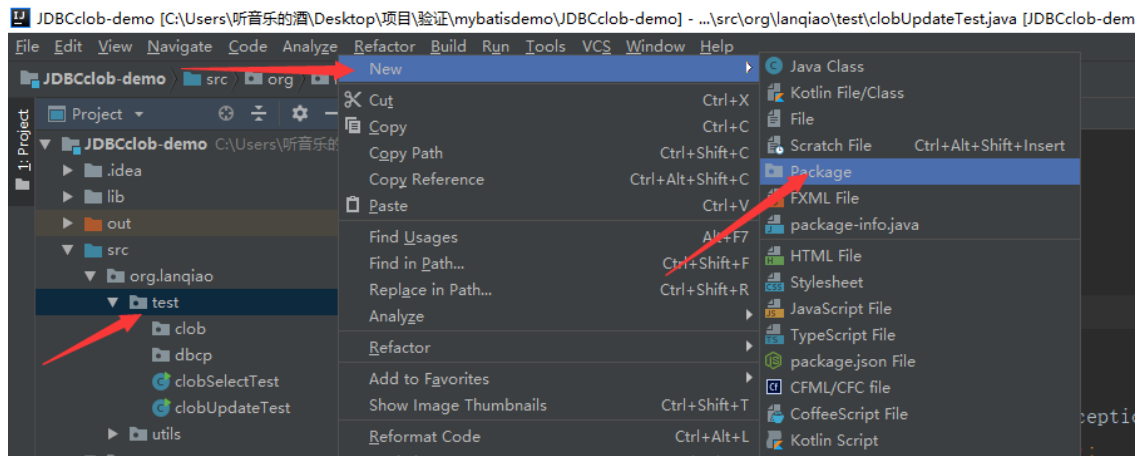
```
Connection con = ds.getConnection();
```

```
BasicDataSource ds = null;  
//创建数据源对象  
ds = new BasicDataSource();  
//设置连接数据库的 驱动  
ds.setDriverClassName("com.mysql.jdbc.Driver");  
//设置连接数据库的 url  
ds.setUrl("jdbc:mysql://localhost:3309/test");  
//设置连接数据库的 用户名  
ds.setUsername("root");  
//设置连接数据库的 密码  
ds.setPassword("1230");  
//设置数据库连接池的 初始连接数  
ds.setInitialSize(5);  
//设置连接池最多可有多少个活动连接数  
ds.setMaxActive(20);  
//设置连接池中最少有多少个空闲连接
```

```
ds.setMinIdle(2);  
//设置连接数据库的 驱动  
ds.setDriverClassName("com.mysql.jdbc.Driver");  
//设置连接数据库的 url  
ds.setUrl("jdbc:mysql://localhost:3309/test");  
//设置连接数据库的 用户名  
ds.setUsername("root");  
//设置连接数据库的 密码  
ds.setPassword("1230");  
//设置数据库连接池的 初始连接数  
ds.setInitialSize(5);  
//设置连接池最多可有多少个活动连接数  
ds.setMaxActive(20);  
//设置连接池中最少有多少个空闲连接  
ds.setMinIdle(2);
```

```
Connection conn = null;  
try {
```

- 项目中添加一个包的示意图：



#### ■ 代码

```
public static void main(String[] args) throws SQLException, IOException {  
    //创建数据源对象  
    BasicDataSource basicDataSource = new BasicDataSource();  
    //创建链接数据源的驱动  
    basicDataSource.setDriverClassName("com.mysql.jdbc.Driver");  
    basicDataSource.setUrl("jdbc:mysql://localhost:3306/test");//设置连接数据库的url  
    basicDataSource.setUsername("root");  
    basicDataSource.setPassword("root");  
    basicDataSource.setMaxActive(20);//最多有多少个活动链接  
    basicDataSource.setMaxIdle(10);  
    basicDataSource.setMinIdle(2);//最少有2个空闲连接  
    basicDataSource.setMaxWait(1000);  
    //创建连接  
    Connection connection = basicDataSource.getConnection();  
    System.out.println(connection);  
}
```

