**excel读取耗时问题优化分析**

目录

**[1 流程分析 1](#_Toc28921)**

[1.1 解析excel整体流程 1](#_Toc14649)

**[2 数据记录 3](#_Toc13505)**

[2.1 未优化数据记录 3](#_Toc22594)

[2.2 对createApacheWorkbook进行优化 4](#_Toc12967)

[2.3 对readSheet优化1 4](#_Toc32560)

[2.4 对readSheet优化2 8](#_Toc18202)

[2.5 setRefObjects优化 8](#_Toc25786)

[2.6 整体优化过后，虚拟机耗时记录 10](#_Toc29174)

**[3 后续 10](#_Toc1372)**

[3.1 为什么setCellType这么耗时 10](#_Toc4489)

**[4 后续问题 13](#_Toc29390)**

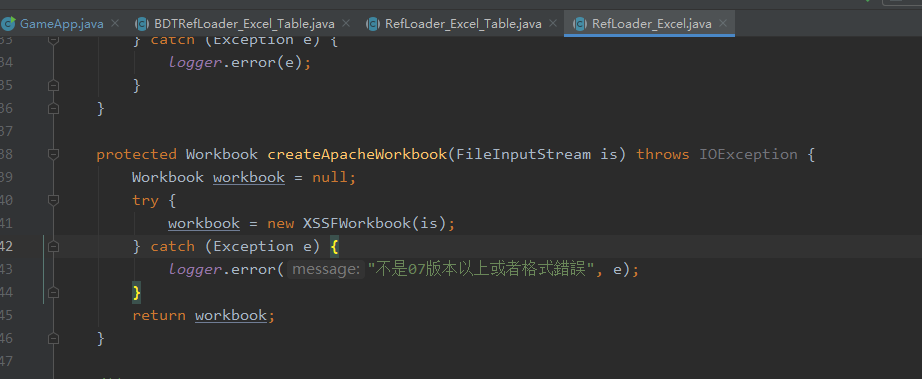
[4.1 如果让修改方案安全落地 13](#_Toc10198)

