【手把手】JavaWeb 入门级项目实战 -- 文章发布系统 （第五节）

在上一节中，我们成功将数据从前台的JSP页面传递到了controller层，但是还没有写service层，老实说还有很多工作没有，尤其是和数据库的链接方面的，所以，这一节，我们专门来处理一下关于数据库连接方面的东西。

01 序言

你可能之前听过了很多新名词，比如数据源，连接池，还有c3p0等等。作为新手很容易被这些名词给吓到，因为一般的培训机构不会告诉你这些，他们仅仅是给你讲了最基本的jdbc，一般来说，就是告诉你用Java代码来操作数据库的几个步骤。

首先，加载驱动类（妈了个鸡蛋糕，对初学者而言，很可能连反射都不知道，就在那学习加载什么驱动类了，天知道学了些什么，越听越糊涂）。

我有很多去培训机构的朋友就是如此，培训都结束了，也不知道为什么要学**jdbc**，只会课堂上那几个干巴巴的案例。去企业面试都只有被嘲笑的份。

然后，加载好了驱动类，需要用一个什么 **DriverManager** 来获取链接。再接着，就写预处理语句，执行一下，拿到结果集 **ResultSet** 。（妈了个鸡蛋糕，对初学者而言，都不知道基本的迭代器，循环，就在那写什么 **while(rs.next()){ ... }** 了，最终的结果可想而知。

我个人是不太主张去那些不太出名的，或者口碑很差的培训机构，如果一定要去培训，那么起码找个靠谱的吧，毕竟花这么多钱呢！如果花了钱，大部分知识竟然还是要去企业里面从0学起，到那时候你真的是要多郁闷有多郁闷。

不过也没办法，因为就算是再差的培训机构，教的也总是要比学校实用一些。每年都有那么多毕业生毕业后不知道干嘛，说起来真是挺郁闷的。

去找工作吧，屡屡碰壁，好不容易找到了一个面试的，过去一问，原来是培训的。

然后想想自己确实没有什么技术，所以只好咬咬牙花一万多块钱去培训了。

这样的例子太多了，博主的很多朋友和同学就是这样走上了Java程序开发的道路。

我不反对去培训机构学习，只是我想说，如果自制力好的话，还是尽量自学比较划算。

如果实在没有自控力，去培训机构也是一个不错的选择。

总之，视你的具体情况而定吧。

OK，回到正题。

最后，在jdbc操作的末尾，总归少不了关闭链接，否则会导致资源的浪费。

当然，这些东西的确很重要，jdbc是Java代码访问和操作数据库的方式。这些基本的API方法也形成了所有JDBC框架的根，比如**Spring-jdbc**就是对这些相对底层的代码的封装。

这就是所谓的jdbc1.0规范，但这样有一个缺点，就是我每次进行数据库操作都要去获取一个连接，然后再自己关掉。这样很麻烦，人总是聪明的，所以为了改进，就有了jdbc2.0规范。

那些数据源，连接池的概念就属于jdbc2.0规范，就是说，我先创建好一大堆连接，谁要用谁就去拿。这样就解决了jdbc1.0的弊端，不需要用户每次都去创建连接，最后再关闭链接了。

因为这个小项目毕竟是以基础优先的，所以我还是介绍一下jdbc1.0的使用方法吧。我会逐步封装出一个JDBC工具类出来，我以前也没写过，都是直接用框架的，这是我的第一次尝试，难免会有不恰当的地方，还希望各位多多包涵。

如有疑问或者不同的见解，可以在评论区指出，我会虚心改正的，谢谢。

02 预备知识

最好对以下知识有一个初步的了解后，再来看本篇文章，当然，不看也没关系啦。。。

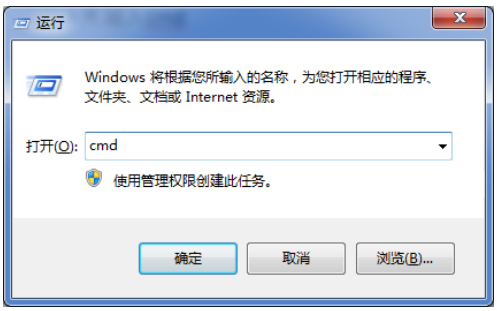
1、反射
2、泛型
3、JDBC API简单使用
4、properties 文件的读取
5、 IO流

