第8章 POI报表的高级应用

- 掌握基于模板打印的POI报表导出
- 理解自定义工具类的执行流程
- 熟练使用SXSSFWorkbook完成百万数据报表打印
- 理解基于事件驱动的POI报表导入

1 模板打印

1.1 概述

自定义生成Excel报表文件还是有很多不尽如意的地方，特别是针对复杂报表头，单元格样式，字体等操作。手写这些代码不仅费时费力，有时候效果还不太理想。那怎么样才能更方便的对报表样式，报表头进行处理呢？答案是使用已经准备好的Excel模板，只需要关注模板中的数据即可。

1.2 模板打印的操作步骤

1. 制作模版文件（模版文件的路径）
2. 导入（加载）模版文件，从而得到一个工作簿
3. 读取工作表
4. 读取行
5. 读取单元格
6. 读取单元格样式
7. 设置单元格内容
8. 其他单元格就可以使用读到的样式了

1.3 代码实现

```
@RequestMapping(value = "/export/{month}", method = RequestMethod.GET)
    public void export(@PathVariable(name = "month") String month) throws Exception {
        //1.构造数据
        List<EmployeeReportResult> list =
userCompanyPersonalService.findByReport(companyId,month+"%");

        //2.加载模板流数据
        Resource resource = new ClassPathResource("excel-template/hr-demo.xlsx");
        FileInputStream fis = new FileInputStream(resource.getFile());

        //3.根据文件流，加载指定的工作簿
        XSSFWorkbook wb = new XSSFWorkbook(fis);

        //4.读取工作表
        Sheet sheet = wb.getSheetAt(0);

        //5.抽取公共的样式
        Row styleRow = sheet.getRow(2);
```

```
        CellStyle [] styles = new CellStyle[styleRow.getLastCellNum()];
        for(int i=0;i<styleRow.getLastCellNum();i++) {
            styles[i] = styleRow.getCell(i).getCellStyle();
        }

        //6.构造每行和单元格数据
        AtomicInteger datasAi = new AtomicInteger(2);

        Cell cell = null;
        for (EmployeeReportResult report : list) {
            Row dataRow = sheet.createRow(datasAi.getAndIncrement());
            //编号
            cell = dataRow.createCell(0);
            cell.setCellValue(report.getUserId());
            cell.setCellStyle(styles[0]);
            //姓名
            cell = dataRow.createCell(1);
            cell.setCellValue(report.getUsername());
            cell.setCellStyle(styles[1]);
            //手机
            cell = dataRow.createCell(2);
            cell.setCellValue(report.getMobile());
            cell.setCellStyle(styles[2]);
            //最高学历
            cell = dataRow.createCell(3);
            cell.setCellValue(report.getTheHighestDegreeOfEducation());
            cell.setCellStyle(styles[3]);
            //国家地区
            cell = dataRow.createCell(4);
            cell.setCellValue(report.getNationalArea());
            cell.setCellStyle(styles[4]);
            //护照号
            cell = dataRow.createCell(5);
            cell.setCellValue(report.getPassportNo());
            cell.setCellStyle(styles[5]);
            //籍贯
            cell = dataRow.createCell(6);
            cell.setCellValue(report.getNativePlace());
            cell.setCellStyle(styles[6]);
            //生日
            cell = dataRow.createCell(7);
            cell.setCellValue(report.getBirthday());
            cell.setCellStyle(styles[7]);
            //属相
            cell = dataRow.createCell(8);
            cell.setCellValue(report.getZodiac());
            cell.setCellStyle(styles[8]);
            //入职时间
            cell = dataRow.createCell(9);
            cell.setCellValue(report.getTimeOfEntry());
            cell.setCellStyle(styles[9]);
            //离职类型
            cell = dataRow.createCell(10);
```

```
            cell.setCellValue(report.getTypeOfTurnover());
            cell.setCellStyle(styles[10]);
            //离职原因
            cell = dataRow.createCell(11);
            cell.setCellValue(report.getReasonsForLeaving());
            cell.setCellStyle(styles[11]);
            //离职时间
            cell = dataRow.createCell(12);
            cell.setCellStyle(styles[12]);
            cell.setCellValue(report.getResignationTime());
        }
        String fileName = URLEncoder.encode(month+"人员信息.xlsx", "UTF-8");
        response.setContentType("application/octet-stream");
        response.setHeader("content-disposition", "attachment;filename=" + new
String(fileName.getBytes("ISO8859-1")));
        response.setHeader("filename", fileName);
        wb.write(response.getOutputStream());
    }
```

2 自定义工具类

2.1 自定义注解

（1）自定义注解

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface ExcelAttribute {
    /** 对应的列名称 */
    String name() default "";

    /** 列序号 */
    int sort();

