源码跟踪

1. header标题初始化的源码跟踪比较复杂，因为原本有很多builder，而且实际读写设置实体类参数都是继承同一个`Param`类的，所以下面只取读的时候初始化场景，可能有误；

2. 从sheet方法进入

```
@Test
public void tesStudent(){
    EasyExcel.read( pathName: "C:\\Users\\4\\Desktop\\学生信息导入模板.xlsx", StudentInfo.class,
        new StudentInfoListener()).sheet( sheetNo: 0)
        .headRowNumber(3).doRead();
}
```

```
public ExcelReaderSheetBuilder sheet(Integer sheetNo, String sheetName) {
    ExcelReaderSheetBuilder excelReaderSheetBuilder = new ExcelReaderSheetBuilder(build());
    if (sheetNo != null) {
        excelReaderSheetBuilder.sheetNo(sheetNo);
    }
    if (sheetName != null) {
        excelReaderSheetBuilder.sheetName(sheetName);
    }
    return excelReaderSheetBuilder;
}
```

```
public ExcelReader build() {
    return new ExcelReader(readWorkbook);
}
```

```
public ExcelAnalyserImpl(ReadWorkbook readWorkbook) {
    try {
        choiceExcelExecutor(readWorkbook);
    } catch (RuntimeException e) {
        finish();
        throw e;
    } catch (Throwable e) {
        finish();
        throw new ExcelAnalysisException(e);
    }
}
```

```java
switch (excelType) {
    case XLS:
        POIFSFileSystem poifsFileSystem;
        if (readWorkbook.getFile() != null) {
            poifsFileSystem = new POIFSFileSystem(readWorkbook.getFile());
        } else {
            poifsFileSystem = new POIFSFileSystem(readWorkbook.getInputStream());
        }
        // So in encrypted excel, it looks like XLS but it's actually XLSX
        if (poifsFileSystem.getRoot().hasEntry(Decryptor.DEFAULT_POIFS_ENTRY)) {
            InputStream decryptedStream = null;
            try {
                decryptedStream = DocumentFactoryHelper
                    .getDecryptedStream(poifsFileSystem.getRoot().getFileSystem(), readWorkbook.getPassword());
                XlsxReadContext xlsxReadContext = new DefaultXlsxReadContext(readWorkbook, ExcelTypeEnum.XLSX);
                analysisContext = xlsxReadContext;
                excelReadExecutor = new XlsxSaxAnalyser(xlsxReadContext, decryptedStream);
                return;
            } finally {
                IOUtils.closeQuietly(decryptedStream);
                // as we processed the full stream already, we can close the filesystem here
                // otherwise file handles are leaked
                poifsFileSystem.close();
            }
        }
        if (readWorkbook.getPassword() != null) {
            Biff8EncryptionKey.setCurrentUserPassword(readWorkbook.getPassword());
        }
        XlsReadContext xlsReadContext = new DefaultXlsReadContext(readWorkbook, ExcelTypeEnum.XLS);
        xlsReadContext.xlsReadWorkbookHolder().setPoifsFileSystem(poifsFileSystem);
        analysisContext = xlsReadContext;
        excelReadExecutor = new XlsSaxAnalyser(xlsReadContext);
        break;
    case XLSX:
        XlsxReadContext xlsxReadContext = new DefaultXlsxReadContext(readWorkbook, ExcelTypeEnum.XLSX);
        analysisContext = xlsxReadContext;
        excelReadExecutor = new XlsxSaxAnalyser(xlsxReadContext,  decryptedStream: null);
        break;
    default:
```

```java
public AnalysisContextImpl(ReadWorkbook readWorkbook, ExcelTypeEnum actualExcelType) {
    if (readWorkbook == null) {
        throw new IllegalArgumentException("Workbook argument cannot be null");
    }
    switch (actualExcelType) {
        case XLS:
            readWorkbookHolder = new XlsReadWorkbookHolder(readWorkbook);
            break;
        case XLSX:
            readWorkbookHolder = new XlsxReadWorkbookHolder(readWorkbook);
            break;
        default:
            break;
    }
    currentReadHolder = readWorkbookHolder;
    analysisEventProcessor = new DefaultAnalysisEventProcessor();
    if (LOGGER.isDebugEnabled()) {
        LOGGER.debug("Initialization 'AnalysisContextImpl' complete");
    }
}
```

```java
public XlsxReadWorkbookHolder(ReadWorkbook readWorkbook) {
    super(readWorkbook);
    this.saxParserFactoryName = readWorkbook.getXlsxSAXParserFactoryName();
    setExcelType(ExcelTypeEnum.XLSX);
}
```

```
public ReadWorkbookHolder(ReadWorkbook readWorkbook) {
    super(readWorkbook,  parentAbstractReadHolder: null, readWorkbook.getConvertAllFiled());
    this.readWorkbook = readWorkbook;
    if (readWorkbook.getInputStream() != null) {
        this.inputStream = readWorkbook.getInputStream();
    }
    this.file = readWorkbook.getFile();
    if (readWorkbook.getMandatoryUseInputStream() == null) {
        this.mandatoryUseInputStream = Boolean.FALSE;
    } else {
        this.mandatoryUseInputStream = readWorkbook.getMandatoryUseInputStream();
    }
    if (readWorkbook.getAutoCloseStream() == null) {
        this.autoCloseStream = Boolean.TRUE;
```

```
public AbstractReadHolder(ReadBasicParameter readBasicParameter, AbstractReadHolder parentAbstractReadHolder,
    Boolean convertAllFiled) {
    super(readBasicParameter, parentAbstractReadHolder);
    if (readBasicParameter.getUse1904windowing() == null && parentAbstractReadHolder != null) {
        getGlobalConfiguration()
            .setUse1904windowing(parentAbstractReadHolder.getGlobalConfiguration().getUse1904windowing());
    } else {
        getGlobalConfiguration().setUse1904windowing(readBasicParameter.getUse1904windowing());
    }

    if (readBasicParameter.getUseScientificFormat() == null) {
        if (parentAbstractReadHolder == null) {
            getGlobalConfiguration().setUseScientificFormat(Boolean.FALSE);
        } else {
            getGlobalConfiguration()
                .setUseScientificFormat(parentAbstractReadHolder.getGlobalConfiguration().getUseScientificFormat());
        }
    } else {
        getGlobalConfiguration().setUseScientificFormat(readBasicParameter.getUseScientificFormat());
    }

    // Initialization property
    this.excelReadHeadProperty = new ExcelReadHeadProperty( holder: this, getClazz(), getHead(), convertAllFiled);
    if (readBasicParameter.getHeadRowNumber() == null) {
        if (parentAbstractReadHolder == null) {
            if (excelReadHeadProperty.hasHead()) {
                this.headRowNumber = excelReadHeadProperty.getHeadRowNumber();
            } else {
                this.headRowNumber = 1;
            }
        } else {
            this.headRowNumber = parentAbstractReadHolder.getHeadRowNumber();
        }
    } else {
        this.headRowNumber = readBasicParameter.getHeadRowNumber();
    }
```

