后台编码实现

第一章 连接池配置

因为项目中选用 c3p0 做连接池，所以首先配置连接池的一些信息，主要就是配置好数据库的一些基本信息。配置信息如图：

```
<?xml version="1.0" encoding="UTF-8"?>
<c3p0-config>
    <default-config>
        <property name="driverClass">com.mysql.jdbc.Driver</property>
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/ssms</property>
        <property name="user">root</property>
        <property name="password">root</property>
    </default-config>
</c3p0-config>
```

上图配置信息中：

driverClass：代表所用数据库驱动的文件位置。

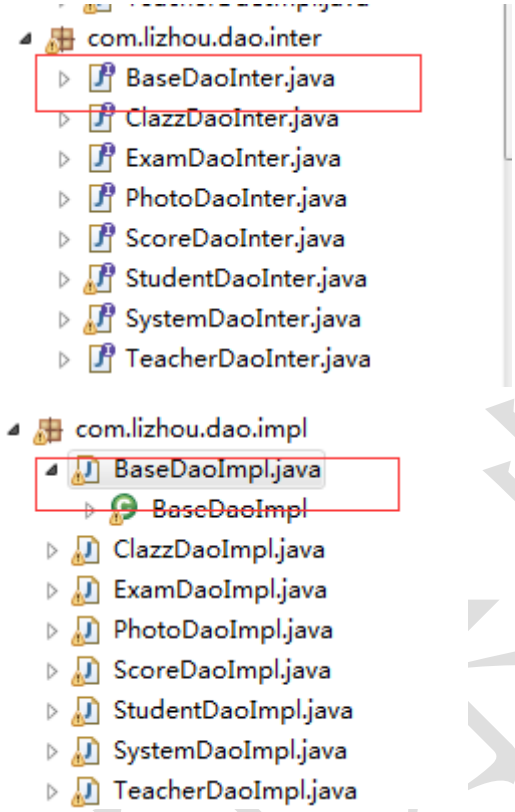
jdbcUrl：代表数据库的连接地址

user:代表数据库中的用户的用户名

user:代表登录用户的登录密码

第二章 数据库操作基本类实现

数据库的操作选用的是 org.apache.commons.dbutils 类中的方法。在编写的 BaseDaoImpl 类中分别实现了数据的基本增删改查操作。具体实现方式可去看源码：



第三章 登录功能

从前台登录页面可以找到点击登录按钮后台调用哪个方法：

```
//登录
$("#submitBtn").click(function(){
    if($("#radio-2").attr("checked") && "${syste
        $.messager.alert("消息提醒", "学生暂不能登录系统!
        return;
    }
    if($("#radio-3").attr("checked") && "${syste
        $.messager.alert("消息提醒", "教师暂不能登录系统!
        return;
    }

    var data = $("#form").serialize();
    $.ajax({
        type: "post",
        url: "LoginServlet?method=Login",
        data: data,
        dataType: "text", //返回数据类型
```

找到 LoginServlet 文件中的对应方法：

```
protected void doPost(HttpServletRequest request, HttpServl
    //获取请求的方法
    String method = request.getParameter("method");

    if("Login".equals(method)){ //验证登录
        login(request, response);
    }
}
```

```
    /**
     * 验证用户登录
     * @param request
     * @param response
     * @throws IOException
     */
    private void login(HttpServletRequest request, HttpServletRespo
        //获取用户输入的账户
        String account = request.getParameter("account");
        //获取用户输入的密码
        String password = request.getParameter("password");
        //获取用户输入的验证码
        String vcode = request.getParameter("vcode");
        //获取登录类型
        int type = Integer.parseInt(request.getParameter("type"));
```

登录方法的实现思路：

① Servlet 类中的实现方式：

首先获得页面的影用户输入信息，获得方式是：

```
//获取用户输入的账户
String account = request.getParameter("account");
//获取用户输入的密码
String password = request.getParameter("password");
//获取用户输入的验证码
String vcode = request.getParameter("vcode");
//获取登录类型
int type = Integer.parseInt(request.getParameter("type"));
```

