XStream 类和内存泄漏分析

作者：96XL
原文链接：https://ld246.com/article/1638340264037
来源网站：链滴

前言

  最近项目在进行性能测试，在最后做稳定性测试的过程中出现一个问题：前
个小时tps一直很稳定，过了某一个时间点，tps随着时间的推移不断下降，直到降为0。观察平台监
，发现内存开销不断上升，通过监控定位到产生问题的代码为一个使用XStream的工具类，工具类中
用XStream实现了xml与bean的互相转换。该工具类导致了内存泄露问题，后来百度了一下，找到了
生问题的原因与解决办法，记录一下处理办法。

原因分析

  在Java中，内存泄漏就是存在一些被分配的对象，这些对象有下面两个特点
首先，这些对象是可达的，即在有向图中，存在通路可以与其相连；其次，这些对象是无用的，即程
以后不会再使用这些对象。如果对象满足这两个条件，这些对象就可以判定为Java中的内存泄漏，这
对象不会被GC所回收，然而它却占用内存。

```
public static String objectToXml(Object obj) {
    XStream xstream = new XStream();
    // xstream使用注解转换
    xstream.processAnnotations(obj.getClass());
    // 启用Annotation
    xstream.autodetectAnnotations(true);
    return xstream.toXML(obj);
}
```

  工具类中在获取XStream对象时每次都使用new的方式，然后Xstream内部
会new一个CompositeClassLoader对象，并且Class.forName调用该loader对象。问题就出现在这
对象上，minor gc不会回收这种class loader对象，每个请求又都会new一个CompositeClassLoade
对象，存在大量可达且无用对象，那就会导致堆空间逐渐被占满，并且产生full gc。

解决方案

  在方法外new一个全局静态XStream对象，这样CompositeClassLoader对
只会被new一个，虽然CompositeClassLoader对象存在堆区，但是静态finnal级别的XStream对象
在堆区，因此不存在可达且无用对象，从而minor gc会回收这种class loader对象。

```
private static final XStream xstream = new XStream(new DomDriver());

public static String objectToXml(Object obj) {
    // xstream使用注解转换
    xstream.processAnnotations(obj.getClass());
    // 启用Annotation
    xstream.autodetectAnnotations(true);
    return xstream.toXML(obj);
}
```