百万级报表数据前端导出优化

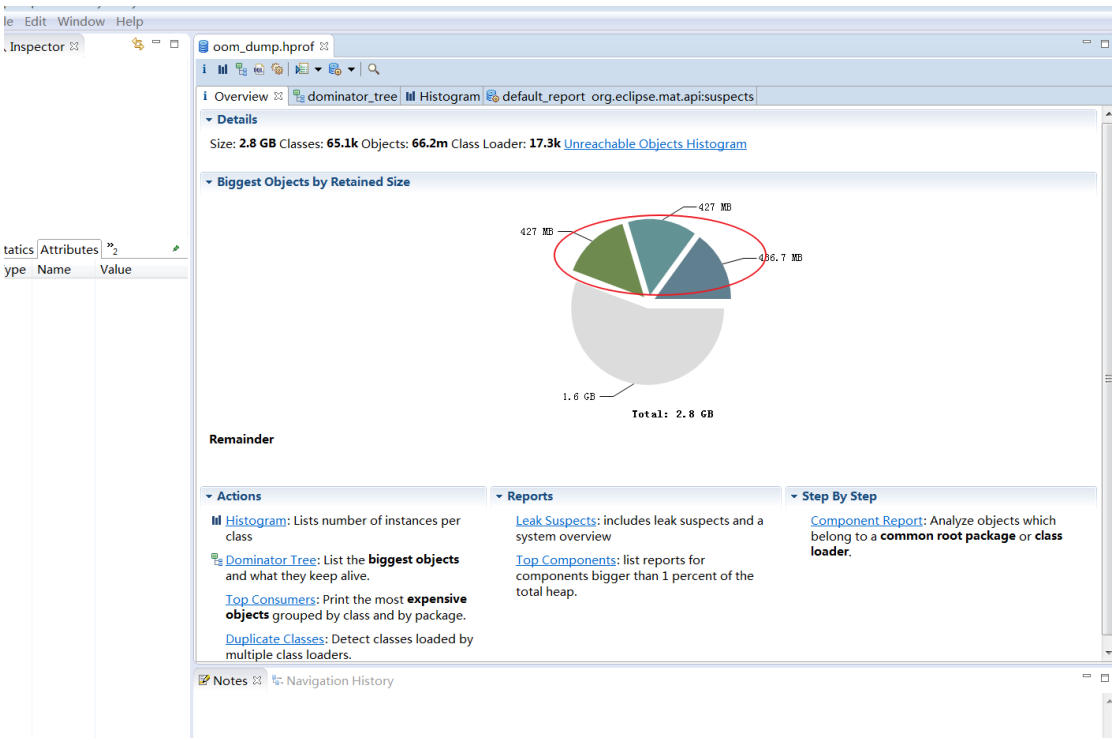
1. 发现问题

京博控股工业园智慧岗亭联网项目，现场(李卫卫 5)反馈大概半个月到一个月平台崩溃一次，重启后会恢复，严重影响用户体验，急需找到问题原因并解决。

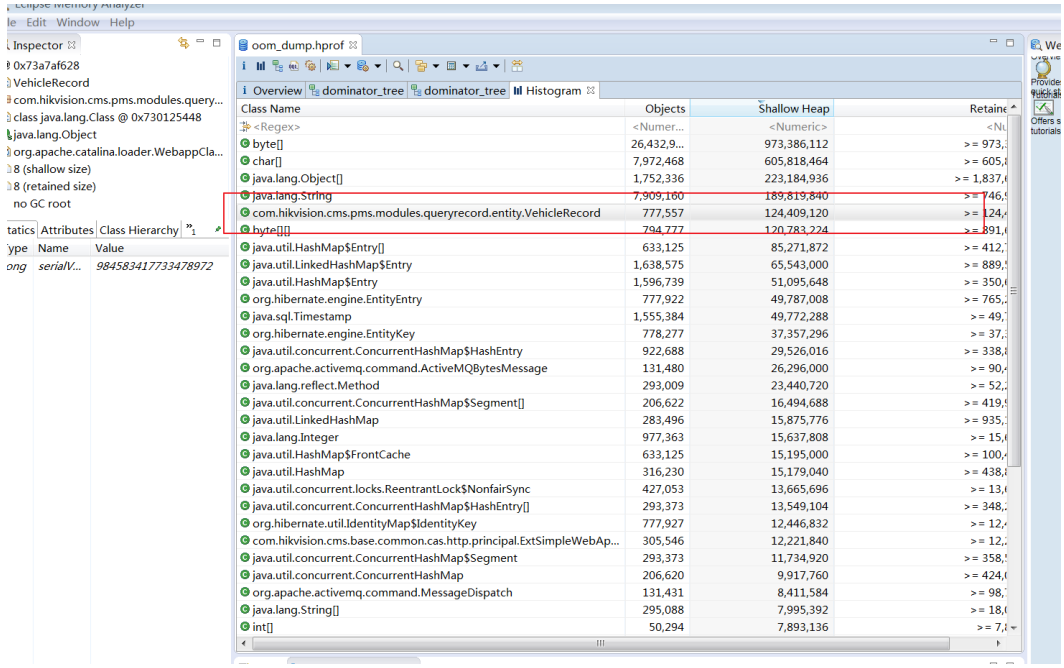
2. 定位问题

首先是获取现场 dump 文件，然后分析内存泄漏点，找到解决问题的原因，并最终给出解决方案。

