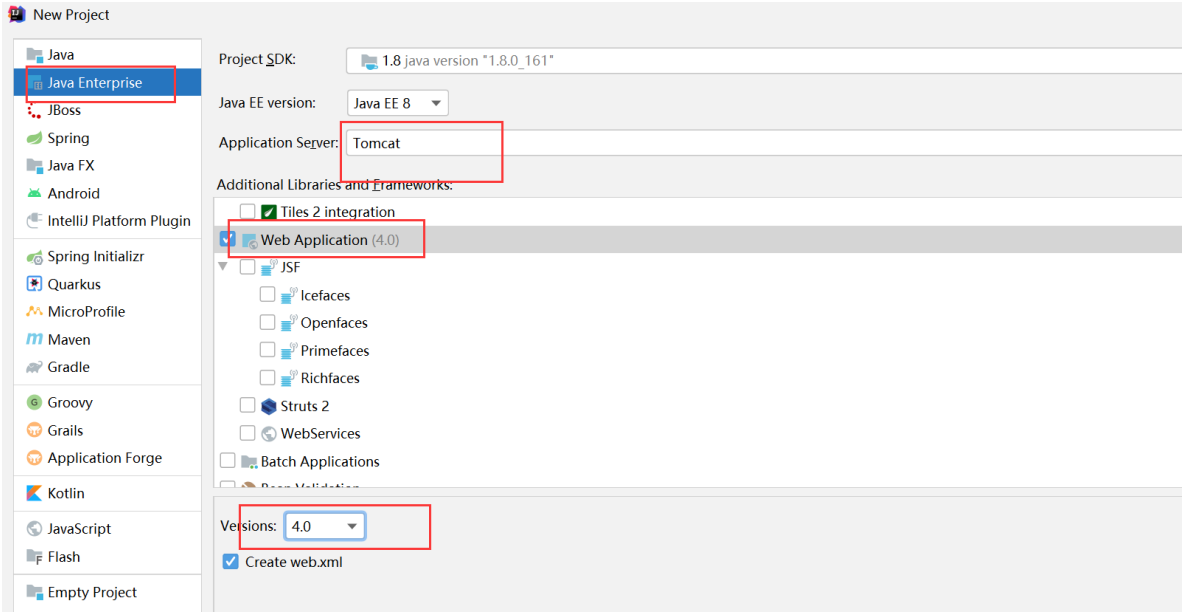
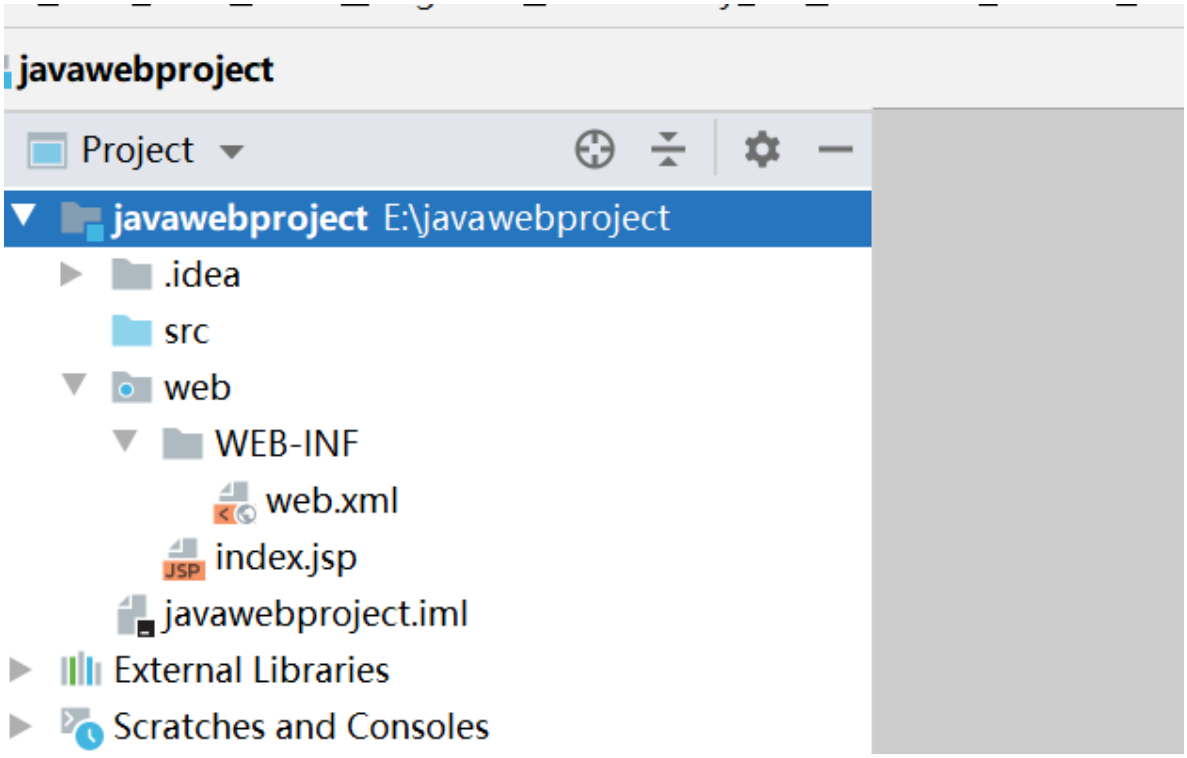
2、项目搭建