**[5 参考资料 13](#_Toc18518)**

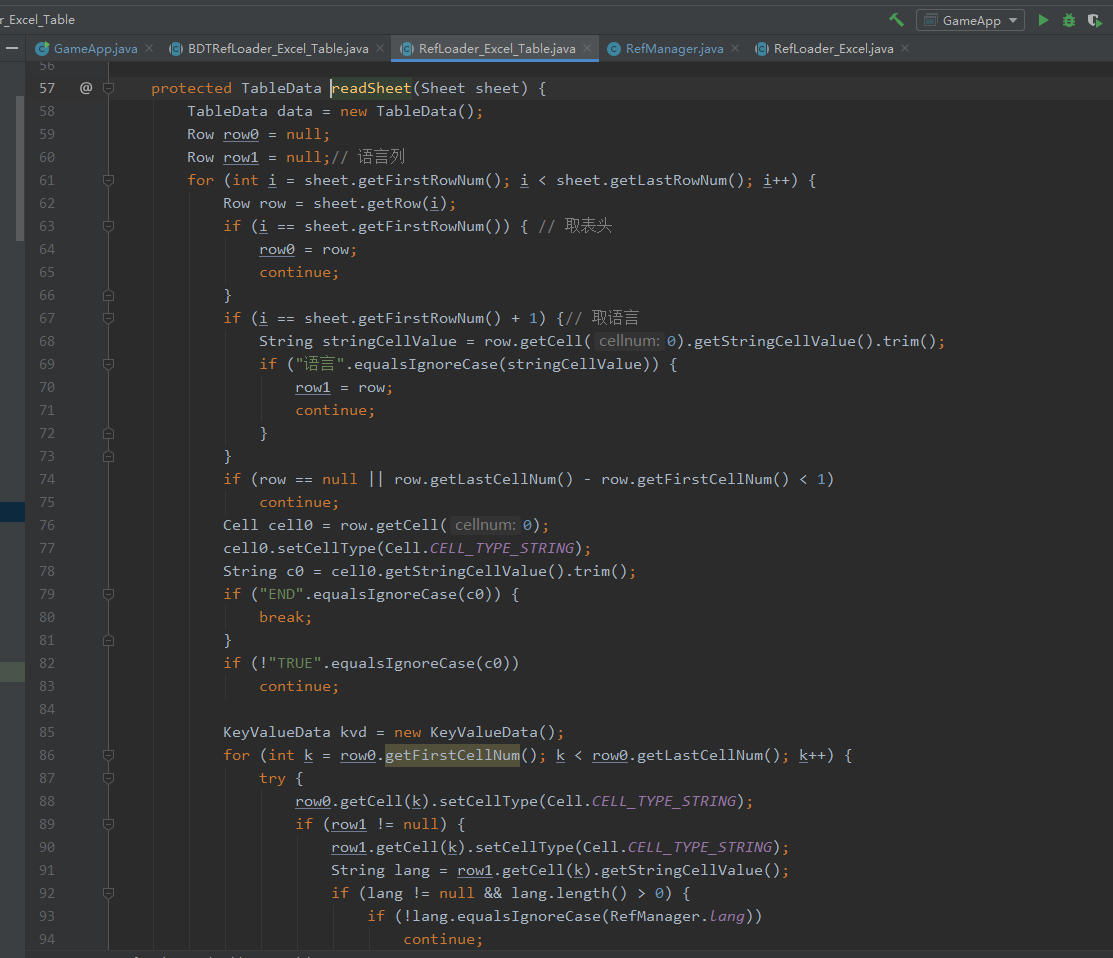
# 流程分析

## 解析excel整体流程

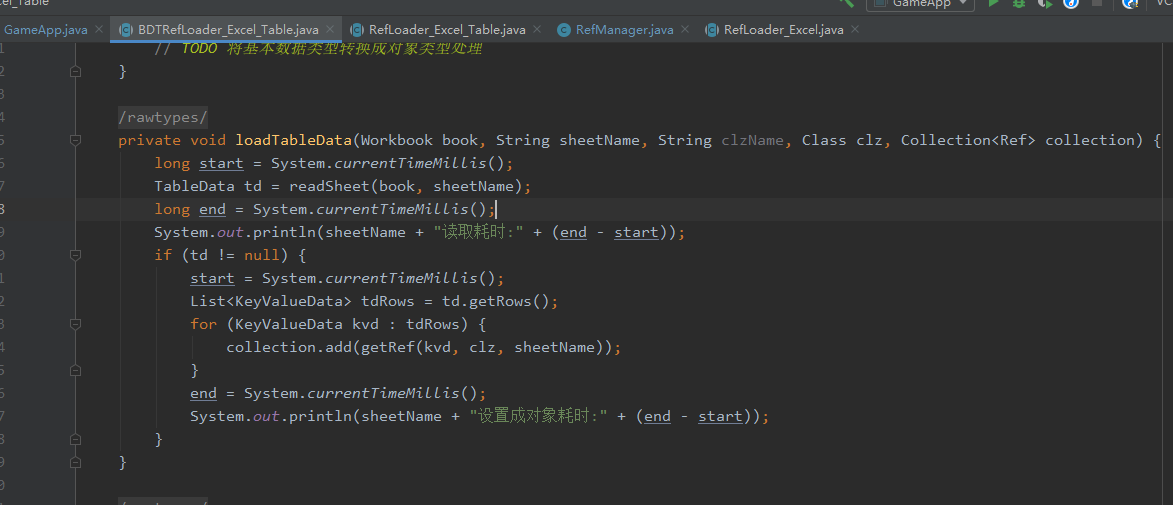
* 将excel文件解析为java对象



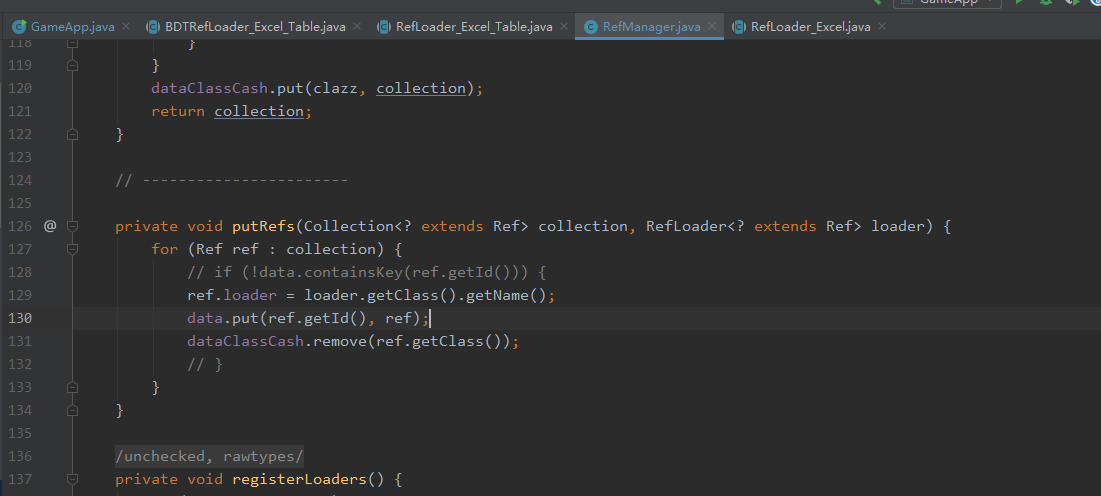
* 将工作薄的数据转换为 TableData



* 将TableData里面rowsData集合转换为对应Ref实例对象集合



* 转换的Ref实例对象集合所有Ref对象以refId为key放入RefManager的data里面



# 数据记录

## 未优化数据记录

* 本机未处理任何优化

|  |  |  |
| --- | --- | --- |
| 说明 | 总耗时(ms) | 细分统计(ms) |
| 第1次 | 44126 | readSheet:24718  createApacheWorkbook:13277  setRefObjects:3485  sumTime:41481 |
| 第2次 | 45101 | readSheet:25867  createApacheWorkbook:13183  setRefObjects:3418  sumTime:42469 |
| 第3次 | 45437 | readSheet:25902  createApacheWorkbook:13333  setRefObjects:3473  sumTime:42709 |
| 平均 | 44888(接近1分钟) |  |

