【手把手】JavaWeb 入门级项目实战 -- 文章发布系统 （第六节）

继续上一节的内容，首先我们将配置方法写在static块里面吧，不然每次调用DataBaseUtils都需要去配置一下，这样比较麻烦。

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

08 查询方法：queryForList 实现

queryForList方法是在实际开发中比较常用的一个方法，它的意思就是说，如果你从数据库里查询出来10条数据，那么用一个List包裹起来，每一条数据就是一个Map。

上代码

```
/**
 * 查询出数据，并且list返回
 * @param sql
 * @param objects
 * @return
 * @throws SQLException
 */
public static List<Map<String,Object>> queryForList(String sql,Object...objects){
    List<Map<String,Object>> result = new ArrayList<Map<String,Object>>();
    Connection connection = getConnection();
    PreparedStatement statement = null;
    ResultSet rs = null;
    try {
        statement = connection.prepareStatement(sql);
        for (int i = 0; i < objects.length; i++) {
            statement.setObject(i+1, objects[i]);
        }

        rs = statement.executeQuery();
        while(rs.next()){
            ResultSetMetaData resultSetMetaData = rs.getMetaData();
            int count = resultSetMetaData.getColumnCount(); //获取列数
            Map<String,Object> map = new HashMap<String, Object>();
            for (int i = 0; i < count; i++) {
                map.put(resultSetMetaData.getColumnName(i+1), rs.getObject(resultSetMetaData.getColumnName(i+1)));
            }
            result.add(map);
        };
    } catch (SQLException e) {
        e.printStackTrace();
    }finally{
        closeConnection(connection, statement, rs);
    }

    return result;
}
```

测试：

我刚才又添加了几条数据到数据库了，现在我们希望将查出来的结果集放到一个list中。

```
DataBaseUtils.config("jdbc.properties");
List list = DataBaseUtils.queryForList("select * from t_user");
System.out.println(list);
```

结果：

```
[{id=24cb5136-39cf-4109-a28b-d412efd6ad2d, sex=0, headerPic=null, username=赵六, update_time=2016-10-05 13:01:33.0, address=保密, email=null, create_time=2016-10-05 00:00:00.0, is_
```

成功了。

09 查询方法：queryForMap 实现

这个方法的作用是查询出一条数据，因为一个HashMap实际上就对应数据库表的一行数据。刚才，我们已经实现了queryForList方法，所以，现在只需要稍微调用一下，queryForMap 方法就出来了。

思路就是我先查出一个list，然后取第一条就OK了。

代码：

```
/**
 * 查询出数据，并且map返回
 * @param sql
 * @param objects
 * @return
 * @throws SQLException
 */
public static Map<String,Object> queryForMap(String sql,Object...objects) throws SQLException{
    Map<String,Object> result = new HashMap<String,Object>();
    List<Map<String,Object>> list = queryForList(sql, objects);
    if(list.size() != 1){
        return null;
    }
    result = list.get(0);
    return result;
}
```

测试，查询所有的数据：

```
DataBaseUtils.config("jdbc.properties");
Map map = DataBaseUtils.queryForMap("select * from t_user");
System.out.println(map);
```

运行结果为 null

因为查询出来的不止一条数据，所以返回了null。

现在我们就查一条数据

```
DataBaseUtils.config("jdbc.properties");
Map map = DataBaseUtils.queryForMap("select * from t_user where username = ?","王五");
System.out.println(map);
```

运行结果：

```
{id=3fe31336-a792-488f-9445-f3ebd30f9de3, sex=0, headerPic=null, username=王五, update_time=2016-10-05 13:01:33.0, address=保密, email=null, create_time=2016-10-05 00:00:00.0, is_d
```