1、打开Intellij Idea IDE，然后点击Create New Project**

2、左侧选择Java Enterprise，右侧选择Web Application

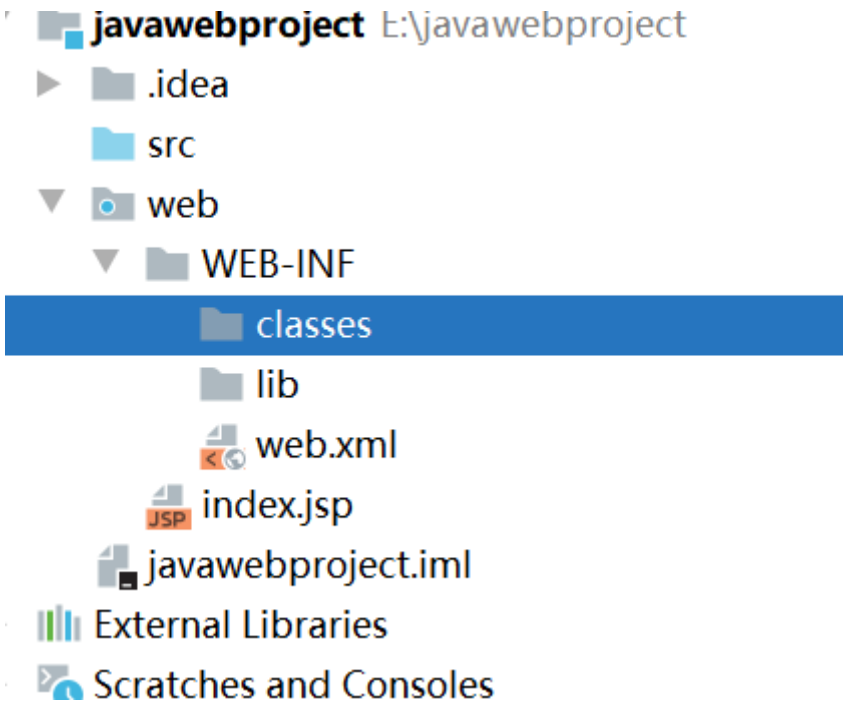


3、输出项目名，路径完成即可



2、 项目配置:

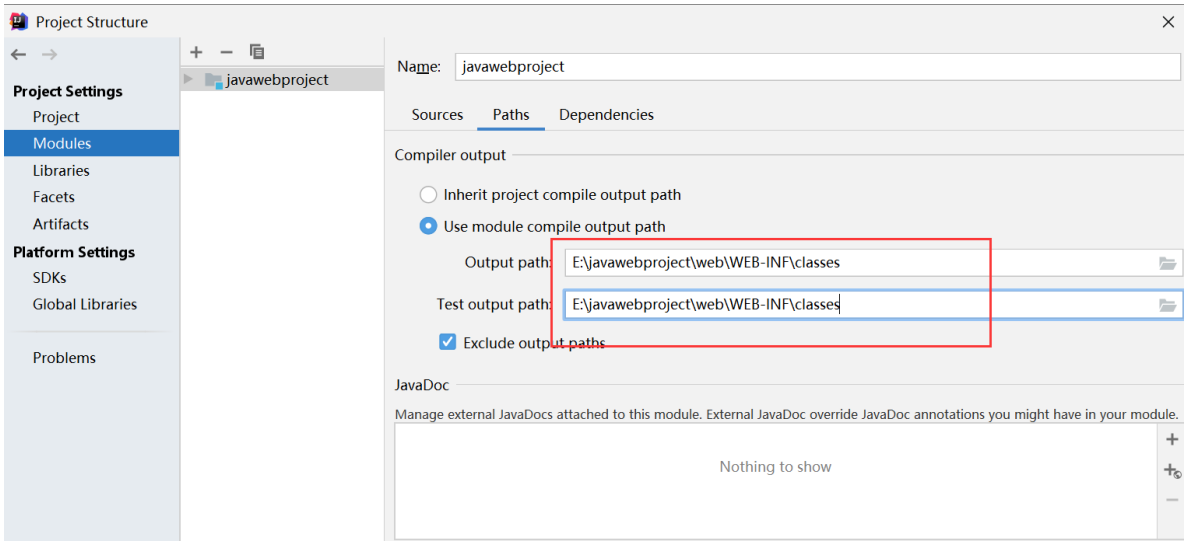
1、在web/WEB-INF下创建两个文件夹classes和lib，classes用来存放编译后输出的classes文件，lib用于存放第三方jar包。



