

课程设计

音乐播放器

Window 资源管理器

&

基于 *JProfiler* 的内存检测

Student Name: Cap

Email: littlegreedy@qq.com

日期：三月三号

完成对各处拼接字符串的优化，系统内存资源占用降低 15MB 左右。

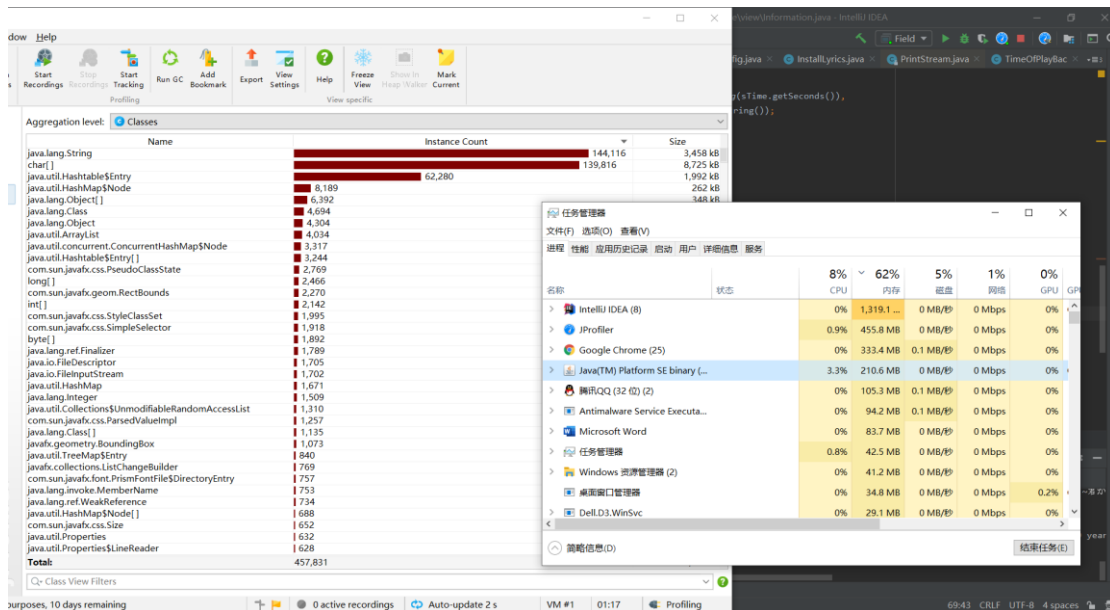
目前有现象表明在 MediaPlayer 类准备事件监听下的：歌词展示、图表、信息摘取等方法类添加的 synchronized 关键字会使资源占用随时间异常递增。初步推测可能是线程阻塞，直至三月四号尚未有理论或实践证据直接证明现象成因。

日期：三月四号

（留意图表时间轴）

减少部分同步锁，转而对需要 CPU 内存屏障的 Clock、Lyrics 的对象添加 volatile 关键词，资源占用同样随时间异常递增。





又一个减少对象创建运用静态方法的优化：把 `timeShowStart` 方法调用 `Clock` 类的构造函数显式地创建对象的方式，改为传递对象的引用。

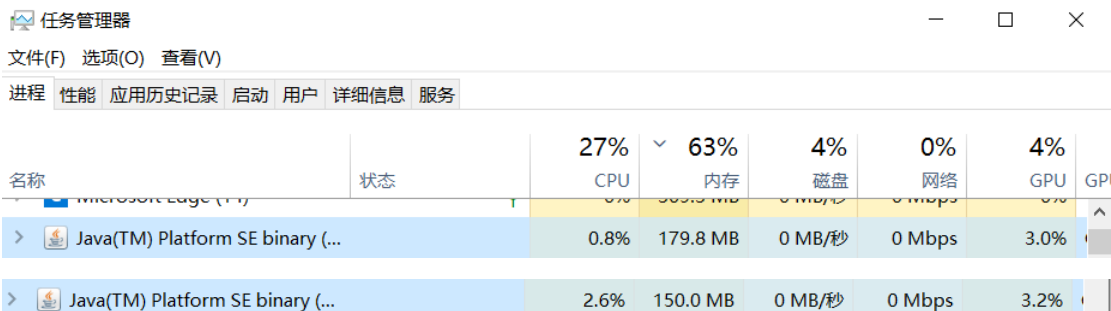
```
private void currentTimeProperty() throws Exception{
    mPlayer.currentTimeProperty().addListener(new ChangeListener<Duration>() {
        // ContinuousAudioDataStream
        @Override
        public void changed(ObservableValue<? extends Duration> observable, Duration oldValue, Duration newValue) {
            if(!mouse){
                sPlayback.setValue(newValue.toSeconds());
                // 歌曲时间文本
                timeOfPlayback.timeShowStart(mPlayer,text);
                // 歌词滚动
                timeOfPlayback.lyricsStart(mPlayer,installLyrics.getLyricText(),installLyrics.getLyrics());
            }
            // 歌曲结束，下一步做什么依赖此时播放模式
            double exp=10E-2;
            if(Math.abs(newValue.toSeconds()-mPlayer.getStopTime().toSeconds()) < exp ){
                playPattern();
            }
        }
    });
}
```