然后查询用户信息登录是否正确，这边需要去数据库中和既存的用户数据进行比对。所以这里 servlet 层会调用 service 层的方法。

```
//查询用户是否存在
User loginUser = service.getAdmin(user);
```

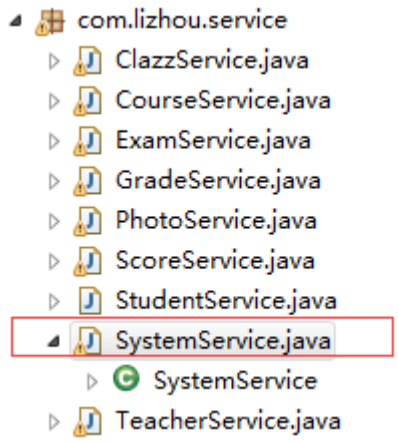
最后通过 service 返回数据判断用户登录角色，最后将数据返回。

```
//返回登录信息
response.getWriter().write(msg);
```

前台根据接收到的信息判断应该是跳转登录成功相对应的页面还是切换验证码之类的。

```
$.ajax({
    type: "post",
    url: "LoginServlet?method=Login",
    data: data,
    dataType: "text", //返回数据类型
    success: function(msg){
        if("vcodeError" == msg){
            $.messager.alert("消息提醒", "验证码错误!", "warning");
            $("#vcodeImg").click();//切换验证码
            $("input[name='vcode']").val("");//清空验证码输入框
        } else if("loginError" == msg){
            $.messager.alert("消息提醒", "用户名或密码错误!", "warning");
            $("#vcodeImg").click();//切换验证码
            $("input[name='vcode']").val("");//清空验证码输入框
        } else if("admin" == msg){
            window.location.href = "SystemServlet?method=toAdminView";
        } else if("student" == msg){
            window.location.href = "SystemServlet?method=toStudentView";
        } else if("teacher" == msg){
            window.location.href = "SystemServlet?method=toTeacherView";
        }
    }
```

② service 类中的实现方式：

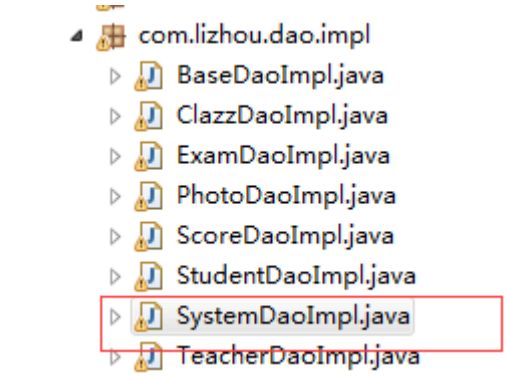



```
/**
 * 登录验证
 * @param user
 * @return
 */
public User getAdmin(User user) {
    User searchUser = (User) dao.getObject(User.class,
            "SELECT * FROM user WHERE account=? AND password=? AND type=?",
            new Object[]{user.getAccount(), user.getPassword(), user.getType()});

    return searchUser;
}
```

Service 层主要用来进行业务逻辑判断的，因为登录功能中没有别的逻辑判断，只需要得到 dao 层对数据库数据的操作结果即可，所以登录的 service 类中只调用了 dao 层中获取对象（getObject）的方法。

③ dao 类中的实现方式




```java
/**
 * 系统数据层
 * @author bojiangzhou
 *
 */
public class SystemDaoImpl extends BaseDaoImpl implements SystemDaoInter {

}
```