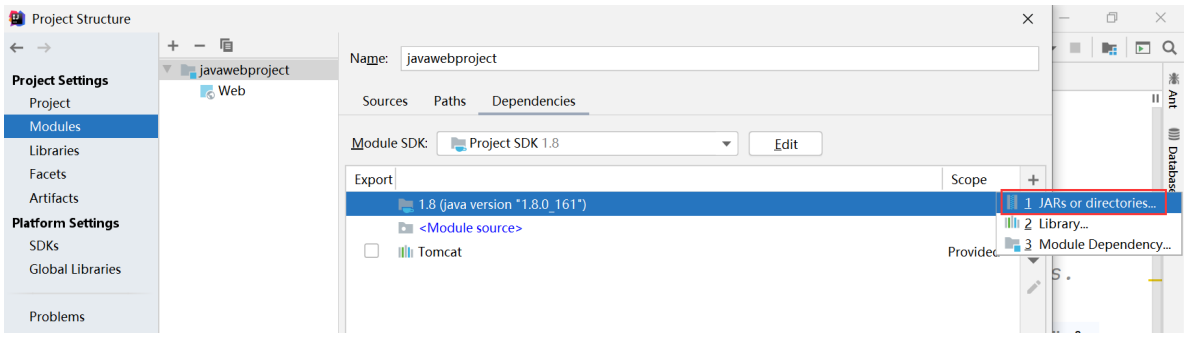
2、配置文件夹路径

File -> Project Structure (快捷键：Ctrl + Shift + Alt + S) -> 选择Module：

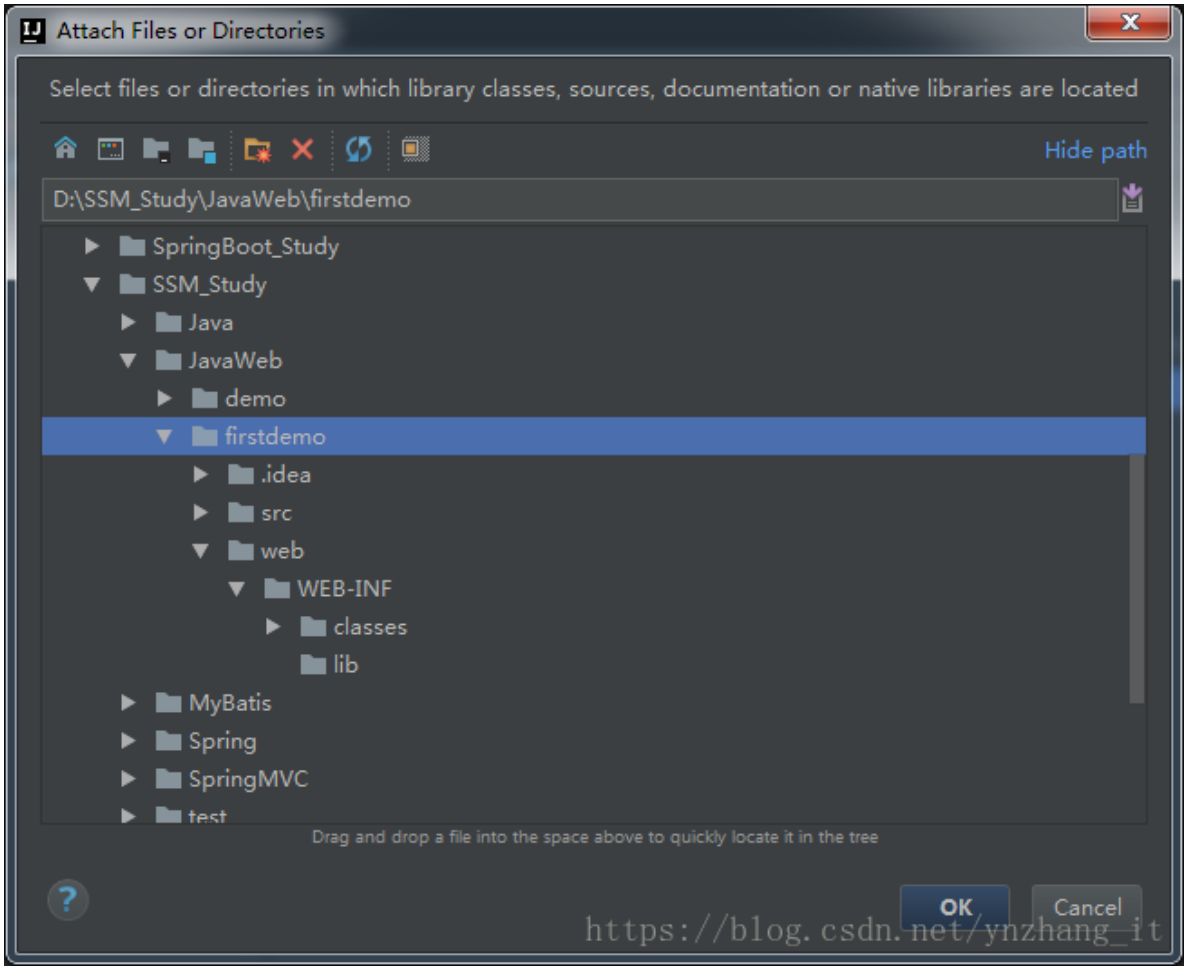
选择 Paths -> 选择"Use module compile output path" -> 将Output path和Test output path都选择刚刚创建的classes文件夹。