03 准备好mysql

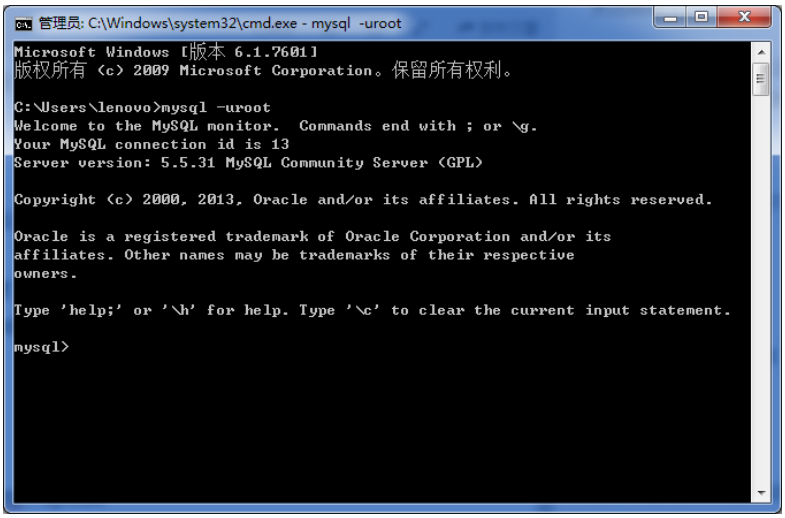
好了，正式开始吧。我在本地已经安装好了mysql。mysql安装的话，随便百度一下就可以了。

在上一节中，我们新建了一个数据库Article，用户名就是root，密码没有。root用户享有本地mysql的最高权限。

win + R,输入cmd



输入 mysql -uroot (密码不需要填写)



输入 use Article

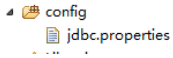
好了，数据库方面的工作就做好了。

如果无法进入mysql，可能是没有启动。在命令行输入：net start mysql即可。

04 编写jdbc.properties

我们要连接数据库，最基本的信息有用户名，密码，还有数据库名。

于是，我们在src目录上新建一个config源文件夹（注意，是源文件夹，不是普通的文件夹，所有的源文件夹不会真的产生一个文件夹，它里面的文件默认在 **CLASSPATH** 根目录下），再创建一个jdbc.properties文件:



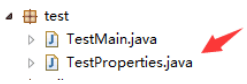
内容

```
db.username=root
db.password=
db.dataBaseName=article
```

OK，这一步也完成了，接下来就是如何读取这个文件里的信息。

05 读取properties文件

在test包下新建一个测试类，来试着读取properties文件。




```
package test;
/**
 * 读取Properties文件的测试类
 * @author 剽悍一小兔
 */
public class TestProperties {
    public static void main(String[] args) {

    }
}
```

properties文件（配置文件）就是一个类似于Map的东西，为什么要把连接信息放在文件里呢？那是因为，如果你写在Java类中，这固然是可以的。但是，不利于维护啊。

比如，项目一旦上线，基本上就是专门的维护人员在跟进了，一旦要改个什么配置信息，作为开发人员，你只需要和对方讲，找到一个什么什么properties文件，然后将某一行改掉就好了。如果你不用配置文件，直接将信息写在Java类中，那么你就很难描述清楚了，改起来也特别麻烦。

这就是原因。

以下是读取的代码，流程就是我先把这个文件变成一个输入流InputStream，然后new一个Properties，再去加载之前获得的输入流。

jdbc.properties是一个实实在在的文件。而且，它还是一个特殊的文件，因为它里面的内容都是 **key=value** 的形式。现在的问题就在于怎么用Java代码来读取里面的信息呢？

```
InputStream inputStream =  TestProperties.class.getClassLoader().getResourceAsStream("jdbc.properties");
Properties p = new Properties();
try {
    p.load(inputStream);
    System.out.println(p);
} catch (IOException e) {
    e.printStackTrace();
}
```

你可以把InputStream想象成一根吸管

```
InputStream inputStream =  TestProperties.class.getClassLoader().getResourceAsStream("jdbc.properties");
```

这行代码就相当于插了一根吸管在 jdbc.properties 文件上面，准备抽取里面的信息。

```
p.load(inputStream);
```

这行代码表示将吸管和 Properties 对象接通。

额，如果实在没有Java IO的基础，就暂时这么想象一下吧。。。

```
 System.out.println(p);
```

输出：
{db.password=, db.dataBaseName=article, db.username=root}

分开来打印：

```
System.out.println(p.getProperty("db.username"));
System.out.println(p.getProperty("db.password"));
System.out.println(p.getProperty("db.dataBaseName"));
```

输出：
root

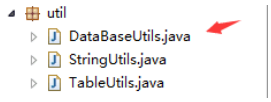
article

（因为密码为空，所以没显示出来）

这样的话，加载资源文件也没有问题了。

06 开始封装自己的DataBaseUtils

DataBaseUtils的意思就是数据库工具类，你可以把这个看成是一个自己的小框架。



我们已经知道要访问数据库的话，需要有username，password，还有dataBaseName。所以，这三个数据就作为工具类的属性吧。

```
private static String username; //用户名
private static String password; //密码
private static String dataBaseName; //数据库名
```