* 虚拟机未处理任何优化

|  |  |  |
| --- | --- | --- |
| 说明 | 总耗时(ms) | 细分统计(ms) |
| 第1次 | 157702 | readSheet:96705  createApacheWorkbook:43113  setRefObjects:10987  sumTime:150806 |
| 第2次 | 161143 | readSheet:96154  createApacheWorkbook:46813  setRefObjects:11252  sumTime:154221 |
| 第3次 | 164230 | readSheet:101902  createApacheWorkbook:46197  setRefObjects:10269  sumTime:158369 |
| 平均 | 161025(接近3分钟) |  |

## 对createApacheWorkbook进行优化

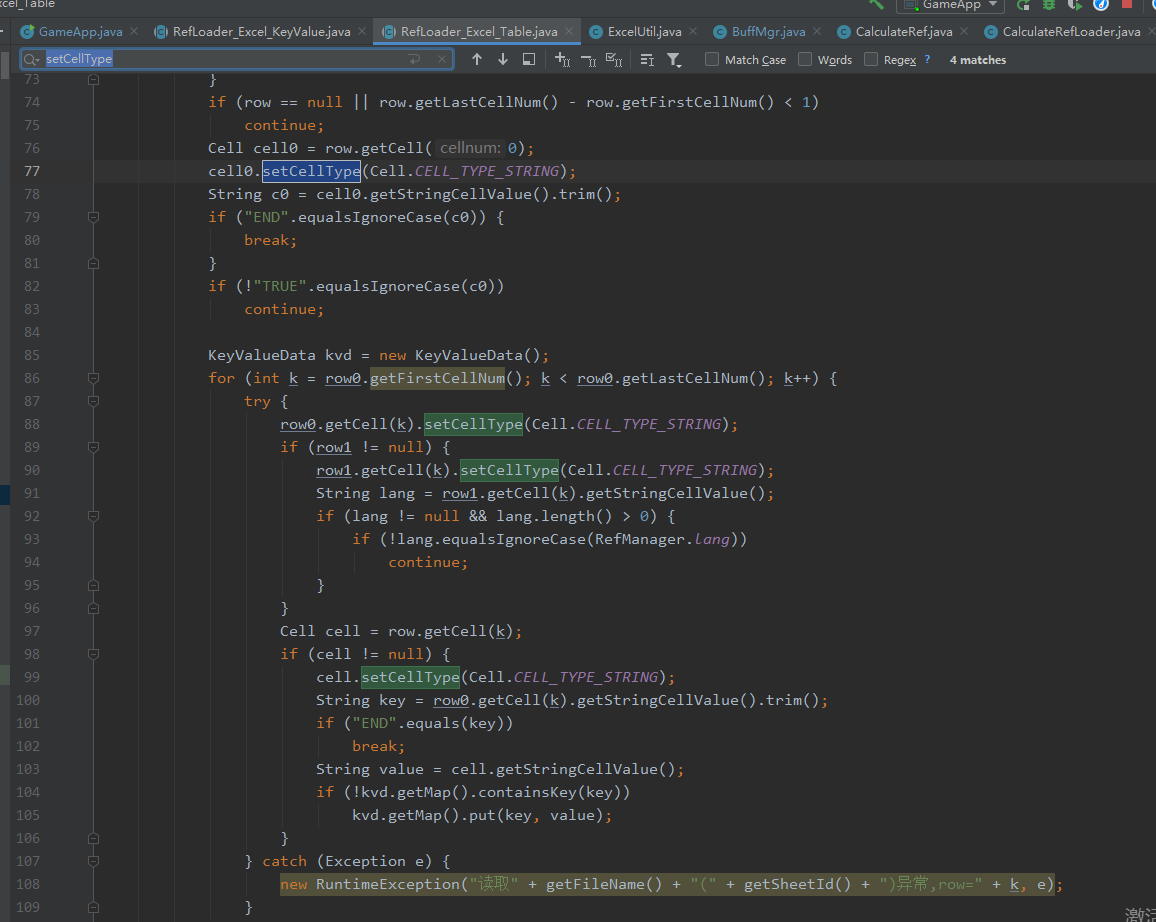
* 优化思路，由于本身需要读取全部文件内容
* 直接把文件内容读成字节
* XSSFWorkbook的InputStream 采用 ByteArrayInputStream
* 不采用FileInputStream

|  |  |  |
| --- | --- | --- |
| 说明 | 总耗时(ms) | 细分统计(ms) |
| 第1次 | 44274 | readSheet:25080  createApacheWorkbook:12997  setRefObjects:3683  sumTime:41760 |
| 第2次 | 44854 | readSheet:25613  createApacheWorkbook:13149  setRefObjects:3549  sumTime:42312 |
| 第3次 | 44695 | readSheet:23875  createApacheWorkbook:13366  setRefObjects:4977  sumTime:42220 |
| 平均 | 44607(接近1分钟) |  |