直到这里，我们能找到`ExcelReadHeadProperty`初始化的代码，因为这个类的实例就是程序匹配excel标题时用到的。

3. 进入到`ExcelHeadProperty`类的构造方法（读写都是继承该类的）

```java
public ExcelHeadProperty(Holder holder, Class headClazz, List<List<String>> head, Boolean convertAllFiled) {
    this.headClazz = headClazz;
    headMap = new TreeMap<Integer, Head>();
    contentPropertyMap = new TreeMap<Integer, ExcelContentProperty>();
    fieldNameContentPropertyMap = new HashMap<String, ExcelContentProperty>();
    ignoreMap = new HashMap<String, Field>( initialCapacity: 16);
    headKind = HeadKindEnum.NONE;
    headRowNumber = 0;
    if (head != null && !head.isEmpty()) {
        int headIndex = 0;
        for (int i = 0; i < head.size(); i++) {
            if (holder instanceof AbstractWriteHolder) {
                if (((AbstractWriteHolder) holder).ignore( fieldName: null, i)) {
                    continue;
                }
            }
            headMap.put(headIndex, new Head(headIndex,  fieldName: null, head.get(i), Boolean.FALSE, Boolean.TRUE));
            contentPropertyMap.put(headIndex, null);
            headIndex++;
        }
        headKind = HeadKindEnum.STRING;
    }
    // convert headClazz to head
    initColumnProperties(holder, convertAllFiled);

    initHeadRowNumber();
    if (LOGGER.isDebugEnabled()) {
        LOGGER.debug("The initialization sheet/table 'ExcelHeadProperty' is complete , head kind is {}", headKind);
    }
}
```

注意这里`headKind`，如果你是通过字符串数组的方式设置header（标题）的话，则这里的headKind就是`HeadKindEnum.STRING`；如果是通过实体类，则走`initColumnProperties(holder, convertAllFiled);`方法