这样就对了。

10 查询方法：queryForBean 实现

终于到queryForBean了，这个方法的意思就是说，将查询出来的一条数据（注意，只能是一条数据）转换成JavaBean,也就是一个Java对象。

比如，我从t_user表中查一条数据出来，然后它就能给我返回一个User对象。很显然，这个方法肯定需要用到Java的反射API和泛型。

因为我们之前已经写好了queryForMap，所以在这个方法中，可以直接调用那个方法，这样省去了很多的工作量。

大体的思路就是：

1.拿到map。

2.新建一个JavaBean，因为事先不知道JavaBean的类型，所以要传进来一个class属性，然后方法的返回值需要用到泛型，不然没法new。

3.遍历map中所有的key，想办法获取对应的set方法。（注意，你的JavaBean必须要符合Bean规范，否则会有问题的）

4.通过反射来调用set方法。

5. 返回已经注入好的JavaBean。

这个方法呢，我大概写了2个小时，因为实在没有写过类似的代码，只能自己在那一点点摸索。

代码的话呢，肯定还有不足的地方，我大概测试了一下，不论怎么样，差不多可以用。

OK，上代码：

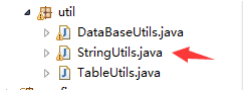
```
/**
 * 查询出数据，并且返回一个JavaBean
 * @param sql
 * @param clazz
 * @param objects
 * @return
 * @throws NoSuchFieldException
 * @throws SecurityException
 */
public static <T>T queryForBean(String sql,Class clazz,Object...objects){
    T obj = null;
    Map<String,Object> map = null;
    Field field = null;
    try {
        obj = (T) clazz.newInstance();   //创建一个新的Bean实例
        map = queryForMap(sql, objects); //先将结果集放在一个Map中
    } catch (InstantiationException | IllegalAccessException e) {
        e.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    if(map == null){
        return null;
    }
    //遍历Map
    for (String columnName : map.keySet()) {
        Method method = null;
        String propertyName = StringUtils.columnToProperty(columnName); //属性名称

        try {
            field = clazz.getDeclaredField(propertyName);
        } catch (NoSuchFieldException e1) {
            e1.printStackTrace();
        } catch (SecurityException e1) {
            e1.printStackTrace();
        }
        //获取JavaBean中的字段
        String fieldType = field.toString().split(" ")[1]; //获取该字段的类型
        //System.out.println(fieldType);
        Object value = map.get(columnName);
        if(value == null){
            continue;
        }
        /**获取set方法的名字* */
        String setMethodName = "set" + StringUtils.upperCaseFirstCharacter(propertyName);
        //System.out.println(setMethodName);
        try {
            /**获取值的类型* */
            String valueType = value.getClass().getName();

            /**查看类型是否匹配* */
            if(!fieldType.equalsIgnoreCase(valueType)){
                //System.out.println("类型不匹配");
                if(fieldType.equalsIgnoreCase("java.lang.Integer")){
                    value = Integer.parseInt(String.valueOf(value));
                }else if(fieldType.equalsIgnoreCase("java.lang.String")){
                    value = String.valueOf(value);
                }else if(fieldType.equalsIgnoreCase("java.util.Date")){
                    valueType = "java.util.Date";
                    //将value转换成java.util.Date
                    String dateStr = String.valueOf(value);
                    Timestamp ts = Timestamp.valueOf(dateStr);
                    Date date = new Date(ts.getTime());
                    value = date;
                }
            }

            /**获取set方法* */
            //System.out.println(valueType);
            method = clazz.getDeclaredMethod(setMethodName,Class.forName(fieldType));
            /**执行set方法* */
            method.invoke(obj, value);
        }catch(Exception e){
            e.printStackTrace();
            /**如果报错，基本上是因为类型不匹配* */
        }
    }
    //System.out.println(obj);
    return obj;
}
```

这段代码涉及到两个方法，我这边也分享出来吧，我已经把它们放在StringUtils里面了。如果你自己也打算写类似的小框架的话呢，这两个方法很可能会被用到。




```
/**
 * 把数据库字段名改为驼峰方式
 * @param column
 * @return
 */
public static String columnToProperty(String column) {
    /**如果字段名为空，就返回空字符串* */
    if(isEmpty(column)) return "";
    /**获取字段的长度，一般来说字段长度不可能有几百个字节的，所以用Byte就行了* */
    Byte length = (byte) column.length();

    StringBuilder sb = new StringBuilder(length);
    int i = 0;
    /**遍历字段的每一个字符* */
    for (int j = 0; j < length; j++) {
         /**匹配到第一个_* */
        if (column.charAt(j) == '_') {
            /**如果后面还有_,也就是连续的_,那么j就需要自增一个单位，直到后面不是_为止* */
            while (column.charAt(j + 1) == '_') {
                j += 1;
            }
            sb.append(("" + column.charAt(++j)).toUpperCase());
```

```
        } else {
             /**如果循环到的字符不是_,那么就保存下来* */
                sb.append(column.charAt(j));
            }
        }

     return sb.toString();
 }

/**
 * 将一个字符串的首字母改成大写
 * @param str
 * @return
 */
public static String upperCaseFirstCharacter(String str){
    StringBuilder sb = new StringBuilder();
    char[] arr = str.toCharArray();
    for (int i = 0; i < arr.length; i++) {
        if(i==0) sb.append((arr[i] + "").toUpperCase());
        else sb.append((arr[i]+""));
    }
    return sb.toString();
}
```