3、接着选择Dependencies -> 将Module SDK选择为1.8 -> 点击右边的“+”号 -> 选择1 “Jars or Directories”



-> 选择刚刚创建的lib文件夹



-> 选择“jar directory” -> 接着返回一路OK就行了~~

4、配置Tomcat容器

打开菜单Run -> 选择Edit Configuration

点击“+”号 -> 选择“Tomcat Server” -> 选择“Local”

在"Name"处输入新的服务名，点击"Application server"后面的"Configure..."，弹出Tomcat Server窗口，选择本地安装的Tomcat目录 -> OK

在"Run/Debug Configurations"窗口的"Server"选项板中，取消勾选"After launch"，设置"HTTP port"和"JMX port"（默认值即可），点击 Apply -> OK， 至此Tomcat配置完成。

5、在Tomcat中部署并运行项目

Run -> Edit Configurations，进入"Run/Debug Configurations"窗口 -> 选择刚刚建立的Tomcat容器 -> 选择Deployment -> 点击右边的“+”号 -> 选择Artifact

->选择web项目 -> Application context可以填“/firstdemo”(其实也可以不填的~~) -> OK

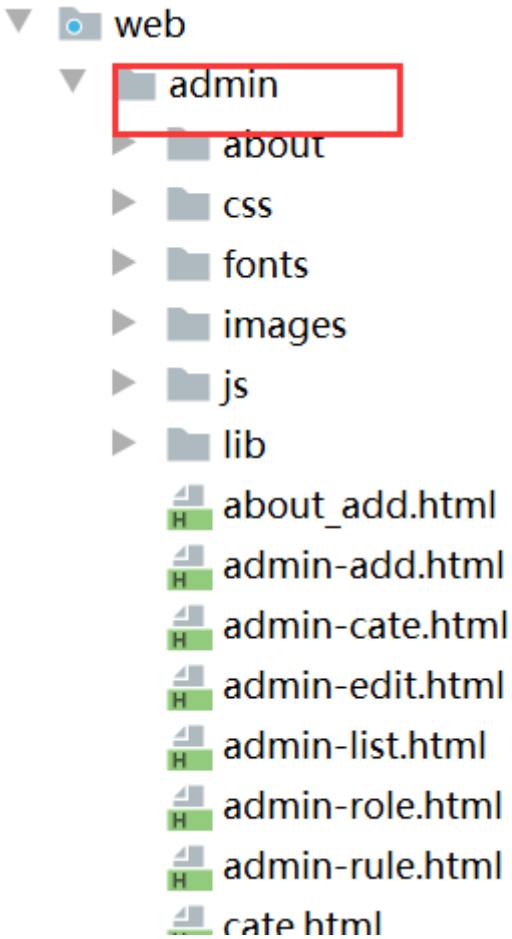
编辑index.jsp文件

运行Tomcat,在浏览器中查看运行结果

3、静态页面引入

3.1 引入后端静态页面

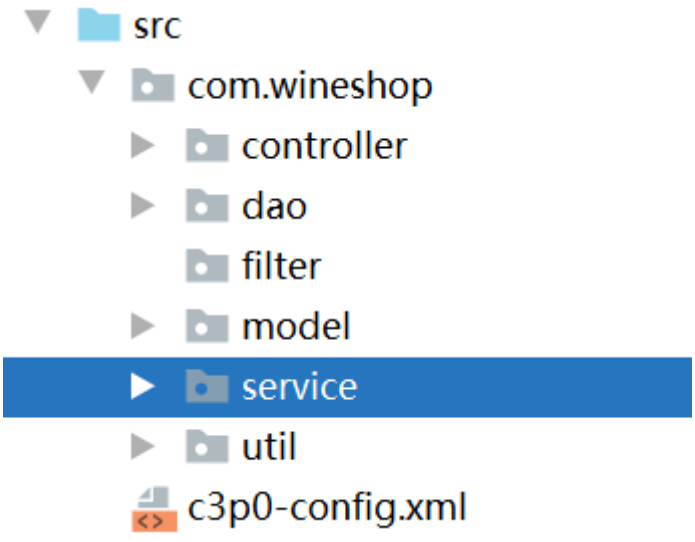
页面内容来自x-admin下载内容或者提供的后端页面内容



3.2 引入前端静态页面

参考后端引入即可

4、后端项目层级关系



5、基础工具类创建

前置工作先导入相关的依赖包，最好把提供的依赖包都导入

5.1 c3p0-config.xml 文件创建

```
<?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/wineshop</property>
        <property name="user">root</property>
        <property name="password">root</property>
        <property name="initialPoolSize">10</property>
        <property name="maxPoolSize">30</property>
        <property name="minPoolSize">10</property>
    </default-config>

</c3p0-config>
```

5.2 c3p0 数据库连接工具类

```
package com.wineshop.util;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.mchange.v2.c3p0.ComboPooledDataSource;

public class C3P0Utils {
    private static ComboPooledDataSource dataSource = new
ComboPooledDataSource();

    public static Connection getConnection() {
        try {
            return dataSource.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
            throw new ExceptionInInitializerError("初始化失败，请检查配置文件");
        }
    }


    /**
     * 关闭连接对象
     * @param conn
     * @param stmt
     * @param rs
     */
    public static void release(Connection conn, Statement stmt, ResultSet rs) {
        if(rs != null) {
            try {
                rs.close();
            } catch (SQLException e) {