    /** 字段类型对应的格式 */
    String format() default "";

}
```

（2）导出工具类

```
@Getter
@Setter
public class ExcelExportUtil<T> {

    private int rowIndex;
    private int styleIndex;
    private String templatePath;
    private Class clazz;
    private  Field fields[];
```

```
    public ExcelExportUtil(Class clazz,int rowIndex,int styleIndex) {
        this.clazz = clazz;
        this.rowIndex = rowIndex;
        this.styleIndex = styleIndex;
        fields = clazz.getDeclaredFields();
    }

    /**
     * 基于注解导出
     */
    public void export(HttpServletResponse response,InputStream is, List<T> objs,String
fileName) throws Exception {

        XSSFWorkbook workbook = new XSSFWorkbook(is);
        Sheet sheet = workbook.getSheetAt(0);

        CellStyle[] styles = getTemplateStyles(sheet.getRow(styleIndex));

        AtomicInteger datasAi = new AtomicInteger(rowIndex);
        for (T t : objs) {
            Row row = sheet.createRow(datasAi.getAndIncrement());
            for(int i=0;i<styles.length;i++) {
                Cell cell = row.createCell(i);
                cell.setCellStyle(styles[i]);
                for (Field field : fields) {
                    if(field.isAnnotationPresent(ExcelAttribute.class)){
                        field.setAccessible(true);
                        ExcelAttribute ea = field.getAnnotation(ExcelAttribute.class);
                        if(i == ea.sort()) {
                            cell.setCellValue(field.get(t).toString());
                        }
                    }
                }
            }
        }
        fileName = URLEncoder.encode(fileName, "UTF-8");
        response.setContentType("application/octet-stream");
        response.setHeader("content-disposition", "attachment;filename=" + new
String(fileName.getBytes("ISO8859-1")));
        response.setHeader("filename", fileName);
        workbook.write(response.getOutputStream());
    }

    public CellStyle[] getTemplateStyles(Row row) {
        CellStyle [] styles = new CellStyle[row.getLastCellNum()];
        for(int i=0;i<row.getLastCellNum();i++) {
            styles[i] = row.getCell(i).getCellStyle();
        }
        return styles;
    }
}
```

（3）导入工具类

```
public class ExcelImportUtil<T> {

    private Class clazz;
    private  Field fields[];

    public ExcelImportUtil(Class clazz) {
        this.clazz = clazz;
        fields = clazz.getDeclaredFields();
    }

    /**
     * 基于注解读取excel
     */
    public List<T> readExcel(InputStream is, int rowIndex,int cellIndex) {
        List<T> list = new ArrayList<T>();
        T entity = null;
        try {
            XSSFWorkbook workbook = new XSSFWorkbook(is);
            Sheet sheet = workbook.getSheetAt(0);
            // 不准确
            int rowLength = sheet.getLastRowNum();

            System.out.println(sheet.getLastRowNum());
            for (int rowNum = rowIndex; rowNum <= sheet.getLastRowNum(); rowNum++) {
                Row row = sheet.getRow(rowNum);
                entity = (T) clazz.newInstance();
                System.out.println(row.getLastCellNum());
                for (int j = cellIndex; j < row.getLastCellNum(); j++) {
                    Cell cell = row.getCell(j);
                    for (Field field : fields) {
                        if(field.isAnnotationPresent(ExcelAttribute.class)){
                            field.setAccessible(true);
                            ExcelAttribute ea =
field.getAnnotation(ExcelAttribute.class);
                            if(j == ea.sort()) {
                                field.set(entity, covertAttrType(field, cell));
                            }
                        }
                    }
                }
                list.add(entity);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return list;
    }


    /**
     * 类型转换 将cell 单元格格式转为 字段类型
```

```
     */
    private Object covertAttrType(Field field, Cell cell) throws Exception {
        String fieldType = field.getType().getSimpleName();
        if ("String".equals(fieldType)) {
            return getValue(cell);
        }else if ("Date".equals(fieldType)) {
            return new SimpleDateFormat("yyyy-MM-dd hh:mm:ss").parse(getValue(cell)) ;
        }else if ("int".equals(fieldType) || "Integer".equals(fieldType)) {
            return Integer.parseInt(getValue(cell));
        }else if ("double".equals(fieldType) || "Double".equals(fieldType)) {
            return Double.parseDouble(getValue(cell));
        }else {
            return null;
        }
    }