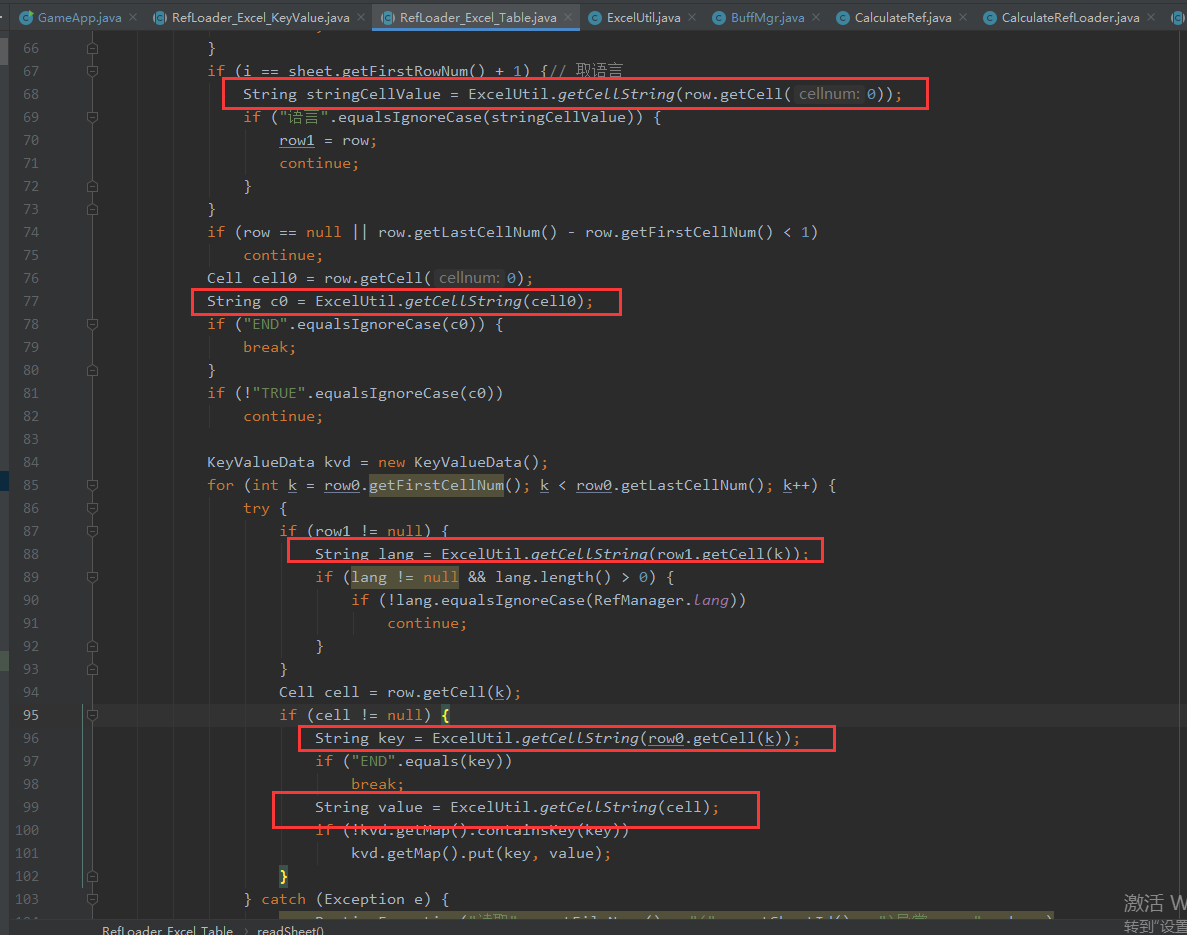
* 无太大改进，createApacheWorkbook基本是浮动

## 对readSheet优化1

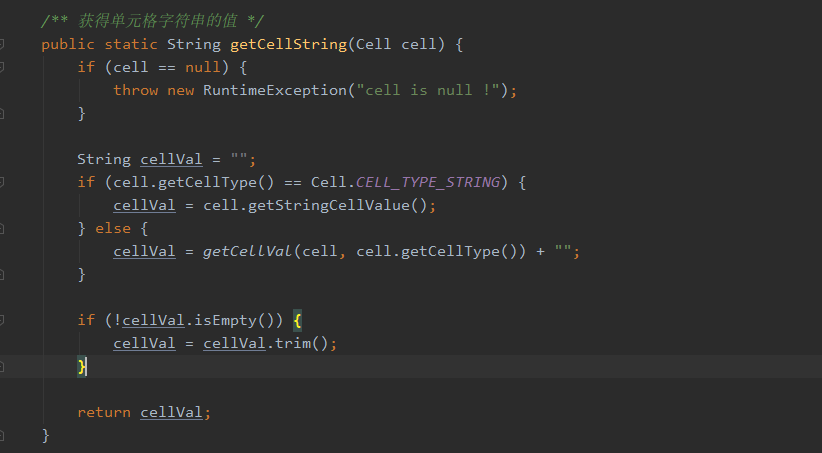
* 志强对readSheet函数进行详细打点
* 发现Cell.setCellType在虚拟机比本机耗时慢非常多
* 有时甚至达到10倍以上，将setCellType去掉，获得string的值改成
* 改之前代码