```

```
                e.printStackTrace();
            }
            rs = null; //赶紧垃圾回收
        }
        if(stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            stmt = null;
        }
        if(conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
            conn = null;
        }
    }
}
```

5.3 jdbc 基础工具类

```
package com.xinhua.util;

import com.wineshop.util.C3POUtils;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.SimpleDateFormat;


/**
 * 对数据库操作底层封装
 * @author zhouk
 *
 */
public class JdbcUtil {
//  // 成员变量  连接数据库的url
//  public static String url="jdbc:mysql://localhost:3306/news";
//  public static String username="root";//用户名
//  public static String pwd="root";//密码
//  /**
//   * 连接数据库
//   */
//  public Connection getCoon() {
//      Connection con=null;
//      try {
//          Class.forName("com.mysql.jdbc.Driver");
//          //获取连接
//          try {
```

```
//              con=DriverManager.getConnection(url, username, pwd);
//          } catch (SQLException e) {
//              System.out.println("连接数据库异常");
//          }
//      } catch (ClassNotFoundException e) {
//          System.out.println("不好意思，你的驱动包没找到");
//      }
//      return con;
//  }

    /**
     * 查询 （通用查询）
     * @param sql  通用的sql语句
     */
    public ResultSet  querySql(String sql) {
        //连接数据库
        Connection con= C3POUtils.getConnection();
        Statement st=null;
        ResultSet rs=null;
        //创建执行对象
        try {
            st=con.createStatement();
            rs=st.executeQuery(sql);
        } catch (SQLException e) {
            System.out.println("创建statement对象有问题");
            e.printStackTrace();
        }
        return rs;
    }


    /**
     * 新增  修改 删除 公用方法
     */
    public int insertOrUpdateOrDeleteInfo(String sql) {
        Connection con=C3POUtils.getConnection();
        Statement st=null;
        int num=0;
        try {
            st=con.createStatement();
            num=st.executeUpdate(sql);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return num;
    }

}
```

