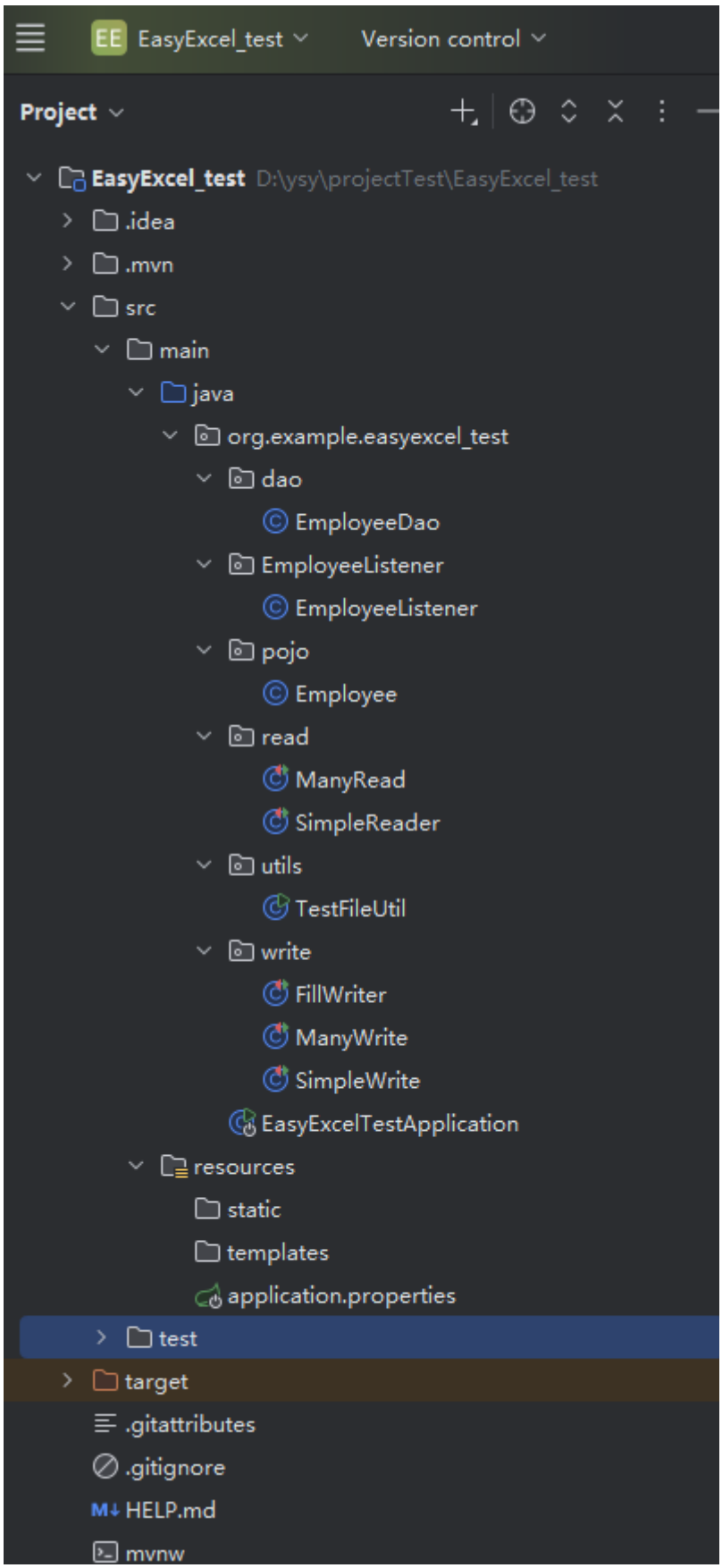
EasyExcel学习

1.环境搭建

```
<dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>easyexcel</artifactId>
    <version>3.1.2</version>
</dependency>
```

项目结构:


EasyExcel_test
Version control
Project
EasyExcel_test D:\ysy\projectTest\EasyExcel_test
.idea
.mvn
src
main
java
org.example.easyexcel_test
dao
EmployeeDao
EmployeeListener
EmployeeListener
pojo
Employee
read
ManyRead
SimpleReader
utils
TestFileUtil
write
FillWriter
ManyWrite
SimpleWrite
EasyExcelTestApplication
resources
static
templates
application.properties
test
target
.gitattributes
.gitignore
HELP.md
mvnw

2.本地模拟实现

1.简单写入excel

第一步：首先创建实体类Employee

> @ExcelProperty(value = "员工id",index = 0)注解就是代表的excel表的表头的每一列的名字

```
@Data@Data

@NoArgsConstructor
@AllArgsConstructor
public class Employee {
    @ExcelProperty(value = "员工id",index = 0)
    private Integer id;
    @ExcelProperty(value = "员工姓名",index = 1)
    private String name;
    @ExcelProperty(value = "入职日期",index = 2)
    private Date date;
    @ExcelProperty(value = "员工工资",index = 3)
    private Double salary;
}
```