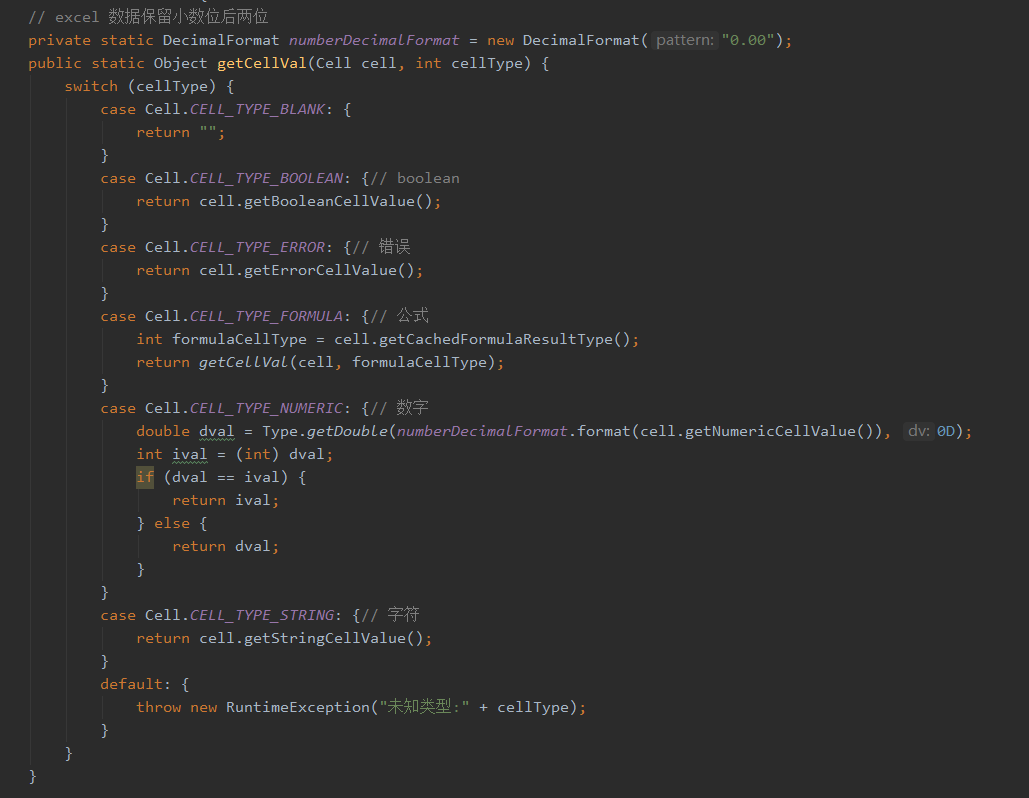
* 改之后代码



* ExcelUtil.getCellString代码



* getCellVal函数代码如下



* 优化后本机耗时记录

|  |  |  |
| --- | --- | --- |
| 说明 | 总耗时(ms) | 细分统计(ms) |
| 第1次 | 24336 | readSheet:4732  createApacheWorkbook:13848  setRefObjects:3338  sumTime:21919 |
| 第2次 | 23898 | readSheet:4423  createApacheWorkbook:13066  setRefObjects:3577  sumTime:21067 |
| 第3次 | 23870 | readSheet:4493  createApacheWorkbook:13199  setRefObjects:3596  sumTime:21288 |
| 平均 | 24034(接近半分钟) |  |

* 可以看到读取耗时减少了一半

## 对readSheet优化2

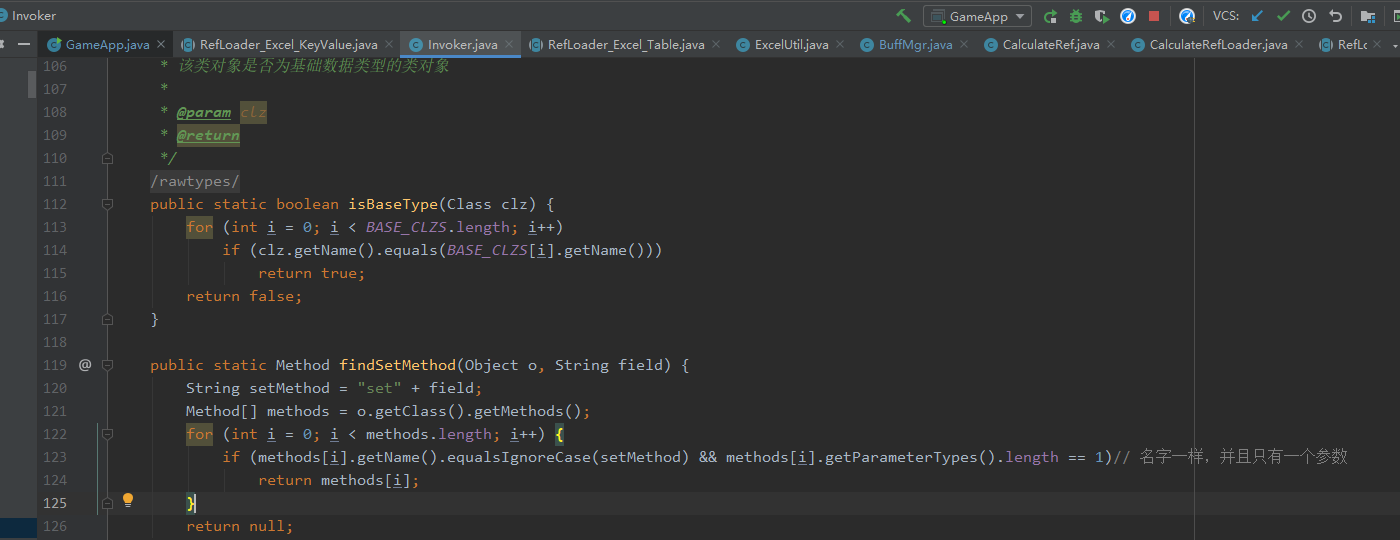
* readSheet代码还有多次读取表头问题，改成统一表头读取
* 本机耗时记录