4. 看具体如何解析实体类（`@ExcelProperty`注解）

```java
private void initColumnProperties(Holder holder, Boolean convertAllFiled) {
    if (headClazz == null) {
        return;
    }
    // Declared fields
    Map<Integer, Field> sortedAllFiledMap = new TreeMap<~>();
    Map<Integer, Field> indexFiledMap = new TreeMap<~>();

    boolean needIgnore = (holder instanceof AbstractWriteHolder) && (
        !CollectionUtils.isEmpty(((AbstractWriteHolder) holder).getExcludeColumnFiledNames()) || !CollectionUtils
            .isEmpty(((AbstractWriteHolder) holder).getExcludeColumnIndexes()) || !CollectionUtils
            .isEmpty(((AbstractWriteHolder) holder).getIncludeColumnFiledNames()) || !CollectionUtils
            .isEmpty(((AbstractWriteHolder) holder).getIncludeColumnIndexes()));
    ClassUtils.declaredFields(headClazz, sortedAllFiledMap, indexFiledMap, ignoreMap, convertAllFiled, needIgnore,
        holder);

    for (Map.Entry<Integer, Field> entry : sortedAllFiledMap.entrySet()) {
        initOneColumnProperty(entry.getKey(), entry.getValue(), indexFiledMap.containsKey(entry.getKey()));
    }
    headKind = HeadKindEnum.CLASS;
}
```

主要看这两个方法，首先说明一下参数：

- `sortedAllFiledMap`：是最终排序好的标题与实体类中`File`对应；
- `indexFiledMap`：是注解中加了index属性的File集合；
- `ignoreMap`：应该是`@ExcelIgnore`标签的File集合（没具体研究了）。

以上参数都在`ClassUtils.declaredFields`方法中解析后赋值；

主要看`ClassUtils.declaredFields`方法，主要跟踪Clazz对象：

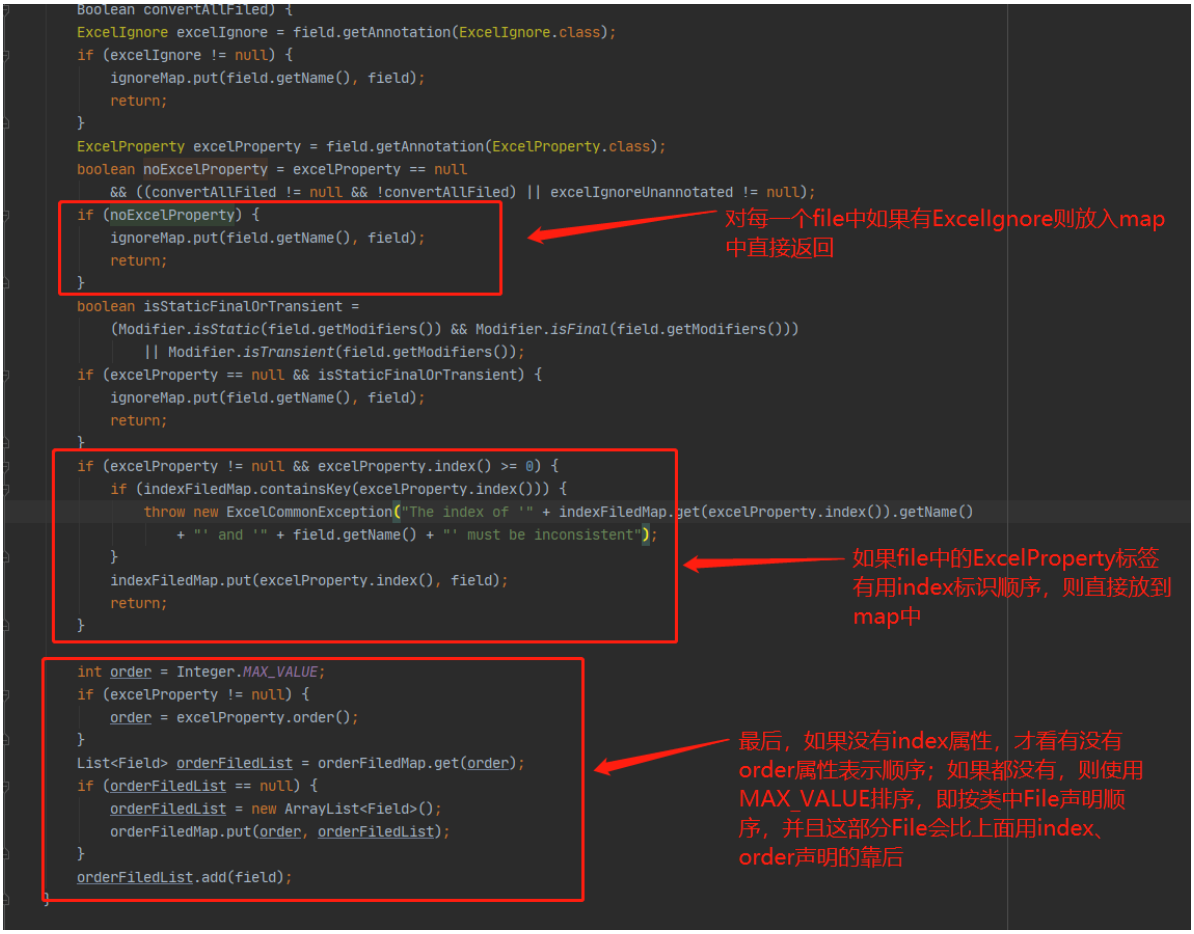
```java
public static void declaredFields(Class clazz, Map<Integer, Field> sortedAllFiledMap,
    Map<Integer, Field> indexFiledMap, Map<String, Field> ignoreMap, Boolean convertAllFiled,
    Boolean needIgnore, Holder holder) {
    FieldCache fieldCache = getFieldCache(clazz, convertAllFiled);
    if (fieldCache == null) {
        return;
    }
    if (ignoreMap != null) {
        ignoreMap.putAll(fieldCache.getIgnoreMap());
    }
    Map<Integer, Field> tempIndexFildMap = indexFiledMap;
    if (tempIndexFildMap == null) {
        tempIndexFildMap = new TreeMap<Integer, Field>();
    }
    tempIndexFildMap.putAll(fieldCache.getIndexFiledMap());
```

```java
private static FieldCache getFieldCache(Class clazz, Boolean convertAllFiled) {
    if (clazz == null) {
        return null;
    }
    SoftReference<FieldCache> fieldCacheSoftReference = FIELD_CACHE.get(clazz);
    if (fieldCacheSoftReference != null && fieldCacheSoftReference.get() != null) {
        return fieldCacheSoftReference.get();
    }
    synchronized (clazz) {
        fieldCacheSoftReference = FIELD_CACHE.get(clazz);
        if (fieldCacheSoftReference != null && fieldCacheSoftReference.get() != null) {
            return fieldCacheSoftReference.get();
        }
        declaredFields(clazz, convertAllFiled);
    }
    return FIELD_CACHE.get(clazz).get();
}
```

```java
private static void declaredFields(Class clazz, Boolean convertAllFiled) {
    List<Field> tempFieldList = new ArrayList<~>();
    Class tempClass = clazz;
    // When the parent class is null, it indicates that the parent class (Object class) has reached the top
    // level.
    while (tempClass != null && tempClass != BaseRowModel.class) {
        Collections.addAll(tempFieldList, tempClass.getDeclaredFields());
        // Get the parent class and give it to yourself
        tempClass = tempClass.getSuperclass();
    }
    // Screening of field
    Map<Integer, List<Field>> orderFiledMap = new TreeMap<~>();
    Map<Integer, Field> indexFiledMap = new TreeMap<~>();
    Map<String, Field> ignoreMap = new HashMap<~>( initialCapacity: 16);

    ExcelIgnoreUnannotated excelIgnoreUnannotated =
        (ExcelIgnoreUnannotated) clazz.getAnnotation(ExcelIgnoreUnannotated.class);
    for (Field field : tempFieldList) {
        declaredOneField(field, orderFiledMap, indexFiledMap, ignoreMap, excelIgnoreUnannotated, convertAllFiled);
    }
    FIELD_CACHE.put(clazz, new SoftReference<FieldCache>(
        new FieldCache(buildSortedAllFiledMap(orderFiledMap, indexFiledMap), indexFiledMap, ignoreMap)));
}
```