测试

```
User user = DataBaseUtils.queryForBean("select * from t_user  limit 1", User.class);
System.out.println(user);
```

结果（我已经给User类改写了toString方法）：

User [id=24cb5136-39cf-4109-a28b-d412efd6ad2d, username=赵六, password=123456, headerPic=null, email=null, sex=0, createTime=2016-10-05 00:00:00.0, updateTime=2016-10-05 13:01:33.0, isDelete=0, address=保密, telephone=保密]

那么，好的，我们的DataBaseUtils暂时告一段落了。虽然还不完善，可是已经有了一个雏形了。所以，写代码不要怕，只要有时间，你肯定能写出来，最多就是不完善罢了。你也可以利用网络上的资源，有些东西慢慢琢磨，基本上都是可以弄出来的。很多初学者最大的问题就是不敢去尝试，仅此而已。

接下来，我们继续写业务。

11 从controller层到service层

为了方便起见，我们就写到Service就OK了，省去dao层。反正我经历的几个公司都是这么做的，他们似乎都有意弱化了dao层，直接写到service层就结束了。

我们继续编写controller的代码，因为要调用service的方法，所以，我们需要把必要的包导入进来。

```
<%@ page language="java" import="java.util.*,service.LoginService,util.StringUtils,bean.*" pageEncoding="UTF-8"%>
```

controller代码

```
<%
    //获取客户端传递过来参数
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    //System.out.println(username);
    //System.out.println(password);
    //如果用户名和密码不为空
    if(StringUtils.isEmpty(username) || StringUtils.isEmpty(password)){
        out.print("-1");//错误码-1 :   用户名和密码不能为空!
    }else{
        //初始化LoginService
        LoginService loginService = new LoginService();
        //接下来判断用户名是否存在
        User user = loginService.getUser(username);
        if(user == null){
            out.print("-2");//错误码-2 :   用户名不存在!
        }else
            //如果用户名存在，那么验证用户名和密码是否匹配
            if(!username.equals(user.getUsername()) || !password.equals(user.getPassword())){
                out.print("-3");//错误码-3 :   用户名或密码错误!
            }else{
                //如果能到这一步，就说明用户的确存在，而且账号密码也正确。那么就把user放在session中
                out.print("1");
                session.setAttribute("user", user);
                session.setAttribute("username", user.getUsername());
            }
    }

%>
```

流程是这样的，首先判断用户名和密码是否为空，如果为空，就直接返回一个错误码-1，接下来依次判断用户名是否存在，以及用户名密码是否都正确。

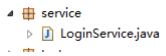
只要有一个不符合，就直接返回对应的错误码。out对象是JSP九大隐式对象的一员，可以将某个值返回给前台。

然后，如果账号密码都正确，那么就返回一个1，表示登录成功，同时，把user对象和用户名放到session中。

session的话，就是浏览器作用域，只要你浏览器开着，里面的值就存在着。浏览器一关，存在session中的东西就没了。当然，你也可以自己设定session失效的时间。

你可以把session想象成一个篮子。篮子嘛，肯定是用来放东西的，就这么简单。

service类：



代码

```
package service;

import bean.User;
import util.DataBaseUtils;

/**
 * 用户登录的服务类
 * @author lenovo
 *
 */
public class LoginService {

    public User getUser(String username){
        String sql = "select * from t_user where username = ?";
        User user = DataBaseUtils.queryForBean(sql, User.class, username);
        if(user == null){
            return null;
        }
        //System.out.println(user);
        return user;
    }

}
```

现在里面只有一个getUser方法，就是传入一个username，然后返回一个User对象。

我们刚才编写的DataBaseUtils终于可以大显身手啦！ ^_^

让我们回到login.jsp，现在可以把登录事件完善一下了。


```
        <div class='mt50 ml32'>
            <button class="login_btn" onclick="login()">登陆</button>
        </div>
    </div>
```