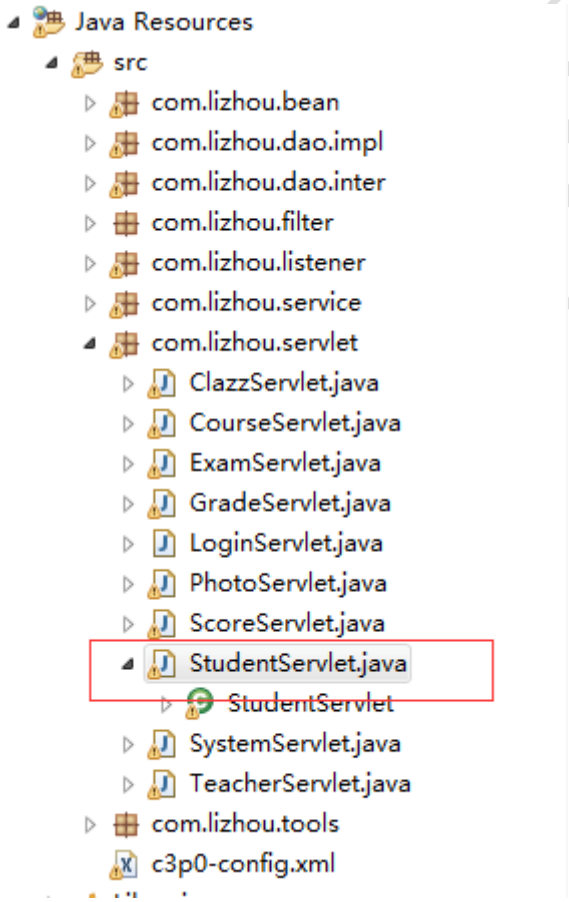
登录对应的 dao 类只继承了 BaseDaoImpl。

第三章 学生数据查询

学生列表数据的获得由前台可知调用的后台方法如图：

```
$(function() {
    //datagrid初始化
    $('#dataList').datagrid({
        title:'学生列表',
        iconCls:'icon-more',//图标
        border: true,
        collapsible:false,//是否可折叠的
        fit: true,//自动大小
        method: "post",
        url:"StudentServlet?method=StudentList&t="+new Date().getTime()
```

① Servlet 类的实现：



```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws S
    //获取请求的方法
    String method = request.getParameter("method");
    //请求分发
    if("StudentList".equalsIgnoreCase(method)){ //获取所有学生数据
        studentList(request, response);
    } else if("AddStudent".equalsIgnoreCase(method)){ //添加学生
        addStudent(request, response);
    } else if("DeleteStudent".equalsIgnoreCase(method)){ //删除学生
        deleteStudent(request, response);
    } else if("EditStudent".equalsIgnoreCase(method)){ //修改学生信息
        editStudent(request, response);
    } else if("StudentClazzList".equalsIgnoreCase(method)){ //获取当前学生班级的所有学生
        studentClazzList(request, response);
    }

}

private void studentList(HttpServletRequest request, HttpServletResponse respon
    //年级ID
    String gradeid = request.getParameter("gradeid");
    //班级ID
    String clazzid = request.getParameter("clazzid");
    //获取分页参数
    int page = Integer.parseInt(request.getParameter("page"));
    int rows = Integer.parseInt(request.getParameter("rows"));

    //封装参数
    Student student = new Student();

    if(!StringTool.isEmpty(gradeid)){
        student.setGradeid(Integer.parseInt(gradeid));
    }
    if(!StringTool.isEmpty(clazzid)){
        student.setClazzid(Integer.parseInt(clazzid));
    }
```

首先获取数据查询条件参数：

```
//年级ID
String gradeid = request.getParameter("gradeid");
//班级ID
String clazzid = request.getParameter("clazzid");
//获取分页参数
int page = Integer.parseInt(request.getParameter("page"));
int rows = Integer.parseInt(request.getParameter("rows"));
```

然后调用 dao 层的获取学生数据的方法。因为这个方法参数传的是一个学生对象，所以在调用此方法之前需要将从页面获得的数据封装成学生对象。

```
//封装参数
Student student = new Student();

if(!StringTool.isEmpty(gradeid)){
    student.setGradeid(Integer.parseInt(gradeid));
}
if(!StringTool.isEmpty(clazzid)){
    student.setClazzid(Integer.parseInt(clazzid));
}

//获取数据
String result = service.getStudentList(student, new Page(page, rows));
```