    /**
     * 格式转为String
     * @param cell
     * @return
     */
    public String getValue(Cell cell) {
        if (cell == null) {
            return "";
        }
        switch (cell.getCellType()) {
            case STRING:
                return cell.getRichStringCellValue().getString().trim();
            case NUMERIC:
                if (DateUtil.isCellDateFormatted(cell)) {
                    Date dt = DateUtil.getJavaDate(cell.getNumericCellValue());
                    return new SimpleDateFormat("yyyy-MM-dd hh:mm:ss").format(dt);
                } else {
                    // 防止数值变成科学计数法
                    String strCell = "";
                    Double num = cell.getNumericCellValue();
                    BigDecimal bd = new BigDecimal(num.toString());
                    if (bd != null) {
                        strCell = bd.toPlainString();
                    }
                    // 去除 浮点型 自动加的 .0
                    if (strCell.endsWith(".0")) {
                        strCell = strCell.substring(0, strCell.indexOf("."));
                    }
                    return strCell;
                }
            case BOOLEAN:
                return String.valueOf(cell.getBooleanCellValue());
            default:
                return "";
        }
    }
```

```
}
```

2.2 工具类完成导入导出

（1）导入数据

```
List<User> list = new ExcelImportUtil(User.class).readExcel(is, 1, 2);
```

（2）导出数据

```
    @RequestMapping(value = "/export/{month}", method = RequestMethod.GET)
    public void export(@PathVariable(name = "month") String month) throws Exception {
        //1.构造数据
        List<EmployeeReportResult> list =
userCompanyPersonalService.findByReport(companyId,month+"%");

        //2.加载模板流数据
        Resource resource = new ClassPathResource("excel-template/hr-demo.xlsx");
        FileInputStream fis = new FileInputStream(resource.getFile());