第二步：创建工具类：用于获取当前文件夹的路径

```
public class TestFileUtil {
    public static String getPath(){
        return TestFileUtil.class.getResource("/").getPath().replace("classes/", "");
    }

    public static void main(String[] args) {
        System.out.println(getPath());
    }
}
```

第三步：写操作主要代码：

```
public class SimpleWrite {

    //准备测试数据的方法
    private List<Employee> data(int count) {
        List<Employee> list = ListUtils.newArrayList();
        for (int i = 1; i < count; i++) {

            list.add(new Employee(i,"测试数据1"+i,new Date(),6.6*i));
        }
        return list;
    }
    //快速入门写数据
    @Test
    public void write(){
          String  fileName = TestFileUtil.getPath() + "simpleWrite" +
System.currentTimeMillis() + ".xlsx";
        // 这里 需要指定写用哪个class去写，然后写到第一个sheet，名字为模板 然后文件流会自动关闭
```

```
        EasyExcel.write(fileName, Employee.class).sheet("模板").doWrite(data(10));
    }
}
```

2.简单读出excel

只需要一个方法，就可以读出excel表格信息并在控制台打印

```
//快速入门读数据
public class SimpleReader {
    @Test
    public void read(){

        String fileName = TestFileUtil.getPath() +  "simpleWrite1762401148464.xlsx";
        // 这里默认每次会读取100条数据 然后返回过来 直接调用使用数据就行
        // 具体需要返回多少行可以在`PageReadListener`的构造函数设置
        EasyExcel.read(fileName, Employee.class, new PageReadListener<Employee>(dataList ->
{
            for (Employee demoData : dataList) {
                System.out.println(demoData);
            }
        })).sheet().doRead();
    }
}
```

3.批量写入excel

```
public class ManyWrite {
    //准备测试数据的方法
    private List<Employee> data(int count) {
        List<Employee> list = ListUtils.newArrayList();
        for (int i = 1; i < count; i++) {
            list.add(new Employee(i,"测试数据1"+i,new Date(),6.6*i));
        }
        return list;
    }
    //批量写数据
    @Test
    public void write(){
        // 方法2: 如果写到不同的sheet 同一个对象
        String fileName = TestFileUtil.getPath() + "repeatedWrite" +
System.currentTimeMillis() + ".xlsx";
        // 这里 指定文件
        try (ExcelWriter excelWriter = EasyExcel.write(fileName, Employee.class).build()) {

            WriteSheet writeSheet = EasyExcel.writerSheet( "测试数据").build();
            long t1 =System.currentTimeMillis();
            // 去调用写入
            for (int i = 0; i < 100; i++) {

                // 分页去数据库查询数据 这里可以去数据库查询每一页的数据
                List<Employee> data = data(10000);
```

```
                excelWriter.write(data, writeSheet);
            }
            long t2 =System.currentTimeMillis();
            System.out.println("耗时: "+(t2-t1));
        }
    }
}
```

4.读海量数据

读海量数据需要自定义监听器EmployeeListener

```
//自定义监听器读数据
public class EmployeeListener implements ReadListener<Employee> {
    private ArrayList<Employee> list = new ArrayList<>();
    private int count = 100;
    private EmployeeDao dao;
    public EmployeeListener(EmployeeDao dao) {
        this.dao = dao;
    }
    //每读完一行数据，就会调用此方法
    @Override
    public void invoke(Employee employee, AnalysisContext analysisContext) {
        //将读取的一行数据保存到list中
        list.add(employee);
        //判断是不是到达缓存量了
        if(list.size()>=100){
            //操作数据库
            dao.save(list);
            list = new ArrayList<>(count);
        }
    }
    //读完整个excel之后后调用此方法
    @Override
    public void doAfterAllAnalysed(AnalysisContext analysisContext) {
        if(list.size()>0){
            //操作数据库
            dao.save(list);
            list = new ArrayList<>(count);
        }
    }
}
```

以下是Dao层模拟数据库操作

```
//模拟操作数据库
public class EmployeeDao {
    public void save(List<Employee> list){
        System.out.println(list.size()+"模拟操作数据库");

    }
}
```

```
//读海量数据
public class ManyRead {
    @Test
    public void read(){
        String fileName = TestFileUtil.getPath() +  "repeatedWrite1762406295595.xlsx";
        ExcelReader reader = EasyExcel.read(fileName, Employee.class, new
EmployeeListener(new EmployeeDao())).build();
        ReadSheet sheet = EasyExcel.readSheet().build();
        reader.read(sheet);
    }
}
```