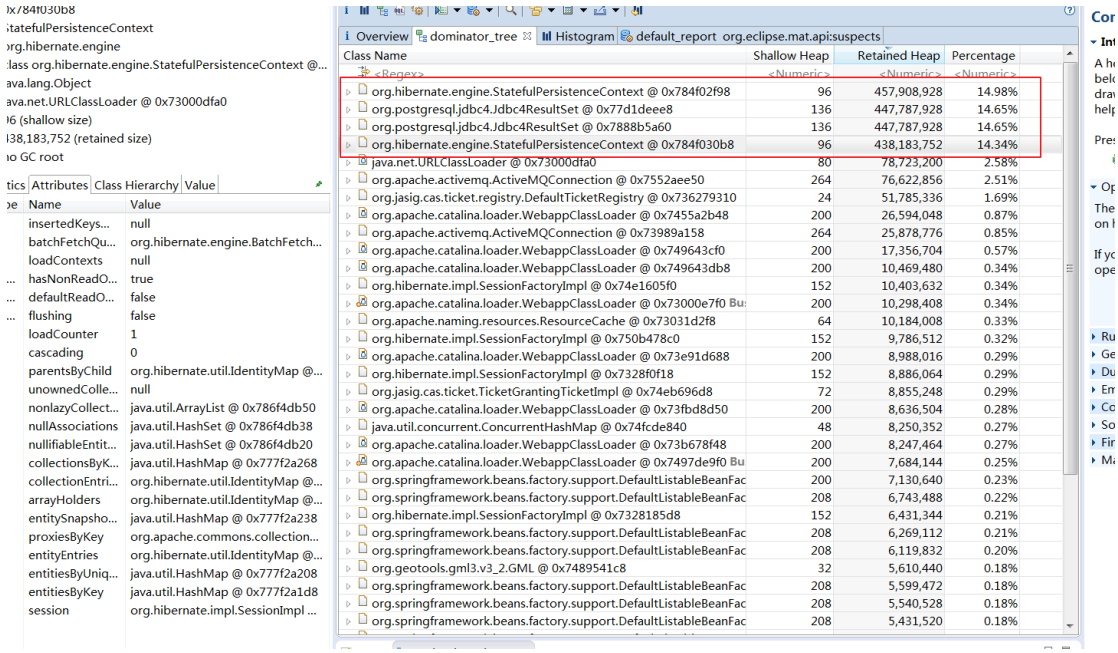
拿到 dump 文件后，用 EcLipse Memory Analyzer 工具打开 dump 文件，看到占用内存 2.8G。其中有三个地方有异常，如下图：



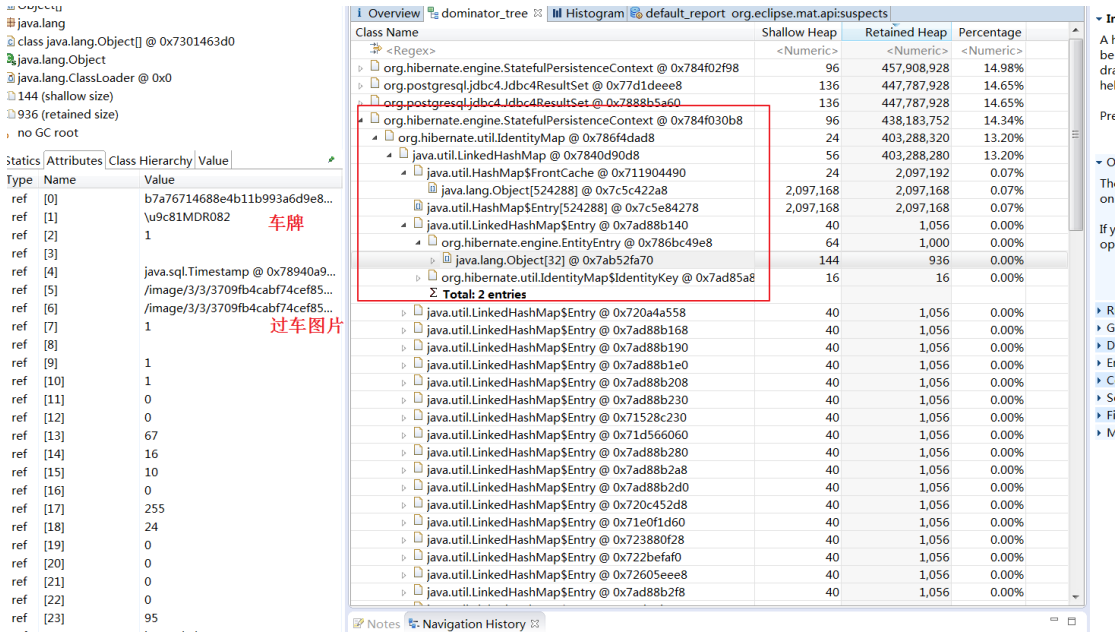
看下 Histogram 列表，会发现有各类 VehicleRecord 占用内从较多，所以初步考虑是停车场（pms）的过车记录相关操作所致，如下图：