        new ExcelExportUtil(EmployeeReportResult.class,2,2).
                export(response,fis,list,"人事报表.xlsx");
    }
```

3 百万数据报表概述

3.1 概述

我们都知道Excel可以分为早期的Excel2003版本（使用POI的HSSF对象操作）和Excel2007版本（使用POI的XSSF操作），两者对百万数据的支持如下：

- Excel 2003：在POI中使用HSSF对象时，excel 2003最多只允许存储65536条数据，一般用来处理较少的数据量。这时对于百万级别数据，Excel肯定容纳不了。
- Excel 2007：当POI升级到XSSF对象时，它可以直接支持excel2007以上版本，因为它采用ooxml格式。这时excel可以支持1048576条数据，单个sheet表就支持近百万条数据。但实际运行时还可能存在问题，原因是执行POI报表所产生的行对象，单元格对象，字体对象，他们都不会销毁，这就导致OOM的风险。

3.2 JDK性能监控工具介绍

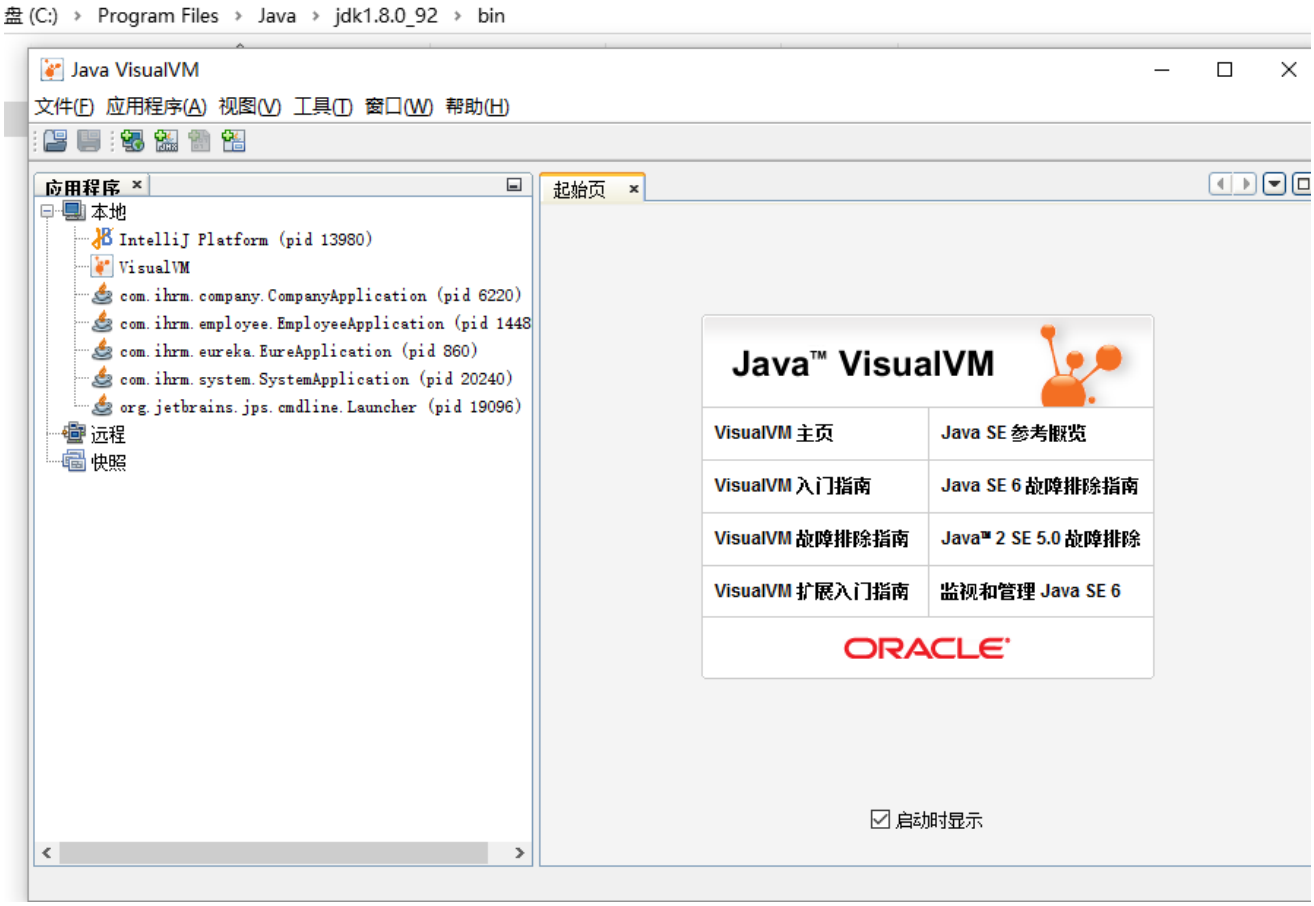
没有性能监控工具一切推论都只能停留在理论阶段，我们可以使用Java的性能监控工具来监视程序的运行情况，包括CUP,垃圾回收，内存的分配和使用情况，这让程序的运行阶段变得更加可控，也可以用来证明我们的推测。这里我们使用JDK提供的性能工具Jvisualvm来监控程序运行。

3.2.1 Jvisualvm概述

VisualVM 是Netbeans的profile子项目，已在JDK6.0 update 7 中自带，能够监控线程，内存情况，查看方法的CPU时间和内存中的对 象，已被GC的对象，反向查看分配的堆栈

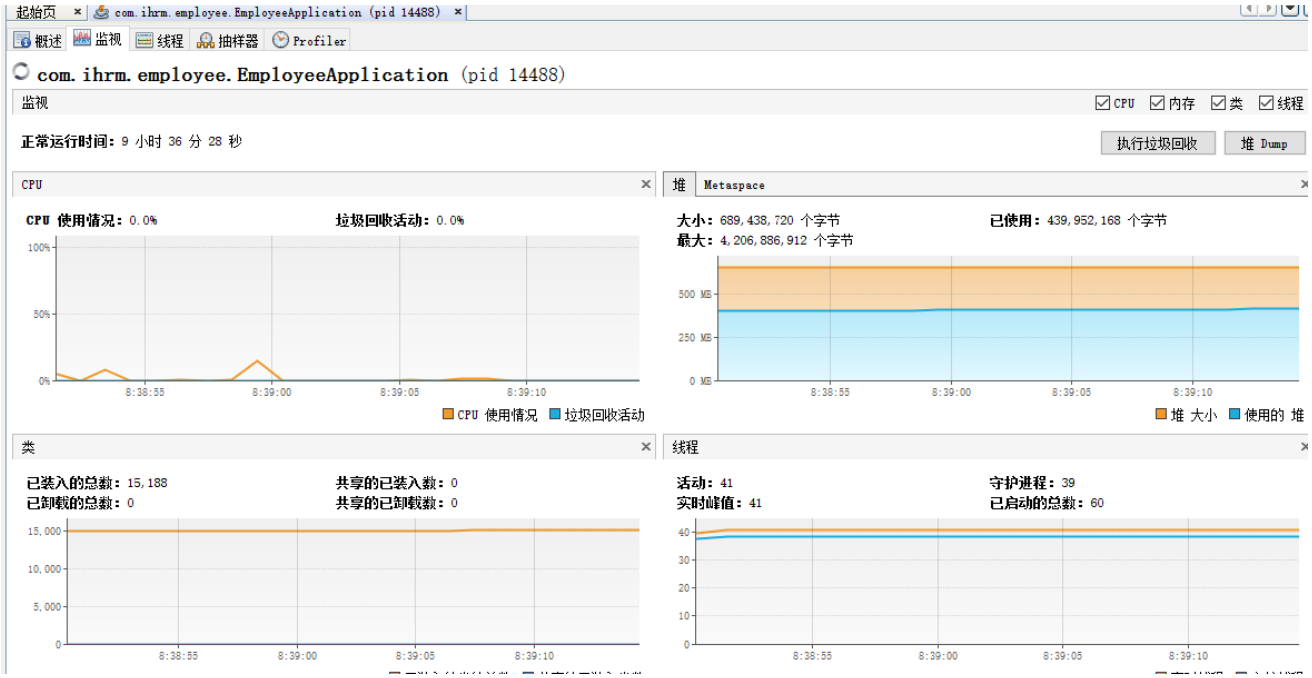
3.2.2 Jvisualvm的位置

Jvisualvm位于JAVA_HOME/bin目录下，直接双击就可以打开该程序。如果只是监控本地的java进程，是不需要配置参数的，直接打开就能够进行监控。首先我们需要在本地打开一个Java程序，例如我打开员工微服务进程，这时在jvisualvm界面就可以看到与IDEA相关的Java进程了：



3.2.3 Jvisualvm的使用