增加对图表刷新的控制，目的是减少占用的空间。

```
private void handleAudioSpectrum() throws Exception{
    //-----
    // mPlayer.setAudioSpectrumInterval(1.0);
    mPlayer.setAudioSpectrumListener(new AudioSpectrumListener() {
        @Override
        public void spectrumDataUpdate(double timestamp, double duration, float[] magnitudes, float[] phases) {
            // ChartsSpectrumData Cs=new ChartsSpectrumData(ap);
            // Cs.buildCharts(100,300,8,4,magnitudes);
            //System.out.print(timestamp);
            //更新高度数据并绘制矩形阵频谱
            chartsSpectrumData.refurbishRecChart(magnitudes,showPic);
            //更新高度数据并绘制环形阵频谱
            chartsSpectrumData.refurbishCirChart(magnitudes,showPic);
            // buildCharts(startX,startY,rectwidth,rectangle_num,magnitudes);
        }
    });
}
```

去除 volatile、synchronization（对异步锁机制了解尚浅，以免处理不当产生堵塞等异常事件）：

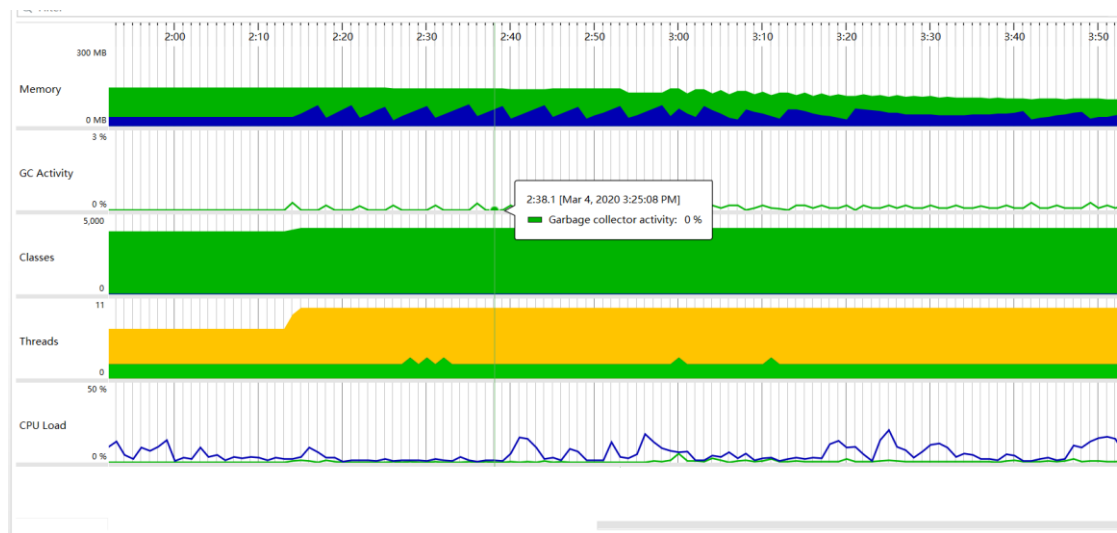
测试显示含有极少数或者忽略不计的堵塞现象。



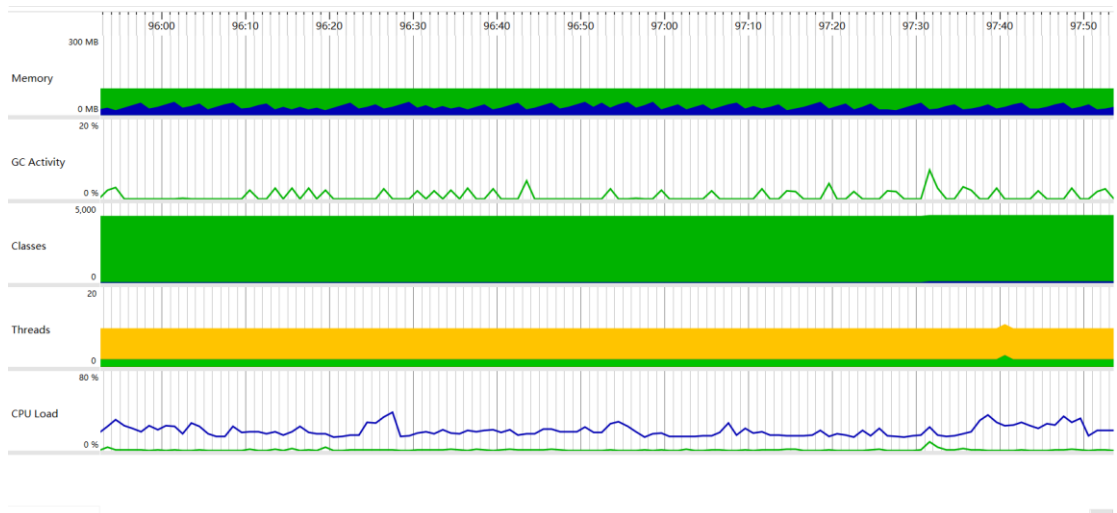
第一个小坡度为添加 10 份文件（八首歌、两份歌词文件）



紧接着出现地坡度群，为切歌时段。每个状态的运行线程数量一致，其中黄色部分为等待线程。



运行时 100 分钟左右资源占用依旧正常。



目前已知可优化：

1) 引入歌曲采用 url 最佳。

2) 歌词文件类的 文件（成员变量）最好改为 TreeMap 存储的歌词。而文件解析在添加时全部完成。

3) 应对多线程不同步的容器进行以下方法包装：

```
Map m = Collections.synchronizedMap(new TreeMap(...));
```

最好在创建时完成这一操作，以防止对映射进行意外的不同步访问

来源：

https://blog.csdn.net/qq_43746676/article/details/87884939

4) 如何高效地进行垃圾回收。在确定了哪些垃圾可以被回收后，垃圾收集器要做的事情就是开始进行垃圾回收。

5) 内存泄漏。 SortedMap 具有强引用，虽然 clear 之后 key 和 value 为空，但是 JVM 的垃圾回收器并不会回收该对象的内存，大量没有释放的对象可能造成内存泄漏。