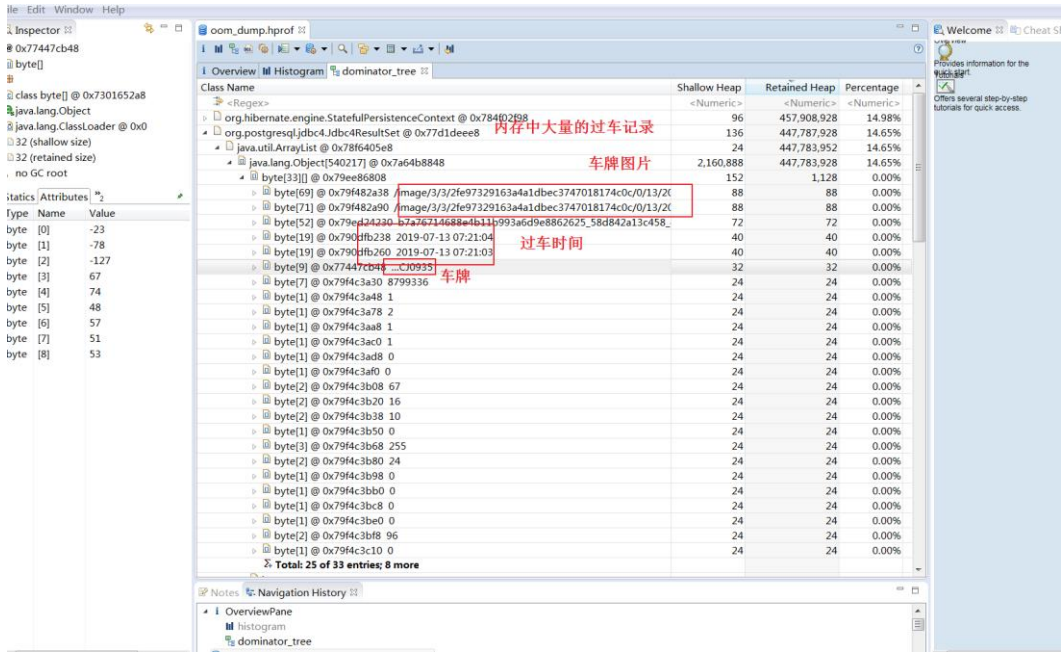


再看下 dominator_tree，其中有四个数据库查询相关操作占用达到 60%，严重异常。

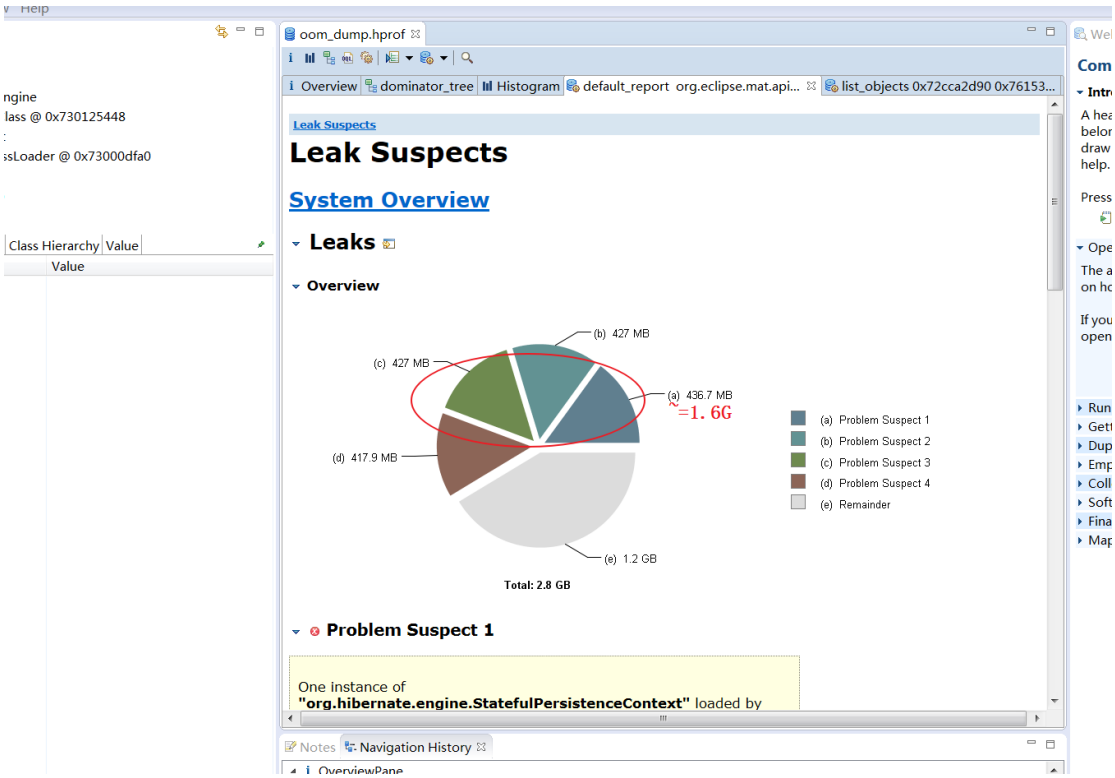


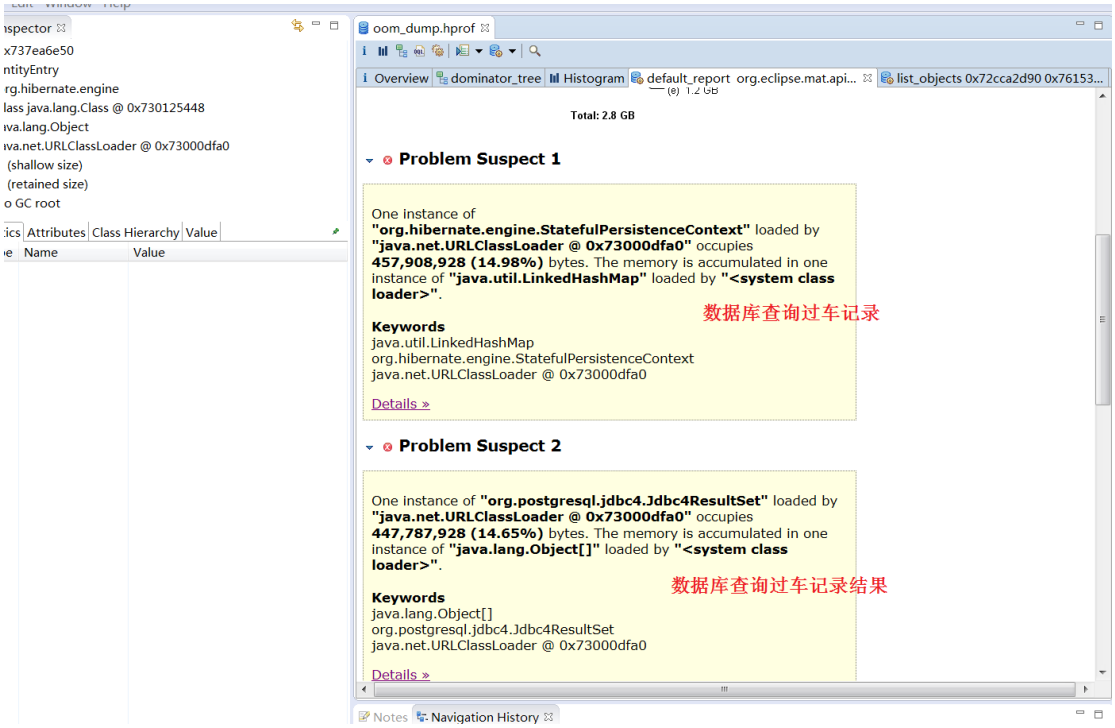
点进去看到是过车记录数据。

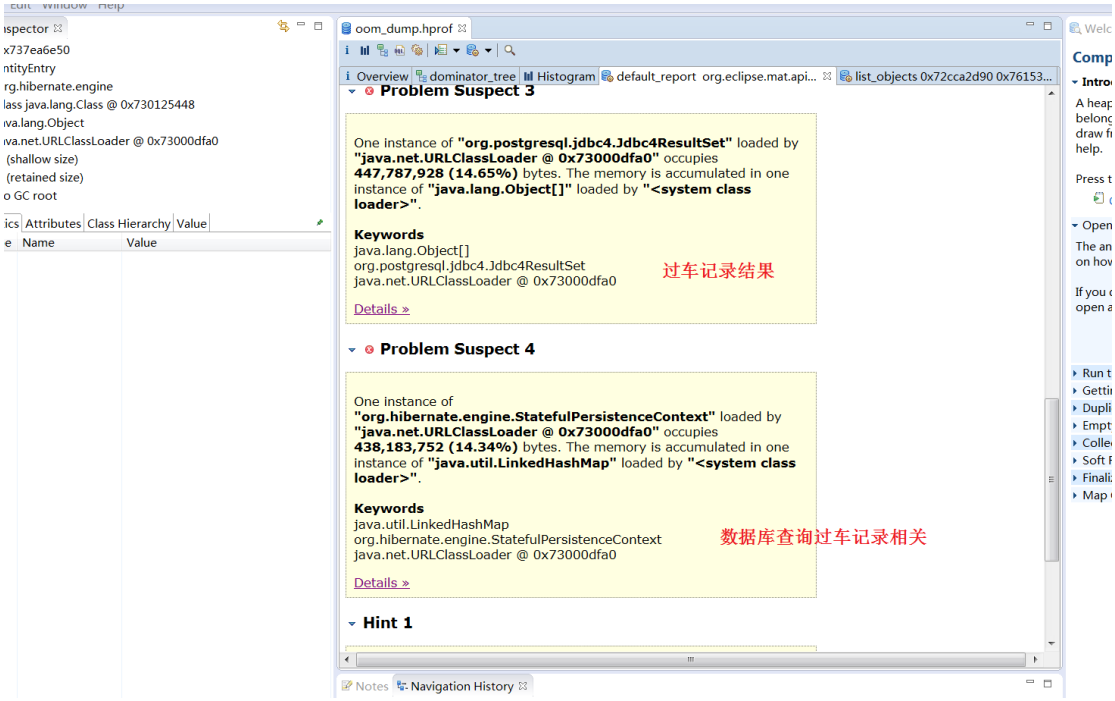




查看 Leak Suspects，进一步证实是过车记录相关操作引起的，1，2，3，4 总共用近 1.6G 内存。

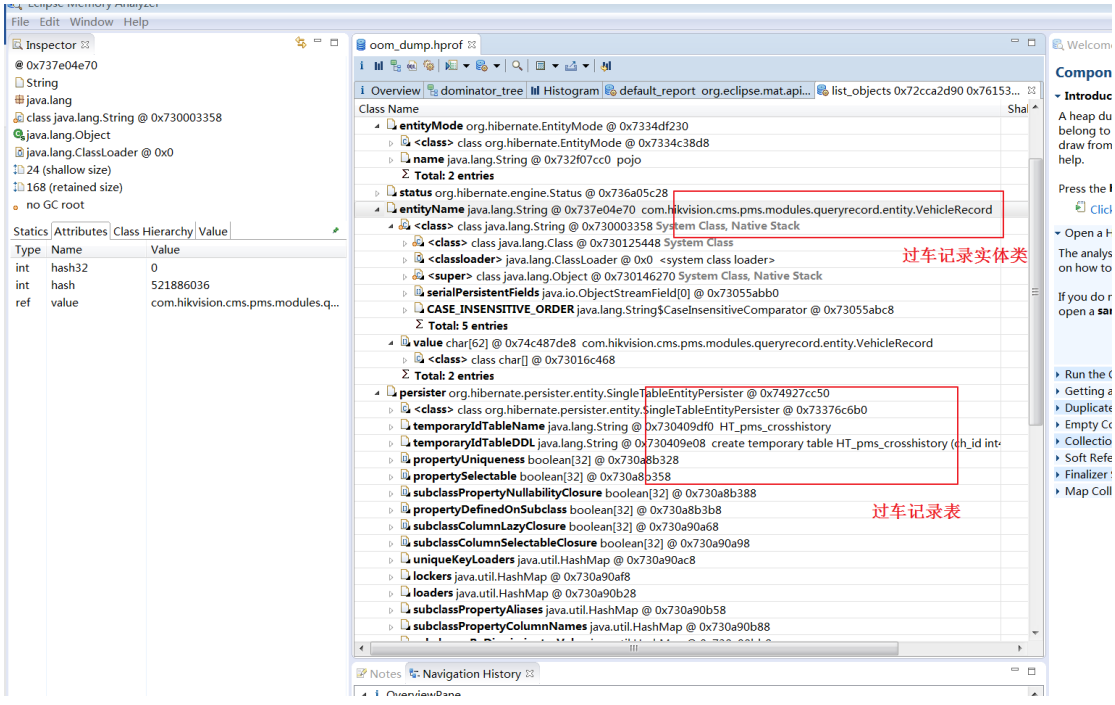

oom_dump.hprof
Overview
dominator_tree
Histogram
default_report org.eclipse.mat.api...
list_objects 0x72cca2d90 0x76153...
Leak Suspects
Leak Suspects
System Overview
Leaks
Overview
(b) 427 MB
(c) 427 MB
(a) 436.7 MB
≈1. 6G
(d) 417.9 MB
(e) 1.2 GB
Total: 2.8 GB
(a) Problem Suspect 1
(b) Problem Suspect 2
(c) Problem Suspect 3
(d) Problem Suspect 4
(e) Remainder
Problem Suspect 1
One instance of
"org.hibernate.engine.StatefulPersistenceContext" loaded by
Notes
Navigation History
Class Hierarchy
Value