Jvisualvm使用起来比较简单，双击点击当前运行的进程即可进入到程序的监控界面



- 概述：可以看到进程的启动参数。
- 监视：左上：cpu利用率，gc状态的监控，右上：堆利用率，永久内存区的利用率，左下：类的监控，右下：线程的监控
- 线程：能够显示线程的名称和运行的状态，在调试多线程时必不可少，而且可以点进一个线程查看这个线程的详细运行情况

3.3 解决方案分析

对于百万数据量的Excel导入导出，只讨论基于Excel2007的解决方法。在ApachePoi 官方提供了对操作大数据量的导入导出的工具和解决办法，操作Excel2007使用XSSF对象，可以分为三种模式：

- 用户模式：用户模式有许多封装好的方法操作简单，但创建太多的对象，非常耗内存（之前使用的方法）
- 事件模式：基于SAX方式解析XML，SAX全称Simple API for XML，它是一个接口，也是一个软件包。它是一种XML解析的替代方法，不同于DOM解析XML文档时把所有内容一次性加载到内存中的方式，它逐行扫描文档，一边扫描，一边解析。
- SXSSF对象：是用来生成海量excel数据文件，主要原理是借助临时存储空间生成excel

	HSSF (.xls)		XSSF (.xlsx)		
	eventmodel	usermodel	eventmodel	usermodel	SXSSF
API Type	Streaming (ala SAX)	In memory tree (ala DOM)	Streaming (ala SAX)	In memory tree (ala DOM)	Buffered Streaming
CPU and Memory Efficiency	Good	Varies	Good	Varies	Good
Forward Only	Yes	No	Yes	No	Yes
Read Files	Yes	Yes	Yes	Yes	No
Write Files	No	Yes	No	Yes	Yes
Feature:					
create sheets / rows / cells	No	Yes	No	Yes	Yes
styling cells	No	Yes	No	Yes	Yes
delete sheets / rows/cells	No	Yes	No	Yes	No
shift rows	No	Yes	No	Yes	No