5.4 通用表的增删改查工具类(可选择使用)

5.4.1 反射机制通用ReflectionUtil.java

```
package com.wineshop.util;
import java.lang.reflect.Field;
```

```
import java.lang.reflect.ParameterizedType;
import java.lang.reflect.Type;

/**
 * 工具类
 */
public class ReflectionUtils {
    /**
     * 通过反射，获得Class定义中声明的父类的泛型参数类型
     *
     * @param clazz 类
     * @return
     */
    public static <T> Class<T> getSuperGenericType(Class clazz) {
        return getSuperClassGenricType(clazz, 0);
    }

    /**
     * 通过反射, 获得定义 Class 时声明的父类的泛型参数的类型
     *
     * @param clazz 类
     * @param index 索引
     * @return 类
     */
    public static Class getSuperClassGenricType(Class clazz, int index) {
        Type genType = clazz.getSuperclass();
        if (!(genType instanceof ParameterizedType)) {
            return Object.class;
        }
        Type[] params = ((ParameterizedType) genType).getActualTypeArguments();
        if (index >= params.length || index < 0) {
            return Object.class;
        }
        if (!(params[index] instanceof Class)) {
            return Object.class;
        }
        return (Class) params[index];
    }

    /**
     * 循环向上转型, 获取对象的 DeclaredMethod
     *
     * @param object    类
     * @param fieldName 属性
     * @return 属性
     */
    public static Field getDeclaredField(Object object, String fieldName) {
        Field field;
        Class<?> clazz = object.getClass();
        for (; clazz != Object.class; clazz = clazz.getSuperclass()) {
            try {
                field = clazz.getDeclaredField(fieldName);
                return field;
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return null;
```

```
    }

    /**
     * 设置参数
     *
     * @param object    对象
     * @param fieldName 属性
     * @param value     值
     */
    public static void setFieldValue(Object object, String fieldName, Object
value) {
        Field field = getDeclaredField(object, fieldName);
        assert field != null;
        field.setAccessible(true);
        try {
            field.set(object, value);
        } catch (IllegalArgumentException | IllegalAccessException e) {
            e.printStackTrace();
        }
    }

    /**
     * 获取参数
     *
     * @param object    对象
     * @param fieldName 属性
     * @return 值
     */
    public static Object getFieldValue(Object object, String fieldName) {
        Field field = getDeclaredField(object, fieldName);
        assert field != null;
        field.setAccessible(true);
        try {
            return field.get(object);
        } catch (IllegalArgumentException | IllegalAccessException e) {
            e.printStackTrace();
        }
        return null;
    }
}
```