oom_dump.hprof
Overview
dominator_tree
Histogram
default_report org.eclipse.mat.api...
list_objects 0x72cca2d90 0x76153...
Total: 2.8 GB
Problem Suspect 1
One instance of
"org.hibernate.engine.StatefulPersistenceContext" loaded by
"java.net.URLClassLoader @ 0x73000dfa0" occupies
457,908,928 (14.98%) bytes. The memory is accumulated in one
instance of "java.util.LinkedHashMap" loaded by "<system class
loader>".
数据库查询过车记录
Keywords
java.util.LinkedHashMap
org.hibernate.engine.StatefulPersistenceContext
java.net.URLClassLoader @ 0x73000dfa0
Details »
Problem Suspect 2
One instance of "org.postgresql.jdbc4.Jdbc4ResultSet" loaded by
"java.net.URLClassLoader @ 0x73000dfa0" occupies
447,787,928 (14.65%) bytes. The memory is accumulated in one
instance of "java.lang.Object[]" loaded by "<system class
loader>".
数据库查询过车记录结果
Keywords
java.lang.Object[]
org.postgresql.jdbc4.Jdbc4ResultSet
java.net.URLClassLoader @ 0x73000dfa0
Details »
Notes
Navigation History
Attributes
Class Hierarchy
Value
Name

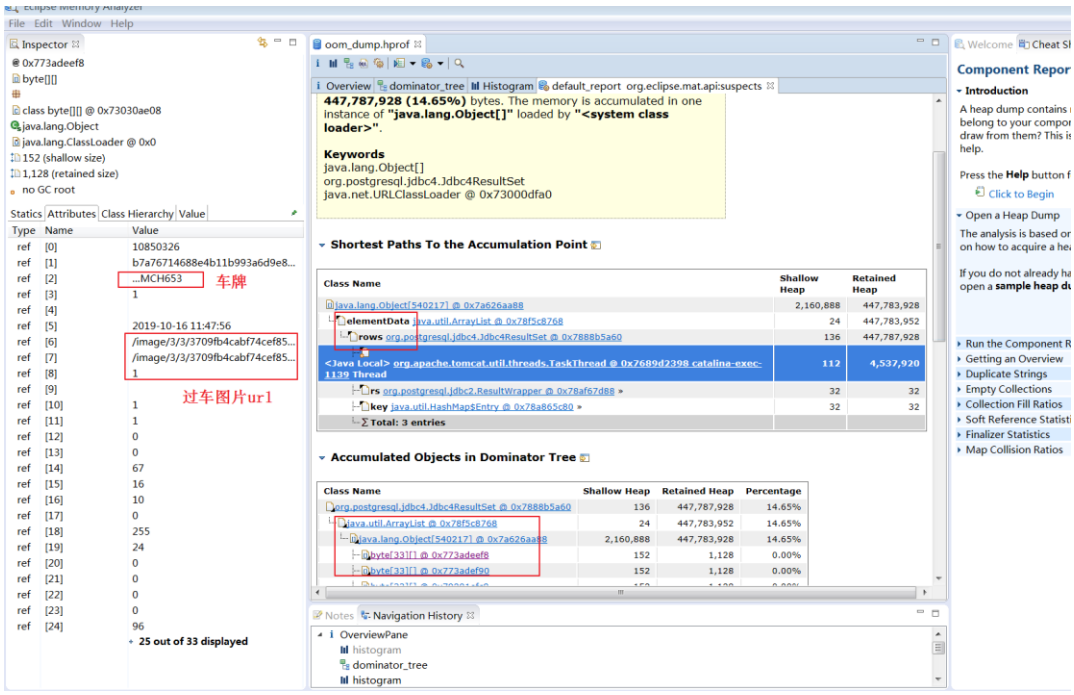


点开详情，都是数据库查询过车记录相关操作

问题点 1 和 4 详情如下：



问题点 2 和 3 详情如下：



经初步沟通，现场视频，门禁，停车场数据量都很大。其中过车辆大约平均每天 2 万辆过车辆。这么多的过车记录肯定不是过车引起的，只能是查询或者导出。

经过沟通得知，现场大约半个月到一个月就会导出一个月的过车数据，做数据分析。约有 70-100 万条记录，而且是一个 Excel 导出。所以初步定为是导出操作引起的。