5. 上面遍历其导入类中File，调用`declaredOneField`方法，解析导入类中easyexcel注解内容。



6. 解析完之后会进行对上面map进行排序，即对标题进行排序。

```
private static void declaredFields(Class clazz, Boolean convertAllFiled) {
    List<Field> tempFieldList = new ArrayList<~>();
    Class tempClass = clazz;
    // When the parent class is null, it indicates that the parent class (Object class) has reached the top
    // level.
    while (tempClass != null && tempClass != BaseRowModel.class) {
        Collections.addAll(tempFieldList, tempClass.getDeclaredFields());
        // Get the parent class and give it to yourself
        tempClass = tempClass.getSuperclass();
    }
    // Screening of field
    Map<Integer, List<Field>> orderFiledMap = new TreeMap<~>();
    Map<Integer, Field> indexFiledMap = new TreeMap<~>();
    Map<String, Field> ignoreMap = new HashMap<~>( initialCapacity: 16);

    ExcelIgnoreUnannotated excelIgnoreUnannotated =
        (ExcelIgnoreUnannotated) clazz.getAnnotation(ExcelIgnoreUnannotated.class);
    for (Field field : tempFieldList) {
        declaredOneField(field, orderFiledMap, indexFiledMap, ignoreMap, excelIgnoreUnannotated, convertAllFiled);
    }
    FIELD_CACHE.put(clazz, new SoftReference<FieldCache>(
        new FieldCache(buildSortedAllFiledMap(orderFiledMap, indexFiledMap), indexFiledMap, ignoreMap)));
}
```

7. 由上面源码可知，正如官网介绍的，`@ExcelProperty`标签中index和order是不能同时定义，index属性会把order属性无效化。