#### ■ 当当

##### ○ 使用properties解耦

```
public static void main(String[] args) throws SQLException, IOException {  
    //创建数据源对象  
    BasicDataSource basicDataSource = new BasicDataSource();  
    /*//创建链接数据源的驱动  
    basicDataSource.setDriverClassName("com.mysql.jdbc.Driver");  
    basicDataSource.setUrl("jdbc:mysql://localhost:3306/test");//设置连接数据库的url  
    basicDataSource.setUsername("root");
```

```

        basicDataSource.setPassword("root");
        basicDataSource.setMaxActive(20); //最多有多少个活动链接
        basicDataSource.setMaxIdle(10);
        basicDataSource.setMinIdle(2); //最少有2个空闲连接
        basicDataSource.setMaxWait(1000); */
        //去掉硬编码，加载配置文件
        InputStream inputStream =
        DBCPTest.class.getClassLoader().getResourceAsStream("dbcp.properties");
        Properties properties = new Properties();
        properties.load(inputStream);

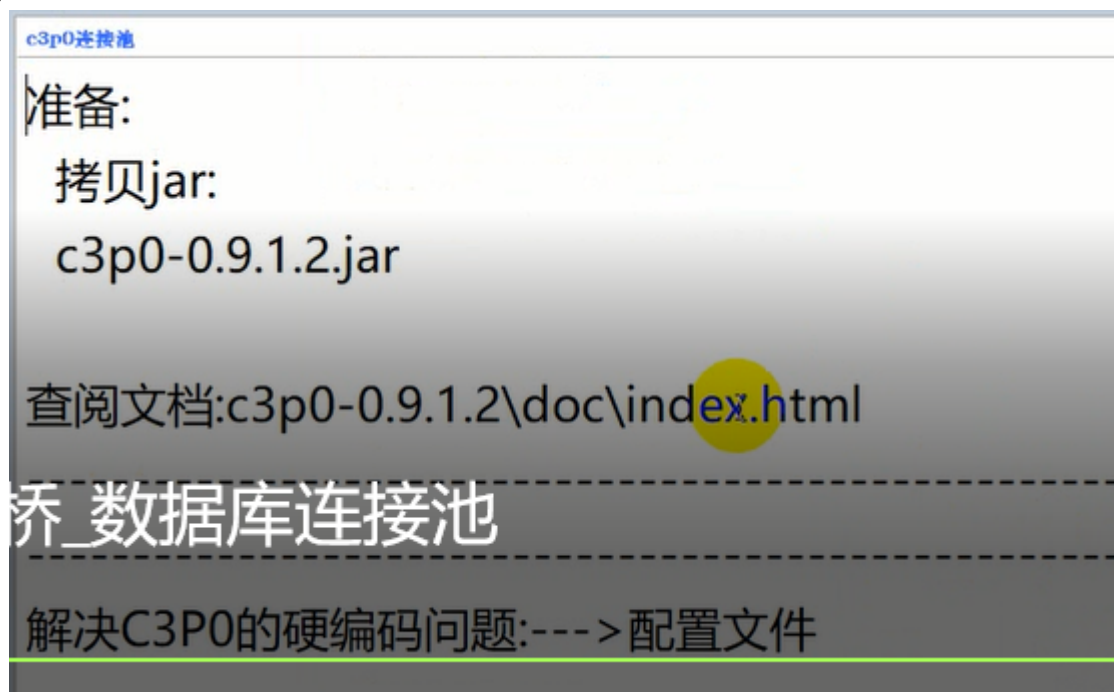
        basicDataSource.setDriverClassName(properties.getProperty("username"));
        //创建连接
        Connection connection = basicDataSource.getConnection();
        System.out.println(connection);
    }

```

◦ 当当

## • c3p0连接池

示图:



◦ 硬编码版本

```

public class C3P9Test {
    public static void main(String[] args) throws PropertyVetoException,
        SQLException {
        ComboPooledDataSource comboPooledDataSource = new
        ComboPooledDataSource();
        comboPooledDataSource.setDriverClass("com.mysql.jdbc.Driver");
        comboPooledDataSource.setJdbcUrl("jdbc:mysql://localhost:3306/test");
        comboPooledDataSource.setUser("root");
        comboPooledDataSource.setPassword("root");
        comboPooledDataSource.setInitialPoolSize(10);
    }
}

```

```

        comboPooledDataSource.setMaxPoolSize(20);
        comboPooledDataSource.setMaxIdleTime(20);
        Connection connection = comboPooledDataSource.getConnection();
        System.out.println(connection);
    }
}

```

- 使用properties解耦

```

//properties解耦

```

- 小结

对与大数据的处理 mysql text longtext oracle clob

```

mysql Blob
oracle blob

```

执行插入操作: Clob :insert 1 获取一个文件的输入流 2 两个放过

setStream(columnIndex,inputStream ,file.length()) Clob:查询: Reader /InputStream getStream();

Blob oracle insert into tablename(字段列表) values(empty\_blob)

需要获取指针 在select 语句后跟一个for update 获取文件流 set\*\*Stream(columnIndex,inputStream ,file.length()) 数据库连接池: 使用数据库连接池: 可以减少链接的创建所产生的资源消耗和时间消耗 从而提高数据库的链接效率 DBCP C3P0: 使用步骤: 1 需要下载所使用的jar包 2 将jar包引入到项目中, 进行builderPath 加入到class类路径 3 需要获取数据源 (DataSource) 4 可以通过datasource来获取链接 连接池的作用: 根据初始化的配置信息, 在程序启动时, 创建一定数量的链接, 存在与连接池中, 当我们需要链接的时候, 不需要去重新创建, 直接从连接处中进行获取。当链接使用完之后, 释放链接的, 不是将链接和数据库直接断开, 而是将链接重新放回到 连接池中, 以供下一个链接访问来调用。

- 当当

## • DRUID: 在分布式系统中, 有一个连接池比较常用 druid

```

public static void main(String[] args) throws SQLException {
    //获取数据源
    DruidDataSource dds = new DruidDataSource();
    dds.setDriverClassName("com.mysql.jdbc.Driver");
    dds.setUrl("jdbc:mysql://localhost:3306/test");
    dds.setUsername("root");
    dds.setPassword("root");
    dds.setMaxActive(20);
    dds.setInitialSize(20);
    //创建连接
    Connection connection = dds.getConnection();
    System.out.println(connection);
}

```

- 代码重构

## 6、当当