让现场在晚上验证，一次导出一个月数据，平台果然崩溃，所以断定是导出一个月的大量过车记录导致的，接下来就是要优化导出操作。

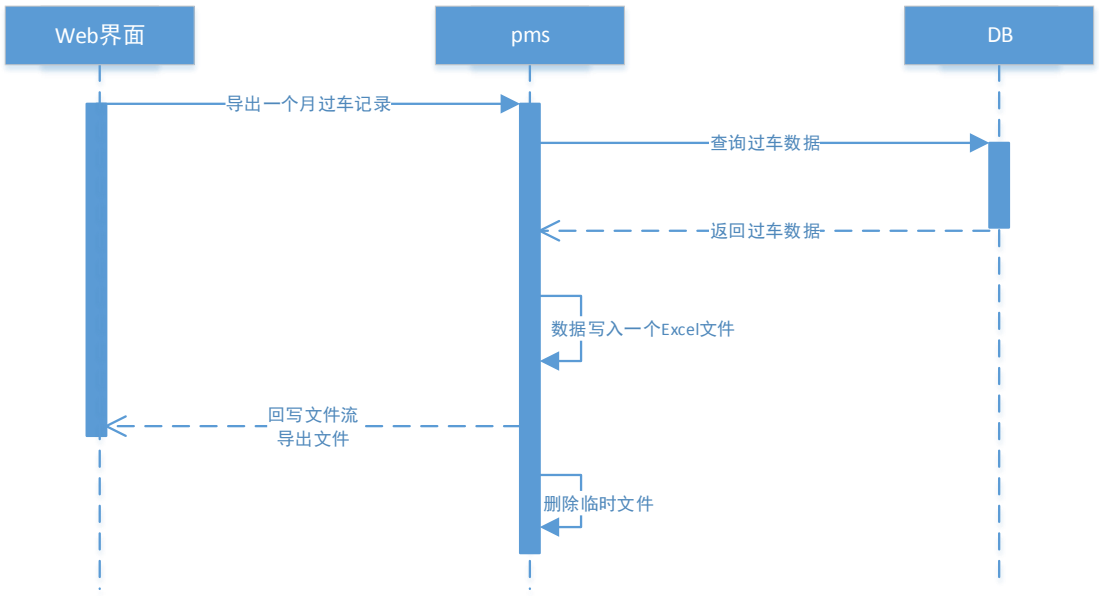
3. 优化思路

一次性导出 100 万条记录，主要耗时点大概分为三个点：1.查询数据库，2.写 excel 文件，3.回传给前端文件数据。

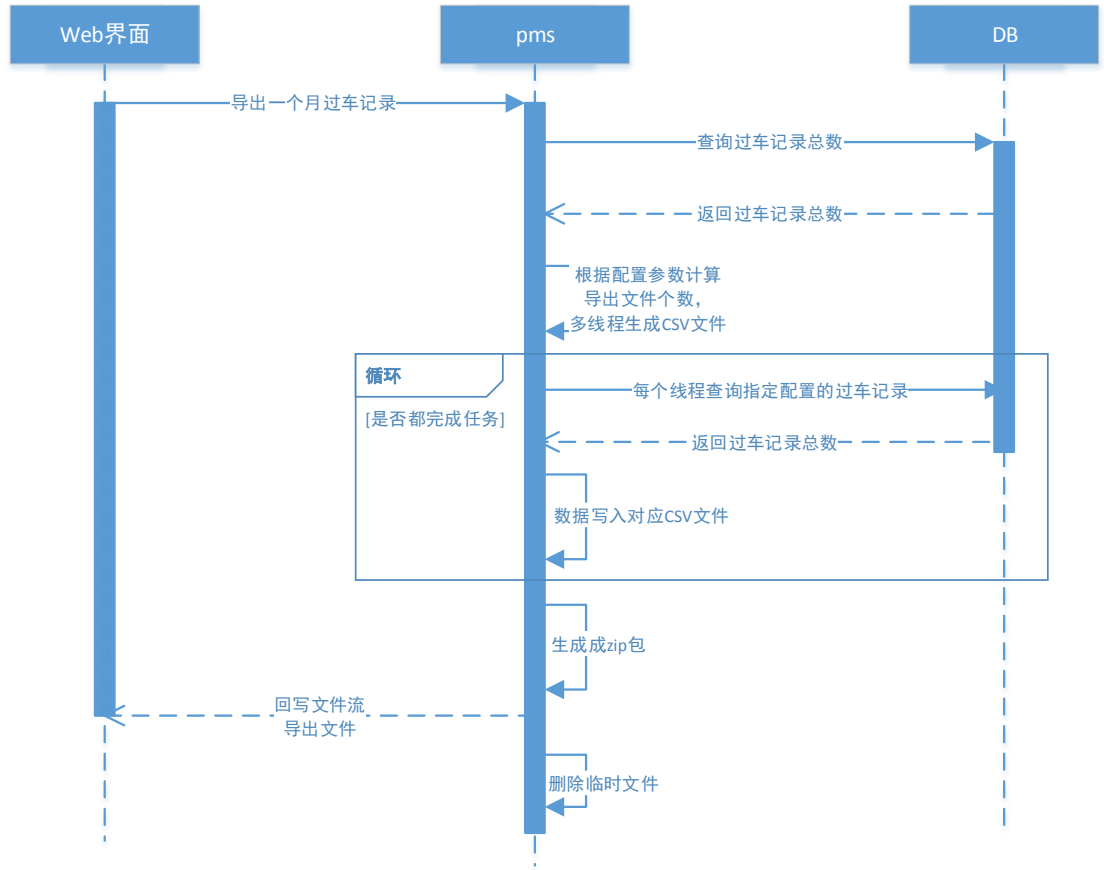
分析可知，查询数据库，由于是单线程操作，较为耗时。且将 100 万条记录写入一个文件，会占用较大内存，且比较耗时的。最后是回传文件，由于文件较

大，回传给前端时也耗时，占用内存。

所以优化思路方案渐渐清晰 :采用多线程、分文件、压缩 zip 包形式导出 CSV，CSV 文件比 Excel 写入要快。优化前后过车记录导出方案时序图对比，如下两图所示：



优化前过车记录导出方案时序图



优化后过车记录导出方案时序图

具体代码实现思路：根据配置的每个文件的记录数，算出需要导出文件个数 N，然后启动 min(文件个数 N，10)的线程。去导出 N 个 CSV 文件，最后文件都完成后在一起压缩导出。