5.4.2 通用增删改查工具类 DbUtils.java

```
package com.wineshop.util;
import java.io.InputStream;
import java.sql.*;
import java.util.*;
/**
 * 通用的增删改查对象
 */
public class DbUtils {

    /**
     * 获取记录数
     */
```

```
public int getCount(String sql) throws SQLException {
    Connection connection = C3P0Utils.getConnection();
    Statement st= connection.createStatement();
    ResultSet rs=st.executeQuery(sql);
    int nums=0;
    while(rs.next()){
        nums++;
    }
    return nums;
}

/**
 * 通用查询
 *
 * @param classes 类
 * @param sql     sql语句
 * @param args    查询参数
 * @param <T>     泛型
 * @return 泛型对象
 */
public static <T> T query(Class<T> classes, String sql, Object... args) {
    T object = null;
    Connection connection = null;
    PreparedStatement preparedStatement = null;
    ResultSet resultSet = null;
    ResultSetMetaData resultSetMetaData;
    try {
        connection = C3P0Utils.getConnection();
        preparedStatement = connection.prepareStatement(sql);
        for (int i = 0; i < args.length; i++) {
            preparedStatement.setObject(i + 1, args[i]);
        }
        resultSet = preparedStatement.executeQuery();
        resultSetMetaData = resultSet.getMetaData();
        Map<String, Object> map = new HashMap<>(16);
        if (resultSet.next()) {
            for (int i = 0; i < resultSetMetaData.getColumnCount(); i++) {
                String columnname = resultSetMetaData.getColumnLabel(i + 1);
                Object obj = resultSet.getObject(i + 1);
                map.put(columnname, obj);
            }
        }
        if (map.size() > 0) {
            object = classes.newInstance();
            for (Map.Entry<String, Object> entry : map.entrySet()) {
                String fieldName = entry.getKey();
                Object value = entry.getValue();
                ReflectionUtils.setFieldValue(object, fieldName, value);
            }
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        C3P0Utils.release(connection, preparedStatement, resultSet);

    }
    return object;
}
```

```
    /**
     * 通用集合查询
     *
     * @param classes 类
     * @param sql     sql语句
     * @param args    查询参数
     * @param <T>     泛型
     * @return 泛型集合
     */
    public static <T> List<T> list(Class<T> classes, String sql, Object... args)
{
        List<T> objects = new ArrayList<>();
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        ResultSetMetaData resultSetMetaData;
        try {
            connection = C3POUtils.getConnection();
            preparedStatement = connection.prepareStatement(sql);

            if(args!=null){
                for (int i = 0; i < args.length; i++) {
                    preparedStatement.setObject(i + 1, args[i]);
                }
            }
            resultSet = preparedStatement.executeQuery();
            resultSetMetaData = resultSet.getMetaData();
            Map<String, Object> map = new HashMap<>(16);
            while (resultSet.next()) {
                T object = null;
                for (int i = 0; i < resultSetMetaData.getColumnCount(); i++) {
                    String columnname = resultSetMetaData.getColumnLabel(i + 1);
                    Object obj = resultSet.getObject(i + 1);
                    map.put(columnname, obj);
                }
                if (map.size() > 0) {
                    object = classes.newInstance();
                    for (Map.Entry<String, Object> entry : map.entrySet()) {
                        String fieldName = entry.getKey();
                        Object value = entry.getValue();
                        ReflectionUtils.setFieldValue(object, fieldName, value);
                    }
                }
                objects.add(object);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            C3POUtils.release(connection, preparedStatement, resultSet);

        }
        return objects;
    }

    /**
     * 通用插入
```

```
     *
     * @param sql  sql语句
     * @param args 查询参数
     * @return ture/fasle
     */
    public static boolean save(String sql, Object... args) {
        boolean state = true;
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = C3P0Utils.getConnection();
            preparedStatement = connection.prepareStatement(sql);
            for (int i = 0; i < args.length; i++) {
                preparedStatement.setObject(i + 1, args[i]);
            }
            preparedStatement.execute();
        } catch (Exception e) {
            state = false;
            e.printStackTrace();
        } finally {
            C3P0Utils.release(connection, preparedStatement, resultSet);
        }
        return state;
    }

    /**
     * 通用删除
     *
     * @param sql  sql语句
     * @param args 查询参数
     * @return ture/fasle
     */
    public static boolean remove(String sql, Object... args) {
        boolean state = true;
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = C3P0Utils.getConnection();
            preparedStatement = connection.prepareStatement(sql);
            for (int i = 0; i < args.length; i++) {
                preparedStatement.setObject(i + 1, args[i]);
            }
            preparedStatement.execute();
        } catch (Exception e) {
            state = false;
            e.printStackTrace();
        } finally {
            C3P0Utils.release(connection, preparedStatement, resultSet);
        }
        return state;
    }

    /**
     * 通用更新
     *
     * @param sql  sql语句
```

```
     * @param args 查询参数
     * @return ture/fasle
     */
    public static boolean update(String sql, Object... args) {
        boolean state = true;
        Connection connection = null;
        PreparedStatement preparedStatement = null;
        ResultSet resultSet = null;
        try {
            connection = C3POUtils.getConnection();
            preparedStatement = connection.prepareStatement(sql);
            for (int i = 0; i < args.length; i++) {
                preparedStatement.setObject(i + 1, args[i]);
            }
            preparedStatement.execute();
        } catch (Exception e) {
            state = false;
            e.printStackTrace();
        } finally {
            C3POUtils.release(connection, preparedStatement, resultSet);
        }
        return state;
    }
}
```

5.5 返回对象的封装

5.5.1 常用返回标识 Constants.java 常量类

```
package com.wineshop.util;

/**
 * 常量类
 */
public class Constants {

    public final static int OK_CODE = 0;
    public final static int FAIL_CODE = 400;
    public final static int OTHER_FAIL_CODE = 333;    // 其它错误
    public final static String OK_MSG = "请求成功";
    public final static String FAIL_MSG = "请求失败";
    public final static int STATUS_0 = 0;   // 可用状态
    public final static int STATUS_1 = 1;   // 禁用状态

    public final static String CACHE_NAME = "KACache";

}
```