5.使用模板填充Excel表格

首先在模板中用{.}定义好每一列的名称对应什么

	A	B	C	D
1	员工id	员工姓名	入职日期	员工工资
2	{.id}	{.name}	{.date}	{.salary}
3				
4				

```
//练习填充数据
public class FillWriter {
    //准备测试数据的方法
    private List<Employee> data(int count) {
        List<Employee> list = ListUtils.newArrayList();
        for (int i = 1; i < count; i++) {

            list.add(new Employee(i,"测试数据1"+i,new Date(),6.6*i));
        }
        return list;
    }
    @Test
    public void write(){
        String fileName = TestFileUtil.getPath() + "listFill" + System.currentTimeMillis()
+ ".xlsx";
        String templateFileName = TestFileUtil.getPath() + "模板.xlsx";
        try (ExcelWriter excelWriter =
EasyExcel.write(fileName).withTemplate(templateFileName).build()) {
            WriteSheet writeSheet = EasyExcel.writerSheet().build();
            long t1 = System.currentTimeMillis();
            for (int i = 0; i < 100; i++) {
                excelWriter.fill(data(10000), writeSheet);
            }
            long t2 = System.currentTimeMillis();
            System.out.println("耗时: "+(t2-t1));
        }
    }
}
```

3.前后端端实现导入导出Excel

listener定义:

```
package com.itheima.listener;

import com.alibaba.excel.context.AnalysisContext;
import com.alibaba.excel.read.listener.ReadListener;
import com.itheima.domain.Employee;
import com.itheima.service.EmployeeService;

import java.util.ArrayList;
import java.util.List;

public class EmployeeListener implements ReadListener<Employee> {
    /**
     * 批量处理的数据量阈值
     */
    private int count = 10000;
    private EmployeeService dao ;

    /**
     * 临时存储读取的数据列表
     */
    private List<Employee> list = new ArrayList<>(count);

    public EmployeeListener(EmployeeService dao) {
        this.dao = dao;
    }

    /**
     * 处理每一行数据
     * 当读取到一行数据时会调用此方法
     * @param employee 当前行的员工数据
     * @param analysisContext 分析上下文
     */
    @Override
    public void invoke(Employee employee, AnalysisContext analysisContext) {
        list.add(employee);
        // 当累积的数据量达到阈值时，执行批量插入操作
        if(list.size()>=count){
            dao.addData(list);
            // 重新初始化列表，准备下一批数据
            list = new ArrayList<>(count);
        }
    }

    /**
     * 所有数据解析完成后的回调方法
     * 用于处理剩余未达到批量阈值的数据
     * @param analysisContext 分析上下文
     */
```

```
    @Override
    public void doAfterAllAnalysed(AnalysisContext analysisContext) {
        // 处理最后一批未达到阈值的数据
        if(list.size()>0){
            dao.addData(list);
        }
    }
}
```

pojo层：

pojo层和上面定义一致：

```
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Employee {
    @ExcelProperty(value = "员工id",index = 0)
    private Integer id;
    @ExcelProperty(value = "员工姓名",index = 1)
    private String name;
    @ExcelProperty(value = "入职日期",index = 2)
    private Date date;
    @ExcelProperty(value = "员工工资",index = 3)
    private Double salary;
}
```

controller层：

主要是两个方法，一个上传，一个下载

```
@Controller
@RequestMapping("/")
public class MyController {
    @Autowired
    private EmployeeService service;

    @RequestMapping("/upload")
    @ResponseBody
    public void upload(MultipartFile file, HttpServletResponse response) throws IOException
{
        long start = System.currentTimeMillis();
        EasyExcel.read(file.getInputStream(), Employee.class, new
EmployeeListener(service)).sheet().doRead();
        long end = System.currentTimeMillis();
        response.setContentType("text/html;charset=utf-8");

        response.getWriter().print("上传成功,共用时："+(end-start));

    }

    @RequestMapping("/download")
    public void download(HttpServletResponse response) throws IOException {
```

```
        // 这里注意 使用swagger 会导致各种问题，直接用浏览器或者用postman
        response.setContentType("application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet");
        response.setCharacterEncoding("utf-8");
        // 这里URLEncoder.encode可以防止中文乱码 当然和easyexcel没有关系
        String fileName = URLEncoder.encode("数据库中导出的数据", "UTF-8").replaceAll("\\+",
"%20");
        response.setHeader("Content-disposition", "attachment;filename*=utf-8''" + fileName
+ ".xlsx");
        ExcelWriter writer = EasyExcel.write(response.getOutputStream(),
Employee.class).build();
        WriteSheet sheet = EasyExcel.writerSheet("数据").build();
        writer.write(service.getData(), sheet);
        writer.close();
    }
}
```

Service层

```
package com.itheima.service;

import com.itheima.domain.Employee;
import org.springframework.stereotype.Service;

import java.util.List;

public interface EmployeeService {
    public List<Employee> getData();
    public void addData(List<Employee> list);
}
```

```
package com.itheima.service.impl;

import com.itheima.domain.Employee;
import com.itheima.mapper.EmployeeMapper;
import com.itheima.service.EmployeeService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class EmployeeServiceImpl implements EmployeeService {

    @Autowired
    private EmployeeMapper dao;

    @Override
    public List<Employee> getData() {
        return dao.getData();
    }
```

```
    @Override
    public void addData(List<Employee> list) {
        dao.beathInsert(list);
    }
}
```

Mapper层

```
public interface EmployeeMapper {
    public void beathInsert(@Param("list") List<Employee> list);

//    @Select("select * from employee ")
    @Select("select * from employee")
    @ResultType(Employee.class)
    List<Employee> getData();

}
```