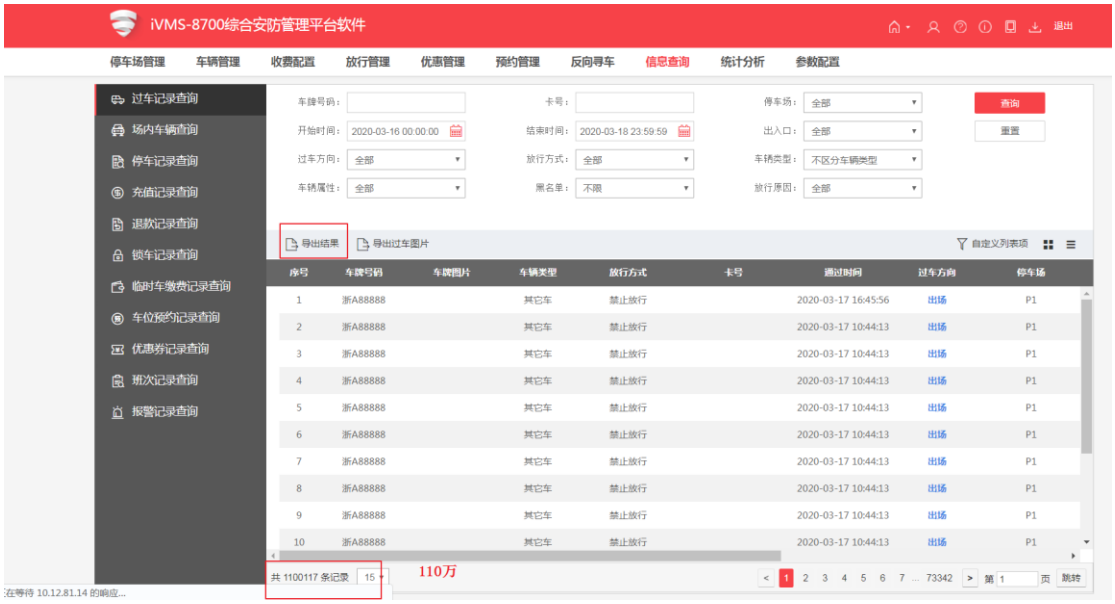
核心代码展示：

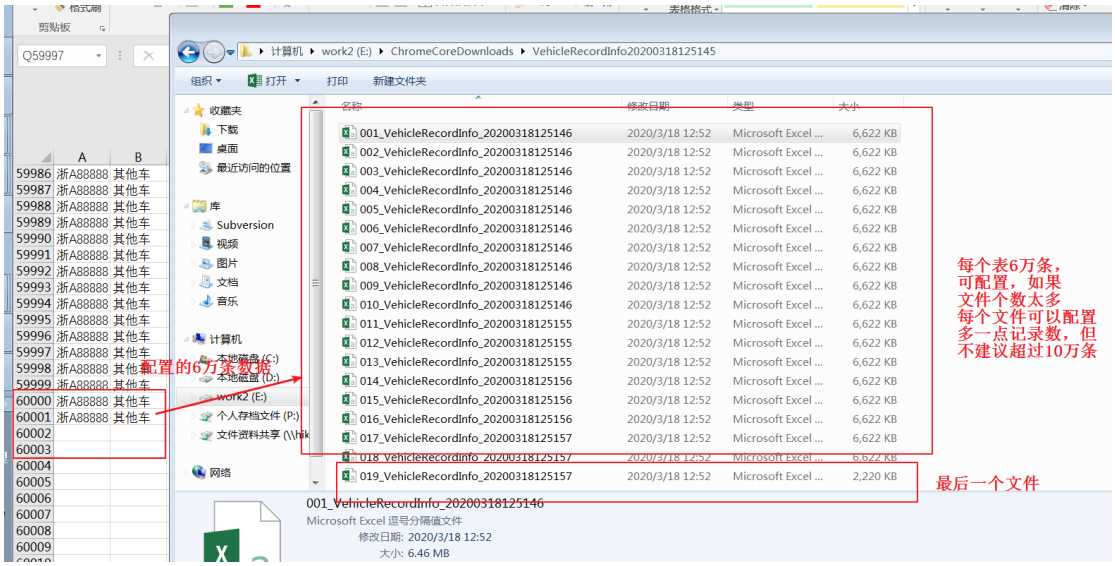
```
final int totalPage = pageBean.getTotalPage();
PmsLogRecord.logInfo("handleCsvFiles => totalPage is " + totalPage);
int totalFileNum = (totalPage * pageSize + totalNumOneFile -1) / totalNumOneFile ;//计算总的文件数，向上取整
final int pageNumEveryFile = totalNumOneFile / pageSize;
if (totalFileNum <= 1){//单个文件时，不开多线程
   exportExcel(headNames, CsvUtil.createCSVFileEx("001_" + fileName, tempDir.getPath()) ,condition, 1, pageNumEveryFile,  totalPage) ;//导出数据
}else{//多线程导出
   ExecutorService executorService = new ThreadPoolExecutor(Math.min(totalFileNum, 10), Math.min(totalFileNum, 10), 0L, TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>(200));
```

```
  final CountDownLatch countDownLatch =  new CountDownLatch(totalFileNum);
   for(int i = 1;i <= totalFileNum ; i++){
      executorService.execute(new ExportWorker(pageNumEveryFile,
i,fileName,headNames,condition,totalPage,countDownLatch,tempDir));//参数太多，可封装为pojo类
   }
   countDownLatch.await();
   executorService.shutdown();
}

//处理导出任务线程 内部类
   private class ExportWorker implements Runnable{
          ......................
   @Override
   public void run() {
      try {
         File csvFile = CsvUtil.createCSVFileEx(getThreeNum(fileNum) + "_" + fileName,
tempDir.getPath());
         Thread.currentThread().setName(csvFile.getName());
         exportExcel(headNames, csvFile ,condition, (fileNum - 1) * pageNumEveryFile + 1,
Math.min(totalPage, fileNum * pageNumEveryFile),  totalPage) ;//导出数据
      }catch (Exception e){
         PmsLogRecord.logException(e, "exception happened when execute export thread",e.getMessage());
      }finally {
         countDownLatch.countDown();
      }
   }
}
```