最后返回页面查询结果。

```
//返回数据
response.getWriter().write(result);
```

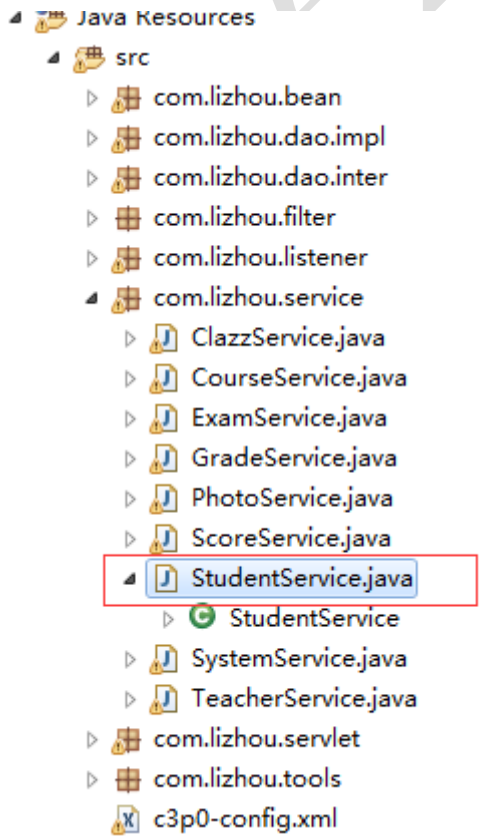
然后将 json 格式的数据显示到前台列表中（easyui 的把表格插件默认接收 json 格式的数据）。

```
    //datagrid初始化
    $('#dataList').datagrid({
        title:'学生列表',
        iconCls:'icon-more',//图标
        border: true,
        collapsible:false,//是否可折叠的
        fit: true,//自动大小
        method: "post",
        url:"StudentServlet?method=StudentList&t="+new Date().getTime(),
        idField:'id',
        singleSelect:false,//是否单选
        pagination:true,//分页控件
        rownumbers:true,//行号
        sortName:'id',
        sortOrder:'DESC',
        remoteSort: false,
        columns: [[
            {field:'chk',checkbox: true,width:50},
            {field:'id',title:'ID',width:50, sortable: true},
            {field:'number',title:'学号',width:200, sortable: true},
            {field:'name',title:'姓名',width:200},
            {field:'sex',title:'性别',width:100},
```

```
        {field:'phone',title:'电话',width:150},
        {field:'qq',title:'QQ',width:150},
        {field:'clazz',title:'班级',width:150,
            formatter: function(value,row,index){
                if (row.clazz){
                    return row.clazz.name;
                } else {
                    return value;
                }
            }
        },
        {field:'grade',title:'年级',width:150,
            formatter: function(value,row,index){
                if (row.grade){
                    return row.grade.name;
                } else {
                    return value;
                }
            }
        },
    ]],
```

这里注意 field 的名字要和返回的 json 格式数据的 key 的名字相同。否则数据不会成功显示。

② Service 类实现：



```
/**
 * 分页获取学生
 * @param student 学生信息
 * @param page 分页
 * @return
 */
public String getStudentList(Student student, Page page){
    //sql语句
    StringBuffer sb = new StringBuffer("SELECT * FROM student ");
    //参数
    List<Object> param = new LinkedList<>();
```

首先根据查询条件来拼接不同的 sql。

```
//sql语句
StringBuffer sb = new StringBuffer("SELECT * FROM student ");
//参数
List<Object> param = new LinkedList<>();
//判断条件
if(student != null){
    if(student.getGrade() != null){//条件：年级
        int gradeid = student.getGrade().getId();
        param.add(gradeid);
        sb.append("AND gradeid=? ");
    }
    if(student.getClazz() != null){
        int clazzid = student.getClazz().getId();
        param.add(clazzid);
        sb.append("AND clazzid=? ");
    }
}
//添加排序
sb.append("ORDER BY id DESC ");

 //分页
 if(page != null){
     param.add(page.getStart());
     param.add(page.getSize());
     sb.append("limit ?,?");
 }
 String sql = sb.toString().replaceFirst("AND", "WHERE");
```