cloning sheets	No	Yes	No	Yes	No
formula evaluation	No	Yes	No	Yes	No
cell comments	No	Yes	No	Yes	No
pictures	No	Yes	No	Yes	Yes

这是一张Apache POI官方提供的图片，描述了基于用户模式，事件模式，以及使用SXSSF三种方式操作Excel的特性以及CUP和内存占用情况。

4 百万数据报表导出

4.1 需求分析

使用Apache POI完成百万数据量的Excel报表导出

4.2 解决方案

4.2.1 思路分析

基于XSSFWork导出Excel报表，是通过将所有单元格对象保存到内存中，当所有的Excel单元格全部创建完成之后一次性写入到Excel并导出。当百万数据级别的Excel导出时，随着表格的不断创建，内存中对象越来越多，直至内存溢出。Apache Poi提供了SXSSFWork对象，专门用于处理大数据量Excel报表导出。

4.2.2 原理分析

在实例化SXSSFWork这个对象时，可以指定在内存中所产生的POI导出相关对象的数量（默认100），一旦内存中的对象的个数达到这个指定值时，就将内存中的这些对象的内容写入到磁盘中（XML的文件格式），就可以将这些对象从内存中销毁，以后只要达到这个值，就会以类似的处理方式处理，直至Excel导出完成。

4.3 代码实现

在原有代码的基础上替换之前的XSSFWorkbook，使用SXSSFWorkbook完成创建过程即可

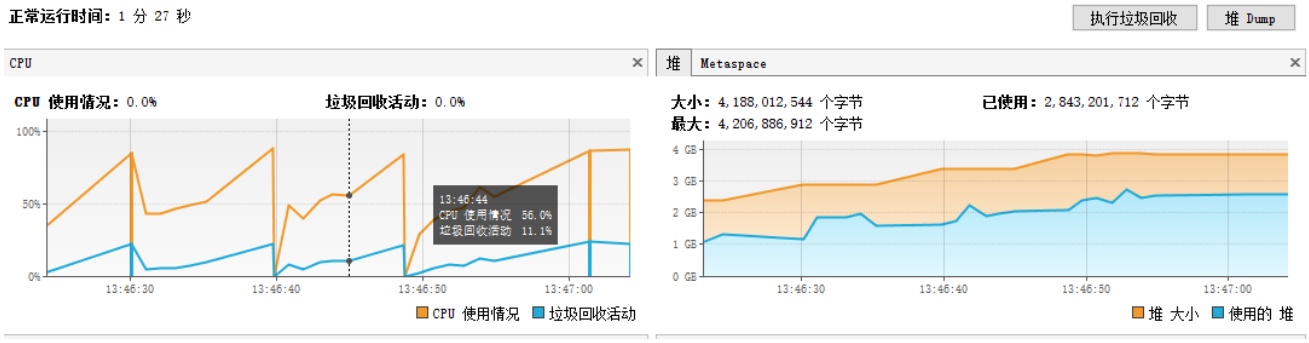
```
  //1.构造数据
        List<EmployeeReportResult> list =
userCompanyPersonalService.findByReport(companyId,month+"%");
        //2.创建工作簿
        SXSSFWorkbook workbook = new SXSSFWorkbook();
        //3.构造sheet
        String[] titles = {"编号", "姓名", "手机","最高学历", "国家地区", "护照号", "籍贯",
"生日", "属相","入职时间","离职类型","离职原因","离职时间"};