最后测试 110 万条数据，导出花费 20-30S 的时间！导出效果如下：





4. 总结分析

定位问题方面：从问题表面触发，借用工具，结合代码和业务，一步一步分析，最终分析发现问题的原因，找到解决办法。

代码设计方面 :平时方案和代码设计时，需要考虑多并发和大数据量的场景，尽量让自己的代码具有较高的稳定性，可靠性，可扩展性。

5. 附件代码

```
/**
 * 多线程 多文件 zip 压缩包导出
 * 目标：100 万条数据，导出，不超过 30 秒
 */
@Override
public void exportVehicleRecord(SearchCondition condition, UserLoginPojo user, HttpServletResponse response) {
    dealWithCondition(condition);
    String[] headNames = {"车牌号码","车辆类型","放行方式","卡号","通过时间","过车方向","停车场","出入口","一户多车车位","放行原因","车辆属性"};//,"缴费状态"标题
    this.pmsRefService.logRecord(null, "logContent.pms.exportVehicleRecord", null);//导入 Excel 文件类型，名称为 VehicleRecord
    String fileName = "VehicleRecordInfo" + DateTimeUtil.dateToString(new Date(),DateTimeUtil.dtLong);
    response.setContentType("application/x-msdownload;");
    response.setHeader("Content-disposition", "attachment; filename=" + fileName + ".zip");
    File tempDir = CsvUtil.newFile("VehicleRecordTempDir" + DateTimeUtil.dateToString(new Date(),DateTimeUtil.dtLong));//适配多客户端导出
    try (ZipOutputStream out = new ZipOutputStream(response.getOutputStream())){
        handleCsvFiles(condition, headNames, "VehicleRecordInfo",tempDir);
        FileUtil.zipDirectoryOrFile(out, tempDir);
        out.flush();
        FileUtil.deletesDir(tempDir);//删除临时文件
    } catch (Exception e) {
        PmsLogRecord.logException(e);
    }
}

// 文件分组问题，等待多文件写完，唤醒主线程 zip 文件
private void handleCsvFiles(final SearchCondition condition, final String[] headNames, final String fileName,final File tempDir) throws InterruptedException {
    long beginTime = System.currentTimeMillis();
    int totalNumOneFile = getEveryFileNum(3) * 20000 ;//每个文件的数量 配置文件配置
    int pageSize = 20000;//每次查询 20000 条
    PmsLogRecord.logInfo("handleCsvFiles => config totalNumOneFile is " + totalNumOneFile);
    if (totalNumOneFile > 200000 || totalNumOneFile < 0){
        totalNumOneFile = 100000;//最大每个不超过 20 万
    }
    PmsLogRecord.logInfo("handleCsvFiles => pageSize is " + pageSize);
    PmsLogRecord.logInfo("handleCsvFiles => last totalNumOneFile is " + totalNumOneFile);
    condition.setPageSize(pageSize);//后续用配置的页数
    condition.setPageNo(1);
    PageBean<VehicleRecord> pageBean = queryVehicleRecordDao.fetchPageBean(condition);//可写简化 sql，为
```

```
了省事不写了，影响不大
    final int totalPage = pageBean.getTotalPage();
    PmsLogRecord.logInfo("handleCsvFiles => totalPage is " + totalPage);
    int totalFileNum = (totalPage * pageSize + totalNumOneFile -1) / totalNumOneFile ;//计算总的文件数，向上取整
    PmsLogRecord.logInfo("handleCsvFiles => totalFileNum is " + totalFileNum);
    final int pageNumEveryFile = totalNumOneFile / pageSize;
    if (totalFileNum <= 1){//单个文件时，不开多线程
        exportExcel(headNames, CsvUtil.createCSVFileEx("001_" + fileName, tempDir.getPath()) ,condition, 1, pageNumEveryFile, totalPage) ;//导出数据
    }else{//多线程导出
        ExecutorService executorService = new ThreadPoolExecutor(Math.min(totalFileNum, 10), Math.min(totalFileNum, 10), 0L, TimeUnit.MILLISECONDS, new LinkedBlockingQueue<Runnable>(200));
        final CountDownLatch countDownLatch = new CountDownLatch(totalFileNum);
        for(int i = 1;i <= totalFileNum ; i++){
            executorService.execute(new ExportWorker(pageNumEveryFile, i,fileName,headNames,condition,totalPage,countDownLatch,tempDir));//参数太多，可封装为 pojo 类
        }
        countDownLatch.await();
        executorService.shutdown();
    }
    PmsLogRecord.logInfo("handleCsvFiles => cost time is "+ (System.currentTimeMillis() - beginTime )+"ms");
}

/**获取配置的单文件记录数
 * 每个 excel 的记录数量=every_file_num * 2 万，最大为 5，为 10 万条，大于十万，默认是 10 万条/excel 文件
 * every_file_num = 3
 */
private static int getEveryFileNum(int defaultEveryFileNum) {
    String everyFileNum = PropertiesReader.getWebProperty("every_file_num");
    PmsLogRecord.logInfo("everyFileNum is " + everyFileNum);
    if (PmsOptUtil.isStringEmpty(everyFileNum) || !everyFileNum.matches("\\d+")){
        return defaultEveryFileNum;
    }
    return Math.max(defaultEveryFileNum, Integer.valueOf(everyFileNum));
}

//处理导出任务线程 内部类
    private class ExportWorker implements Runnable{
    private int fileNum;
    private String fileName;
    private String[] headNames;
    private SearchCondition condition;
```

```
    private int totalPage;
    private int pageNumEveryFile;
    private CountDownLatch countDownLatch;
    private File tempDir ;

    public ExportWorker(int pageNumEveryFile, int fileNum, String fileName, String[] headNames,
SearchCondition condition, int totalPage, CountDownLatch countDownLatch, File tempDir){
        this.condition = condition;
        this.countDownLatch= countDownLatch;
        this.headNames = headNames;
        this.fileName = fileName;
        this.fileNum = fileNum;
        this.totalPage = totalPage;
        this.pageNumEveryFile = pageNumEveryFile;
        this.tempDir = tempDir;
    }
    @Override
    public void run() {
        try {
            File csvFile = CsvUtil.createCSVFileEx(getThreeNum(fileNum) + "_" + fileName, tempDir.getPath());
            Thread.currentThread().setName(csvFile.getName());
            exportExcel(headNames, csvFile ,condition, (fileNum - 1) * pageNumEveryFile + 1,
Math.min(totalPage, fileNum * pageNumEveryFile), totalPage) ;//导出数据
        }catch (Exception e){
            PmsLogRecord.logException(e, "exception happened when execute export thread",e.getMessage());
        }finally {
            countDownLatch.countDown();
        }
    }
}

//补位成 001 开始的文件头
private static String getThreeNum(int k) {
    if (k < 10){
        return "00".concat(String.valueOf(k));
    }else if(k < 100){
        return "0".concat(String.valueOf(k));
    }else{
        return String.valueOf(k);
    }
}
```