接下来，专门定义一个方法来加载properties。

```
/**
 * 配置数据库的基本信息
 * @return void
 */
public static void config(String path){
    InputStream inputStream = DataBaseUtils.class.getClassLoader().getResourceAsStream(path);
    Properties p = new Properties();
    try {
        p.load(inputStream);
        username = p.getProperty("db.username");
        password = p.getProperty("db.password");
        dataBaseName = p.getProperty("db.dataBaseName");
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

一旦调用了这个方法，那么就会给私有属性赋值。

为了方便起见，我们让DataBaseUtils类被加载的时候就自动配置 **jdbc.properties**，比较容易想到的一个方法就是定义一个static块，然后在里面调用一下 config 方法：

```
static {
    config("jdbc.properties");
}
```

这样一来，只要你调用了这个DataBaseUtils中的方法，就会自动配置连接信息了。

获取连接的方法：

```
/**
 * 获取数据库链接
 * @return Connection
 */
public static Connection getConnection(){
    Connection connection = null;
    try {
        Class.forName("com.mysql.jdbc.Driver");
        connection  = DriverManager.getConnection("jdbc:mysql://localhost:3306/"+dataBaseName+"?useUnicode=true&characterEncoding=utf8",username,password);
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return connection;
}
```

测试：

```
DataBaseUtils.config("jdbc.properties");
Connection conn = DataBaseUtils.getConnection();
System.out.println(conn);
```

结果：
com.mysql.jdbc.Connection@661532

这就说明成功获取连接了。

关闭连接和其他资源的方法

```
/**
 * 关闭资源
 * @param connection
 * @param statement
 * @param rs
 */
```

```
public static void closeConnection(Connection connection,PreparedStatement statement,ResultSet rs){
    try {
        if(rs!=null)rs.close();
        if(statement!=null)statement.close();
        if(connection!=null)connection.close();
    } catch (Exception e) {
        e.fillInStackTrace();
    }
}
```

07 DML操作的实现

DML表示—数据操纵语言，也就是SELECT，DELETE，UPDATE，INSERT。

现在我们开始封装DML的操作。

上代码：

```
/**
 * DML操作
 * @param sql
 * @param objects
 */
public static void update(String sql,Object...objects){
    Connection connection = getConnection();
    PreparedStatement statement = null;
    try {
        statement = (PreparedStatement) connection.prepareStatement(sql);
        for (int i = 0; i < objects.length; i++) {
            statement.setObject(i+1, objects[i]);
        }
        statement.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    }finally{
        closeConnection(connection, statement, null);
    }
}
```

Object...objects是变长参数，你可以把它理解成一个Object数组。

测试：

```
String id = UUID.randomUUID() + "";
String createTime = new SimpleDateFormat("yyyy-MM-dd").format(new Date());
update("INSERT INTO t_user(id,username,password,sex,create_time,is_delete,address,telephone) "
        + "VALUES (?, ?, ?, ?, ?, ?, ?, ?)", id,"张三",123456,0,createTime,0,"保密","保密");
```

运行结果：

```
com.mysql.jdbc.MysqlDataTruncation: Data truncation: Data too long for column 'id' at row 1
    at com.mysql.jdbc.MysqlIO.checkErrorPacket(MysqlIO.java:2983)
    at com.mysql.jdbc.MysqlIO.sendCommand(MysqlIO.java:1631)
    at com.mysql.jdbc.MysqlIO.sqlQueryDirect(MysqlIO.java:1723)
    at com.mysql.jdbc.Connection.execSQL(Connection.java:3283)
    at com.mysql.jdbc.PreparedStatement.executeInternal(PreparedStatement.java:1332)
    at com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1604)
    at com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1519)
    at com.mysql.jdbc.PreparedStatement.executeUpdate(PreparedStatement.java:1504)
    at util.DataBaseUtils.update(DataBaseUtils.java:147)
    at util.DataBaseUtils.main(DataBaseUtils.java:165)
```

发现报错了，不慌，看看它说什么。

因为是直播，所以如果我的代码报错了，也会把错误放上来，然后纠正。我也是普通人，很难保证代码一次性就写对。

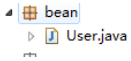
我知道，初学的时候看到报错就害怕，这是很正常的。没事，慢慢来。

好的，让我们看看它说什么。

```
Data truncation: Data too long for column 'id' at row 1
```

哦，这说明id的字段长度太短了，而我们生成的UUID太长。

找到User类



我们把id的长度调大一点，换成varchar(100)吧。

```
@Column(type = "varchar(100)" ,field = "id" ,primaryKey = true ,defaultNull = false)
private String id;          //主键，采用UUID
```

接着，修改一下TableUtils，将建表语句加一个删除旧表的判断。