|  |  |  |
| --- | --- | --- |
| 说明 | 总耗时(ms) | 细分统计(ms) |
| 第1次 | 22013 | readSheet:2386  createApacheWorkbook:13151  setRefObjects:3375  sumTime:18913 |
| 第2次 | 21583 | readSheet:2276  createApacheWorkbook:13276  setRefObjects:3748  sumTime:19301 |
| 第3次 | 21639 | readSheet:2278  createApacheWorkbook:13037  setRefObjects:3519  sumTime:18835 |
| 平均 | 21745(接近半分钟) |  |

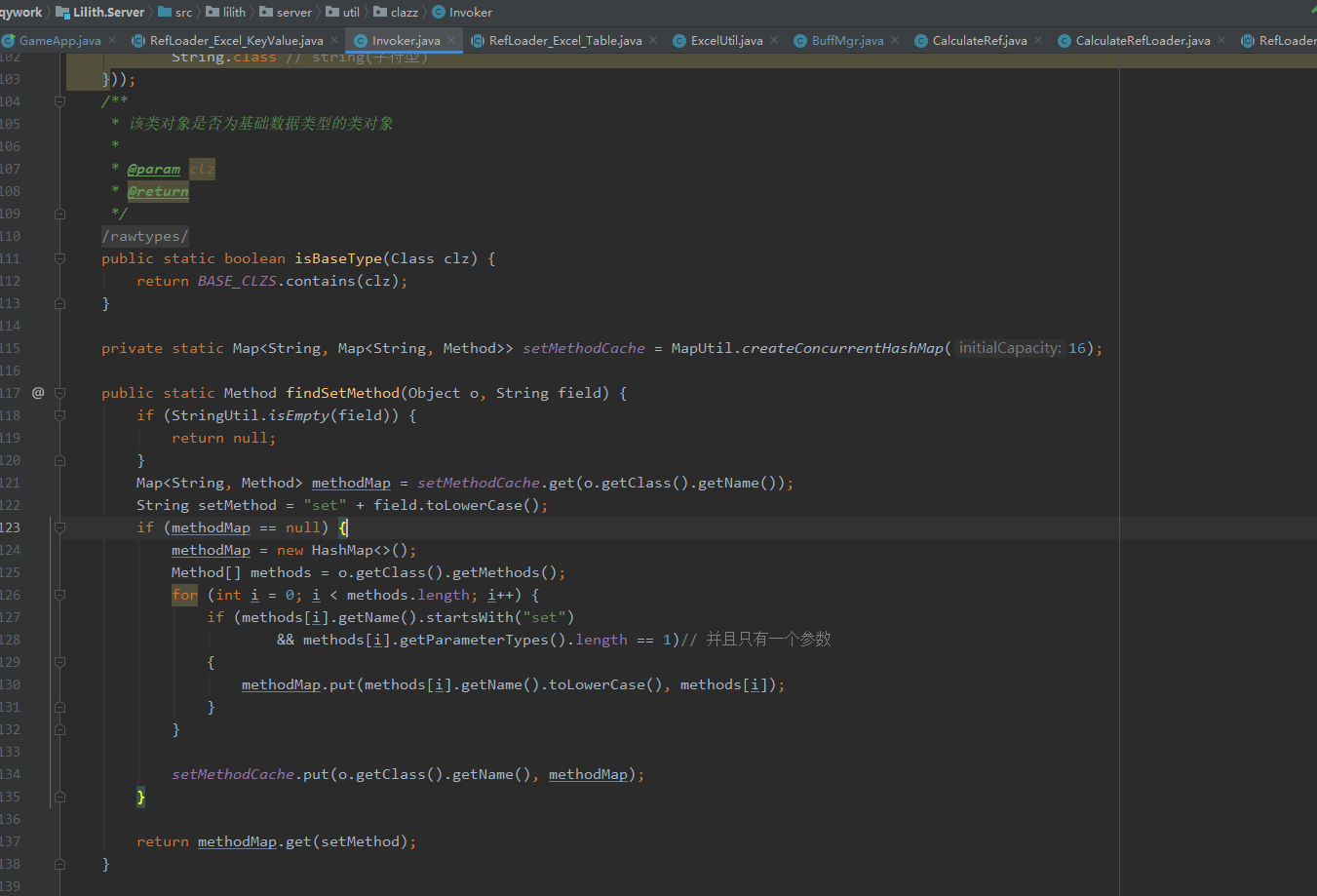
* 可以看到相比之前减少了3秒的耗时

## setRefObjects优化

* setRefObjects实际调用的是Invoker.setArgs
* 该函数的逻辑是将map的key、val数据转换成一个ref数据bean对象
* map里面的key对应bean里面的字段，val就是对应字段的值
* 通过反射执行对应字段的set函数进行设置
* 在进行查找对应set函数，采用遍历的方式查找，导致性能差，代码如下



* 还有isBaseType也存在同样的问题，采用遍历的方式判定
* 对以上两点进行优化
* 对ref bean构建方法缓存，key为函数名字，val为对应的函数
* IsBaseType修改数据结构，改成HashSet
* 更改之后的代码



* 本机耗时记录

|  |  |  |
| --- | --- | --- |