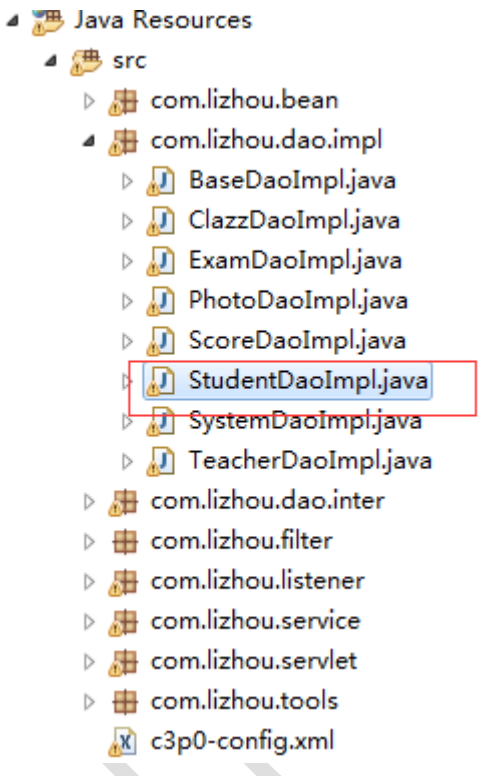
然后去调用 dao 层的方法去查询数据。

```
//获取数据
List<Student> list = dao.getStudentList(sql, param);
```

最后将对象数据转换成 json 格式返回到 servlet 层。

```
//获取总记录数
long total = getCount(student);
//定义Map
Map<String, Object> jsonMap = new HashMap<String, Object>();
//total键 存放总记录数，必须的
jsonMap.put("total", total);
//rows键 存放每页记录 list
jsonMap.put("rows", list);
//格式化Map,以json格式返回数据
String result = JSONObject.fromObject(jsonMap).toString();
//返回
return result;
```

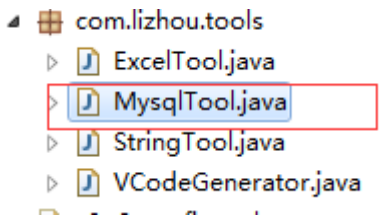
③ Dao 层实现：




```
public List<Student> getStudentList(String sql, List<Object> param) {
    //数据集合
    List<Student> list = new LinkedList<>();
    try {
        //获取数据库连接
```

对数据库进行操作使用的是最基本的 jdbc 连接。首先获得数据连接（这里直接从连接池中取连接）。这里在 tools 文件夹中放了数据库连接池的基本相关操作工具类 MysqlTool 类。

```
//获取数据库连接
Connection conn = MysqlTool.getConnection();
```




```
public static Connection getConnection(){
    Connection conn = tl.get();
    try {
        if(conn == null){
            conn = dataSource.getConnection();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    tl.set(conn);
    return conn;
}
```

然后执行 sql。

```
//预编译
PreparedStatement ps = conn.prepareStatement(sql);
//设置参数
if(param != null && param.size() > 0){
    for(int i = 0;i < param.size();i++){
        ps.setObject(i+1, param.get(i));
    }
}
//执行sql语句
ResultSet rs = ps.executeQuery();
```

再然后将查询到的结果放到 list 集合中。其中会根据每一个学生记录的主键 id 查询学生所在的班级和年级。调用的是 BaseDao 中的查询方法。

```
//获取元数据
ResultSetMetaData meta = rs.getMetaData();
//遍历结果集
while(rs.next()){
    //创建对象
    Student stu = new Student();
    //遍历每个字段
    for(int i=1;i <= meta.getColumnCount();i++){
        String field = meta.getColumnName(i);
        BeanUtils.setProperty(stu, field, rs.getObject(field));
    }
    //查询班级
    Clazz clazz = (Clazz) getObject(Clazz.class, "SELECT * FROM clazz WHERE id=?", new Object[]{stu.
    //查询年级
    Grade grade = (Grade) getObject(Grade.class, "SELECT * FROM grade WHERE id=?", new Object[]{stu.
    //添加
    stu.setClazz(clazz);
    stu.setGrade(grade);
    //添加到集合
    list.add(stu);
}
```

最后关闭连接。

```
//关闭连接
MysqlTool.closeConnection();
MysqlTool.close(ps);
MysqlTool.close(rs);
```

学生的修改，增加和删除操作实现思路和查询操作是一样的。只不过调用的方法不一样而已。其他操作的学习完全可以参照查询来看。