代码

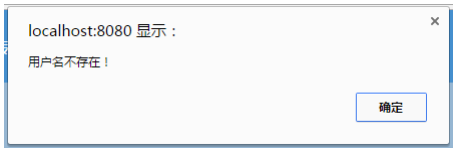
```
<script>

    function login(){
        var username = $('#username').val();
        var password = $('#password').val();
        $.ajax({
            type:"post",//请求方式
            url:"${basePath}/controller/loginController.jsp",//请求地址
            data:{"username":username,"password":password},//传递给controller的json数据
            error:function(){//如果出错了，将事件重新绑定
                alert("登陆出错！");
            },
            success:function(data){ //返回成功执行回调函数。
                if(data == -1){
                    alert('用户名和密码不能为空！');
                }else if(data == -2){
                    alert('用户名不存在！');
                }else if(data == -3){
                    alert('用户名或密码错误！');
                }else{
                    //登录成功后返回首页
                    window.location.href = "${basePath}";
                }
            }
        });


    }
</script>
```

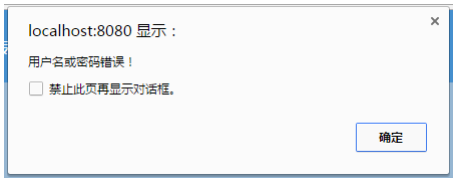
12 登录功能完成

点击登录。



把用户名改为张三

点击登录。



密码改为123456，

点击登录。

成功了，跳转到了首页！

最后说两句，登录功能终于做完了，想想还有点小激动呢。^_^

这也是我第一次封装一个自己的DataBaseUtils，虽然不完善，可是毕竟能用。其实，这已经有那么一点点框架的味道了。我的观点就是这样，不管怎么样，都要去努力尝试，就算自己的代码写得再烂，也要努力去写。

知识点学了就是要拿来用的，不然学它干嘛呢？

培训的时候，泛型，反射这些知识点也会讲到，可是如果我们只是记住了教程里面的那几个例子，多半还是语法级别的内容，是没有太大的意义的。必须要自己反思，我能用这些知识点来做些什么？

俗话说得好，一屋不扫，何以扫天下？尤其是对于初学者，不要总想着什么框架，总想着去学什么新技术。关键还是要自己总结和反思，还有，必须，一定，毫不犹豫地，要自己学着做项目。什么项目都行，但一定要自己去尝试。

很多初学者，都有这么一个思维，就是我一定要做一个OA，我一定要做一个ERP。似乎在他们眼中，只有这种的才算是项目。

结果就是，他们一直在学，永远也学不完，就是不敢去找工作。

其实在我看来，OA，ERP这种，一个人完成是不太现实的，这应该是一个团队的事情。对于初学者而言，你完全可以自己做一个小型的项目，比如日记本系统，学生管理系统。

通过做项目，去把知识点一个个地累积起来。

所谓的OA，不也是这么回事吗？

一个业务系统，里面至少有几十个，甚至几百个菜单。总体来看的确很庞大，但是你只要一细分，你会发现大部分菜单都差不多，无非业务变一下罢了。说穿了还不就是MVC。

所谓的业务系统就是这样的，就是不断地加功能，加菜单。

一个大项目和小项目，单个菜单的难度其实是差不太多的，技术点无非就那么几个。复杂的是业务，所以你有的时候会在面试的时候听到：我们不关心技术，只关心业务的言论。

所以，没必要给自己设限，总是觉得，哎呀，我一定做一个OA系统，一定要学会工作流，甚至我一定要去学大数据。

不是这样的，我感觉这不是初学者该考虑的问题。要知道，那种大型的业务系统，动辄几百个菜单，本来就不是靠一个人完成的，多数情况是一个团队开发一个月，或者几个月的事情。

还有一点就是，不要对框架产生畏惧心理，我很明白，很多人初学Java都会听到SSH的大名，有些人非常迫切地想要一份工作，于是买了很多SSH的书来看。

不看不要紧，一看很容易就懵逼了，因为根本看不懂。

其实，不是你看不懂，而是你做的东西太少。

很多人，明明Java基础挺好的，就是总喜欢把精力放在研究框架的书籍上面，不肯去自己动手写项目。

框架只是一个工具，仅此而已。

要真正理解框架，还是要通过做项目的。

无论这个项目有多小，都无所谓，关键是，你要敢于去尝试。而不是说，只有OA，ERP才是项目，然后，你觉得哎呀，好难啊，我做不了了。我还是继续去买书来看吧。接着就是，“你好，我们是某某培训公司的，我们在网上看到你的简历。。。”

你看再多的视频，买再多的书，都不如自己做个项目来得快。

就是这样的，个人感慨。

本文结束。