        Sheet sheet = workbook.createSheet();

        Row row = sheet.createRow(0);

        AtomicInteger headersAi = new AtomicInteger();

        for (String title : titles) {
            Cell cell = row.createCell(headersAi.getAndIncrement());
            cell.setCellValue(title);
        }

        AtomicInteger datasAi = new AtomicInteger(1);
```

```
        Cell cell = null;
        for(int i=0;i<10000;i++) {
            for (EmployeeReportResult report : list) {
                Row dataRow = sheet.createRow(datasAi.getAndIncrement());
                //编号
                cell = dataRow.createCell(0);
                cell.setCellValue(report.getUserId());
                //姓名
                cell = dataRow.createCell(1);
                cell.setCellValue(report.getUsername());
                //手机
                cell = dataRow.createCell(2);
                cell.setCellValue(report.getMobile());
                //最高学历
                cell = dataRow.createCell(3);
                cell.setCellValue(report.getTheHighestDegreeOfEducation());
                //国家地区
                cell = dataRow.createCell(4);
                cell.setCellValue(report.getNationalArea());
                //护照号
                cell = dataRow.createCell(5);
                cell.setCellValue(report.getPassportNo());
                //籍贯
                cell = dataRow.createCell(6);
                cell.setCellValue(report.getNativePlace());
                //生日
                cell = dataRow.createCell(7);
                cell.setCellValue(report.getBirthday());
                //属相
                cell = dataRow.createCell(8);
                cell.setCellValue(report.getZodiac());
                //入职时间
                cell = dataRow.createCell(9);
                cell.setCellValue(report.getTimeOfEntry());
                //离职类型
                cell = dataRow.createCell(10);
                cell.setCellValue(report.getTypeOfTurnover());
                //离职原因
                cell = dataRow.createCell(11);
                cell.setCellValue(report.getReasonsForLeaving());
                //离职时间
                cell = dataRow.createCell(12);
                cell.setCellValue(report.getResignationTime());
            }
        }
        String fileName = URLEncoder.encode(month+"人员信息.xlsx", "UTF-8");
        response.setContentType("application/octet-stream");
        response.setHeader("content-disposition", "attachment;filename=" + new
String(fileName.getBytes("ISO8859-1")));
        response.setHeader("filename", fileName);
        workbook.write(response.getOutputStream());
```

4.4 对比测试

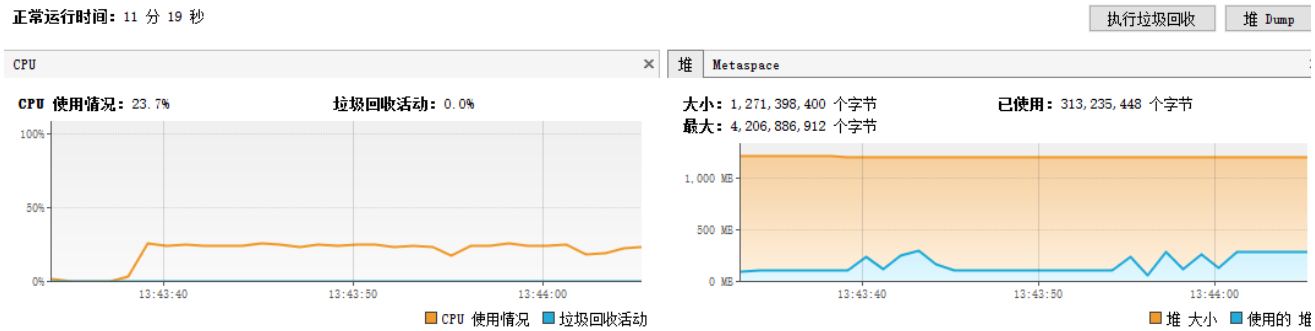
（1）XSSFWorkbook生成百万数据报表

使用XSSFWorkbook生成Excel报表，时间较长，随着时间推移，内存占用原来越多，直至内存溢出



（2）SXSSFWorkbook生成百万数据报表

使用SXSSFWorkbook生成Excel报表，内存占用比较平缓



5 百万数据报表读取

5.1 需求分析

使用POI基于事件模式解析案例提供的Excel文件

5.2 解决方案

5.2.1 思路分析

- 用户模式：加载并读取Excel时，是通过一次性的将所有数据加载到内存中再去解析每个单元格内容。当Excel数据量较大时，由于不同的运行环境可能会造成内存不足甚至OOM异常。
- 事件模式：它逐行扫描文档，一边扫描一边解析。由于应用程序只是在读取数据时检查数据，因此不需要将数据存储在内存中，这对于大型文档的解析是个巨大优势。

5.2.2 步骤分析

（1）设置POI的事件模式

- 根据Excel获取文件流
- 根据文件流创建OPCPackage

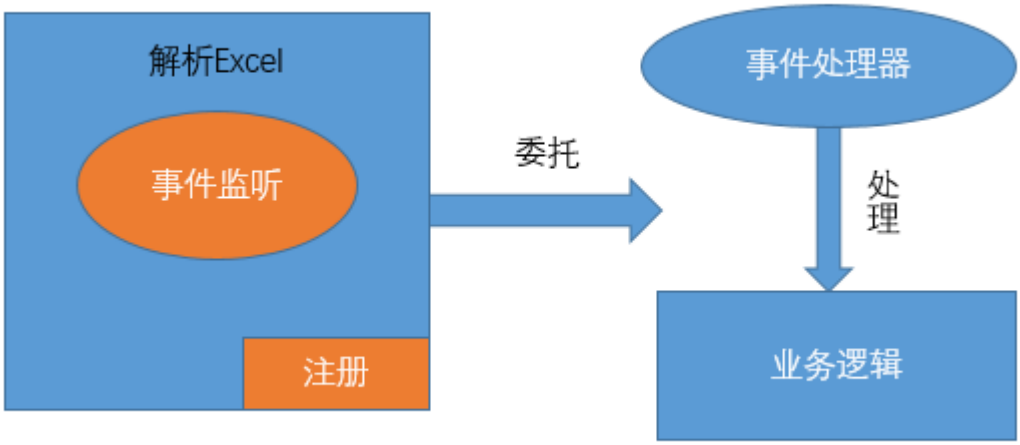
- 创建XSSFReader对象

（2）Sax解析

- 自定义Sheet处理器
- 创建Sax的XmlReader对象
- 设置Sheet的事件处理器
- 逐行读取

5.2.3 原理分析

我们都知道对于Excel2007的实质是一种特殊的XML存储数据，那就可以使用基于SAX的方式解析XML完成Excel的读取。SAX提供了一种从XML文档中读取数据的机制。它逐行扫描文档，一边扫描一边解析。由于应用程序只是在读取数据时检查数据，因此不需要将数据存储在内存中，这对于大型文档的解析是个巨大优势



5.3 代码实现

5.3.1 自定义处理器

```
//自定义Sheet基于Sax的解析处理器
public class SheetHandler implements XSSFSheetXMLHandler.SheetContentsHandler {

    //封装实体对象
    private PoiEntity entity;

    /**
     * 解析行开始
     */
    @Override
    public void startRow(int rowNum) {
        if (rowNum >0 ) {
            entity = new PoiEntity();
        }
    }

    /**
```

```
     * 解析每一个单元格
     */
    @Override
    public void cell(String cellReference, String formattedValue, XSSFComment comment)
{
        if(entity != null) {
            switch (cellReference.substring(0, 1)) {
                case "A":
                    entity.setId(formattedValue);
                    break;
                case "B":
                    entity.setBreast(formattedValue);
                    break;
                case "C":
                    entity.setAdipocytes(formattedValue);
                    break;
                case "D":
                    entity.setNegative(formattedValue);
                    break;
                case "E":
                    entity.setStaining(formattedValue);
                    break;
                case "F":
                    entity.setSupportive(formattedValue);
                    break;
                default:
                    break;
            }
        }
    }