| 说明 | 总耗时(ms) | 细分统计(ms) |
| 第1次 | 19290 | readSheet:2361  createApacheWorkbook:13285  setRefObjects:1398  sumTime:17045 |
| 第2次 | 19724 | readSheet:2426  createApacheWorkbook:13658  setRefObjects:1277  sumTime:17362 |
| 第3次 | 19701 | readSheet:2434  createApacheWorkbook:13567  setRefObjects:1394  sumTime:17396 |
| 平均 | 19571(不到20秒) |  |

* 可以看到相比之前平均减少了2秒的耗时

## 整体优化过后，虚拟机耗时记录

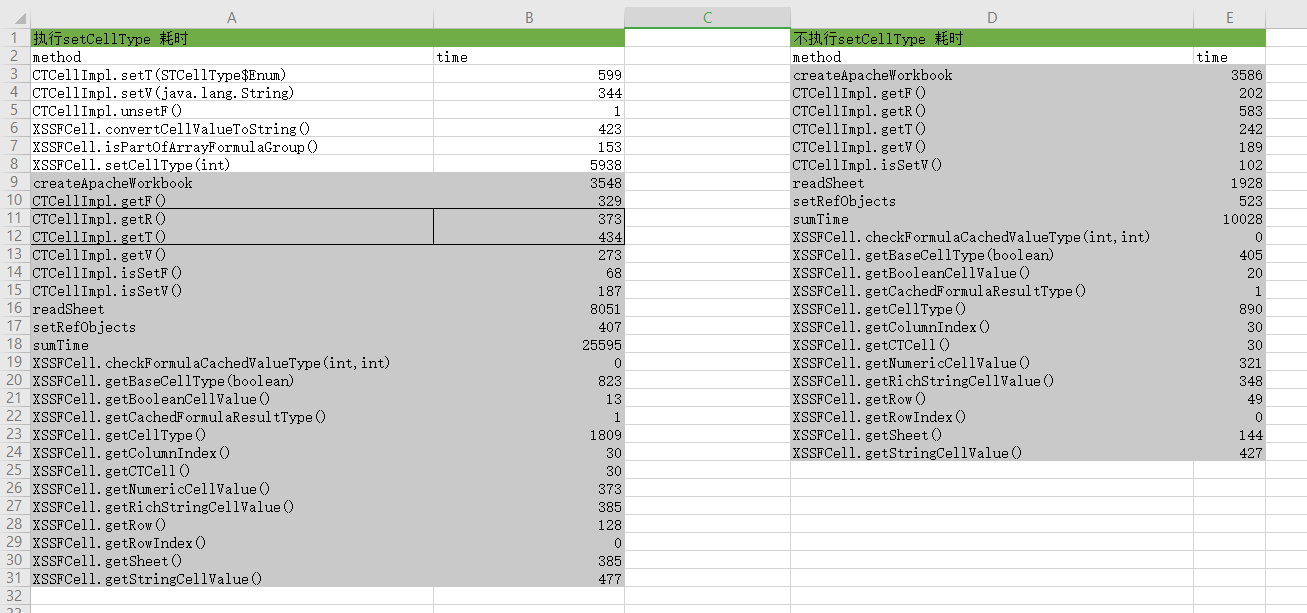
|  |  |  |
| --- | --- | --- |
| 说明 | 总耗时(ms) | 细分统计(ms) |
| 第1次 | 73509 | readSheet:9535  createApacheWorkbook:50953  setRefObjects:5790  sumTime:66279 |
| 第2次 | 81370 | readSheet:11831  createApacheWorkbook:56094  setRefObjects:6287  sumTime:74213 |
| 第3次 | 78407 | readSheet:11623  createApacheWorkbook:52878  setRefObjects:6075  sumTime:70577 |
| 平均 | 77762(1分10多秒) |  |