5.5.2 返回对象封装 R.java

```
package com.wineshop.util;



import java.io.Serializable;
```

```
/**
 * 返回前端 数据封闭类
 */
public class R implements Serializable {

    private static final long serialVersionUID = 1L;

    private Integer code;
    private String msg;
    private Object data;
    private Long count; // 分页信息：总条数

    public R() { }

    private R(int code, String msg, Object data) {
        this.code = code;
        this.msg = msg;
        this.data=data;
    }

    private R(int code, String msg, Object data,long count) {
        this.code = code;
        this.msg = msg;
        this.data=data;
        this.count=count;
    }

    public static R ok() {
        return new R(Constants.OK_CODE, Constants.OK_MSG, null);
    }

    public static R ok(Object data) {
        return new R(Constants.OK_CODE, Constants.OK_MSG, data);
    }

    public static R ok(String msg, long count, Object data) {
        return new R(Constants.OK_CODE, Constants.OK_MSG, data,count);
    }

    public static R ok(String msg, Object data) {
        return new R(Constants.OK_CODE, msg, data);
    }

    public static R fail(String msg) {
        return new R(Constants.FAIL_CODE, msg, null);
    }

    public static R fail(int errorCode, String msg) {
        return new R(errorCode, msg, null);
    }
    public int getCode() {
        return code;
    }

    public String getMsg() {
        return msg;
    }
```

```
    public Object getData() {
        return data;
    }

    public Long getCount() {
        return count;
    }

    public R setCount(Long count) {
        this.count =count;
        return this;
    }

}
```

5.5.3 查询返回列表对象封装 和前端json数据格式保持一致 JsonObject.java

```
package com.wineshop.model;

import java.util.List;

public class JsonObject {
   private Integer code;
   private String msg;
   private List data;
   private Integer count;

    public Integer getCode() {
        return code;
    }

    public void setCode(Integer code) {
        this.code = code;
    }

    public String getMsg() {
        return msg;
    }

    public void setMsg(String msg) {
        this.msg = msg;
    }

    public List getData() {
        return data;
    }

    public void setData(List data) {
        this.data = data;
    }

    public Integer getCount() {
        return count;
    }
```

```
    public void setCount(Integer count) {
        this.count = count;
    }
}
```

5.6 分页工具类PageBean.java

```
package com.wineshop.util;


import java.util.List;


public class PageBean<T> {
    /**
     * 当前页， 默认显示第一页
     */
    private Integer currntPage = 1;
    /**
     * 查询返回的行数（每页显示的行数），默认每页显示10行
     */
    private int pageCount =5;
    /**
     * 总记录数
     */
    private int totalCount;
    /**
     * 总页数 = 总记录数/每页显示的行数（+1）
     */
    private int totalPage;
    /**
     * 分页查询的数据,运用泛型，可以重复利用
     */
    private List<T> pageData;

    public int getTotalPage() {
        if (totalCount % pageCount == 0) {
            totalPage = totalCount / pageCount;
        } else {
            totalPage = totalCount / pageCount + 1;
        }
        return totalPage;
    }

    public void setTotalPage(int totalPage) {
        this.totalPage = totalPage;
    }

    public int getCurrntPage() {
        return currntPage;
    }

    public void setCurrntPage(int currntPage) {
        this.currntPage = currntPage;
    }
```

```
    public int getPageCount() {
        return pageCount;
    }

    public void setPageCount(int pageCount) {
        this.pageCount = pageCount;
    }

    public int getTotalCount() {
        return totalCount;
    }

    public void setTotalCount(int totalCount) {
        this.totalCount = totalCount;
    }


    public List<T> getPageData() {
        return pageData;
    }

    public void setPageData(List<T> pageData) {
        this.pageData = pageData;
    }
}
```

6、乱码处理过滤器EncodingFilter.java

```
package com.wineshop.filter;

import javax.servlet.*;
import java.io.IOException;

public class EncodingFilter implements Filter {

    String code=null;
    public void destroy() {
        // TODO Auto-generated method stub
    }

    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
        if(code!=null) {
            request.setCharacterEncoding(code);
            response.setContentType("text/html; charset="+code);
        }

        chain.doFilter(request, response);
    }

    /**
     * @see Filter#init(FilterConfig)
     */
    public void init(FilterConfig fConfig) throws ServletException {
        code=fConfig.getInitParameter("codeEning");
    }
```

```
}
```

6.2 拦截器配置处理web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0" metadata-complete="false">
    <filter>
        <filter-name>filters</filter-name>
        <filter-class>com.wineshop.filter.EncodingFilter</filter-class>
        <init-param>
            <param-name>codeEning</param-name>
            <param-value>utf-8</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>filters</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

</web-app>
```