    /**
     * 解析行结束
     */
    public void endRow(int rowNum) {
        System.out.println(entity);
    }

    //处理头尾
    public void headerFooter(String text, boolean isHeader, String tagName) {
    }
}
```

5.3.2 自定义解析

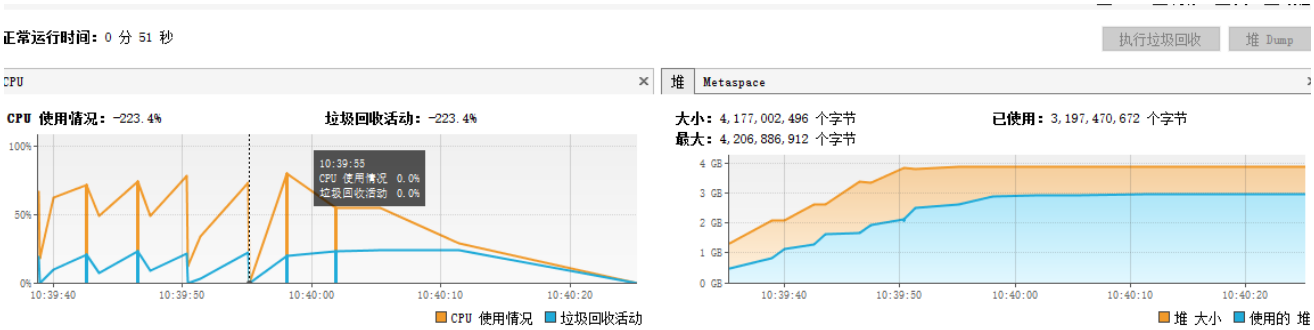
```
/**
 * 自定义Excel解析器
 */
public class ExcelParser {

    public void parse (String path) throws Exception {
        //1.根据Excel获取OPCPackage对象
```

```
        OPCPackage pkg = OPCPackage.open(path, PackageAccess.READ);
        try {
            //2.创建XSSFReader对象
            XSSFReader reader = new XSSFReader(pkg);
            //3.获取SharedStringsTable对象
            SharedStringsTable sst = reader.getSharedStringsTable();
            //4.获取StylesTable对象
            StylesTable styles = reader.getStylesTable();
            //5.创建Sax的XmlReader对象
            XMLReader parser = XMLReaderFactory.createXMLReader();
            //6.设置处理器
            parser.setContentHandler(new XSSFSheetXMLHandler(styles,sst, new
SheetHandler(), false));
            XSSFReader.SheetIterator sheets = (XSSFReader.SheetIterator)
reader.getSheetsData();
            //7.逐行读取
            while (sheets.hasNext()) {
                InputStream sheetstream = sheets.next();
                InputSource sheetSource = new InputSource(sheetstream);
                try {
                    parser.parse(sheetSource);
                } finally {
                    sheetstream.close();
                }
            }
        } finally {
            pkg.close();
        }
    }
}
```

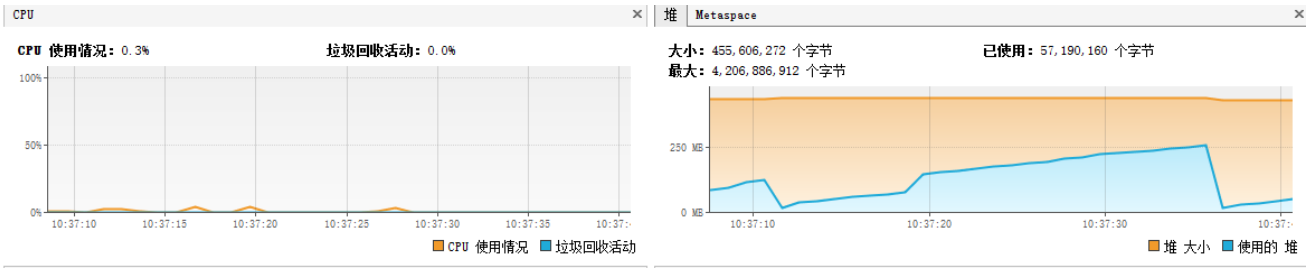
5.3.3 对比测试

用户模式下读取测试Excel文件直接内存溢出，测试Excel文件映射到内存中还是占用了不少内存；事件模式下可以流畅的运行。

（1）使用用户模型解析



（2）使用事件模型解析



5.4 总结

通过简单的分析以及运行两种模式进行比较，可以看到用户模式下使用更简单的代码实现了Excel读取，但是在读取大文件时CPU和内存都不理想；而事件模式虽然代码写起来比较繁琐，但是在读取大文件时CPU和内存更加占优。