* 相比最初耗时(161025ms)减少了一半多

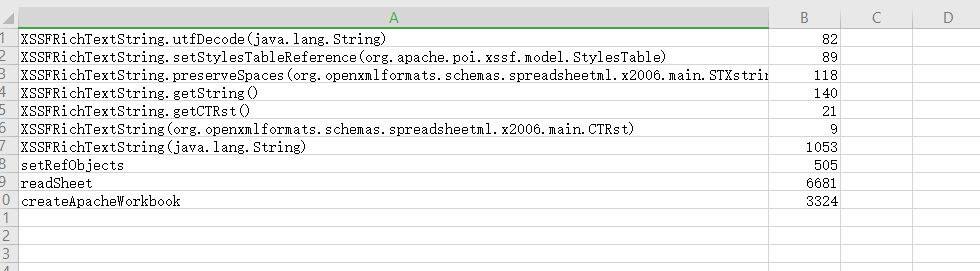
# 后续

## 为什么setCellType这么耗时

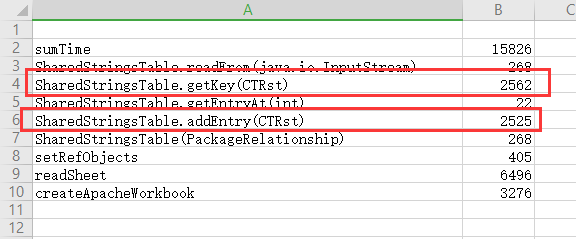
* setCellType里面是jar，无法进行代码埋点
* 翻看XSSFCell.setCellType源码
* 
* 有如下可能耗时的操作
* new XSSFRichTextString
* 执行XSSFRichTextString.setStylesTableReference
* 执行SharedStringsTable.addEntry
* 对XSSFRichTextString类方法耗时埋点
* \_cell.setV/\_cell.setT ,\_cell 的实现类为CTCellImpl
* 首先想到采用btrace进行对XSSFCell/CTCellImpl类方法埋点分析
* Btrace由于未知原因，自己编写的可以，CTCellImpl无法触发
* 所以采用自己实现java agent
* 埋点setCellType执行/不执行对比



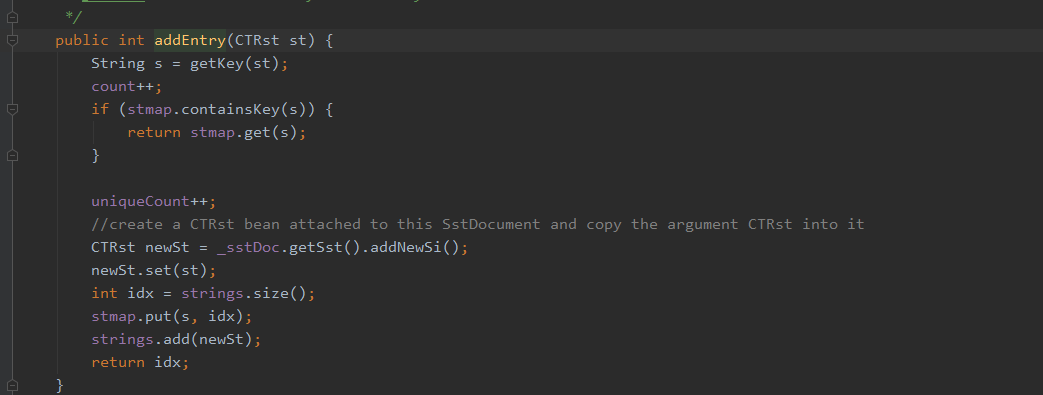
* 可以发现执行setCellType多出来的耗时，只有1500
* 但是setCellType总共耗时接近了6000
* 对XSSFRichTextString类方法埋点分析



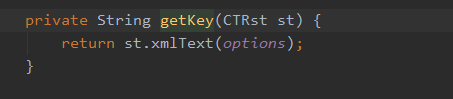
* 对SharedStringsTable类方法耗时埋点



* addEntry源码如下



* addEntry会调用getKey，getKey源码如下



* 将CTRst转换成xml字符串
* 由于每次都将CTRst对象转换成string耗时过长
* 导致setCellType耗时过长

# 后续问题

## 如果让修改方案安全落地

* 由于修改的逻辑过于通用，无法保证语义的一致性
* 需要一个方案来让该修改安全落地

# 参考资料

[手把手教你实现一个方法耗时统计的java agent](https://liuyueyi.github.io/hexblog/2020/03/16/200316-%E6%89%8B%E6%8A%8A%E6%89%8B%E6%95%99%E4%BD%A0%E5%AE%9E%E7%8E%B0%E4%B8%80%E4%B8%AA%E6%96%B9%E6%B3%95%E8%80%97%E6%97%B6%E7%BB%9F%E8%AE%A1%E7%9A%84java-agent/)

[BTrace 问题辅助排查工具使用手册](https://github.com/btraceio/btrace/wiki)