```
public static String getCreateTableSQl(Class<?> clazz){
    StringBuilder sb = new StringBuilder();

    //获取表名
    Table table = (Table) clazz.getAnnotation(Table.class);
    String tableName = table.tableName();
    sb.append("DROP TABLE IF EXISTS ").append(tableName).append(";\n");
    sb.append("create table ");
    sb.append(tableName).append("(\n");

    Field[] fields = clazz.getDeclaredFields();
    String primaryKey = "";
    //遍历所有字段
    for (int i = 0; i < fields.length; i++) {
        Column column = (Column) fields[i].getAnnotations()[0];
        String field = column.field();
        String type = column.type();
        boolean defaultNull = column.defaultNull();
```

```
        sb.append("\t" + field).append(" ").append(type);
        if(defaultNull){
            if(type.toUpperCase().equals("TIMESTAMP")){
                sb.append(",\n");
            }else{
                sb.append(" DEFAULT NULL,\n");
            }
        }else{
            sb.append(" NOT NULL,\n");
            if(column.primaryKey()){
                primaryKey = "PRIMARY KEY ("+field+")";
            }
        }
    }

    if(!StringUtils.isEmpty(primaryKey)){
        sb.append("\t").append(primaryKey);
    }
    sb.append("\n) DEFAULT CHARSET=utf8");

    return sb.toString();
}
```

然后，重新生成一下sql语句。

```
String sql = TableUtils.getCreateTableSQl(User.class);
System.out.println(sql);
```

```
DROP TABLE IF EXISTS t_user;
create table t_user(
        id varchar(100) NOT NULL,
        username VARCHAR(20) DEFAULT NULL,
        password VARCHAR(20) DEFAULT NULL,
        headerPic VARCHAR(60) DEFAULT NULL,
        email VARCHAR(60) DEFAULT NULL,
        sex VARCHAR(2) DEFAULT NULL,
        create_time datetime DEFAULT NULL,
        update_time timestamp,
        is_delete int(1) DEFAULT NULL,
        address VARCHAR(200) DEFAULT NULL,
        telephone VARCHAR(15) DEFAULT NULL,
        PRIMARY KEY (id)
) DEFAULT CHARSET=utf8
```

加分号，回车。

```
mysql> DROP TABLE IF EXISTS t_user;
Query OK, 0 rows affected (0.12 sec)

mysql> create table t_user(
    ->  id varchar(100) NOT NULL,
    ->  username VARCHAR(20) DEFAULT NULL,
    ->  password VARCHAR(20) DEFAULT NULL,
    ->  headerPic VARCHAR(60) DEFAULT NULL,
    ->  email VARCHAR(60) DEFAULT NULL,
    ->  sex VARCHAR(2) DEFAULT NULL,
    ->  create_time datetime DEFAULT NULL,
    ->  update_time timestamp,
    ->  is_delete int(1) DEFAULT NULL,
    ->  address VARCHAR(200) DEFAULT NULL,
    ->  telephone VARCHAR(15) DEFAULT NULL,
    ->  PRIMARY KEY (id)
    -> ) DEFAULT CHARSET=utf8
    -> ;
Query OK, 0 rows affected (0.16 sec)
```

查看表结构

```
mysql> show columns from t_user;
```

```
| id          | varchar(100) | NO   | PRI | NULL              |
          |
```

成功改变了。

好的，回到DataBaseUtils，执行新增操作

```
String id = UUID.randomUUID() + "";
String createTime = new SimpleDateFormat("yyyy-MM-dd").format(new Date());
update("INSERT INTO t_user(id,username,password,sex,create_time,is_delete,address,telephone) "
        + "VALUES (?, ?, ?, ?, ?, ?, ?, ?)", id,"张三",123456,0,createTime,0,"保密","保密");
```

让我们查询一下现在表里有几条数据。

```
mysql> select count(*) from t_user;
+----------+
| count(*) |
+----------+
|        1 |
+----------+
1 row in set (0.00 sec)
```

嗯，应该是成功了呢。

时间关系，本章先介绍到这里。国庆放假正好有时间，过几天还会有一章。

源码的话，从下一章开始，还是以网盘的形式分享出来吧。每一章我都会提供一个版本的。

本文结束。

最近被疯涨的房价弄得有点郁闷，唉，这TM才几个月啊，博主待的小城市的房价就已经远远超出承受范围了。

说多了全是泪。。。

踏入社会后，才知道赚钱的不易。说实话，博主根本没有想到普通小城市的房价也能长成这样。

我当初就是顾虑到房价，也不想每天挤数个小时的地铁去上班，这才没有去北上广。

原本计划在自己的家乡安安稳稳的找份工作，过平静的日子。不料今年的房价猛地一窜，把所有的计划都打乱了。

哎，随遇而安吧。