**[JDK8-废弃永久代（PermGen）迎来元空间（Metaspace）](https://www.cnblogs.com/dennyzhangdd/p/6770188.html)**

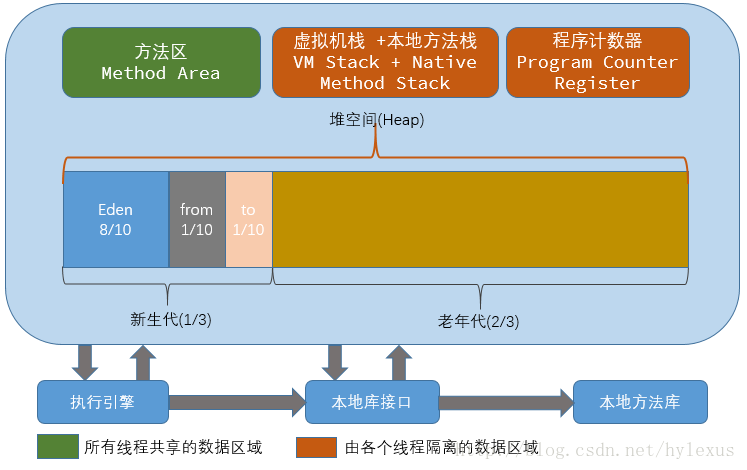
<https://www.cnblogs.com/dennyzhangdd/p/6770188.html>

**目录**

* [一、背景](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label0)
  + [1.1 永久代（PermGen）在哪里？](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label0_0)
  + [1.2 JDK8永久代的废弃](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label0_1)
* [二、为什么废弃永久代（PermGen）](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label1)
  + [2.1 官方说明](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label1_0)
* [Motivation](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label2)
  + [2.2 现实使用中易出问题](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label2_0)
* [三、深入理解元空间（Metaspace）](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label3)
  + [3.1元空间的内存大小](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label3_0)
  + [3.2常用配置参数](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label3_1)
  + [3.3测试并追踪元空间大小](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label3_2)
* [四、总结](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_label4)
* **正文**
* [回到顶部](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_labelTop)

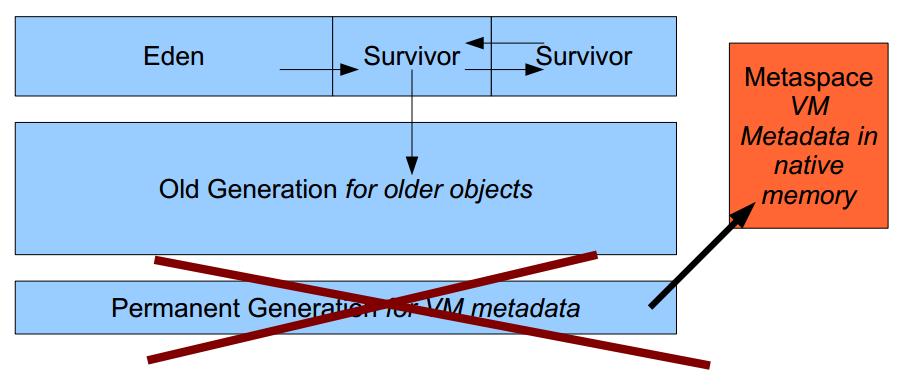
## 一、背景

### 1.1 永久代（PermGen）在哪里？

* 根据，hotspot jvm结构如下(虚拟机栈和本地方法栈合一起了)：
* 
* 上图引自网络，但有个问题：方法区和heap堆都是线程共享的内存区域。
* **关于方法区和永久代：**
* 在HotSpot JVM中，这次讨论的**永久代**，就是上图的方法区（JVM规范中称为方法区）。《Java虚拟机规范》只是规定了有方法区这么个概念和它的作用，并没有规定如何去实现它。在其他JVM上不存在永久代。

### 1.2 JDK8永久代的废弃

JDK8 永久代变化如下图：



1.新生代：Eden+From Survivor+To Survivor

2.老年代：OldGen

3.永久代（方法区的实现） : PermGen----->替换为Metaspace(本地内存中)

[回到顶部](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_labelTop)

## 二、为什么废弃永久代（PermGen）

### 2.1 官方说明

参照JEP122：http://openjdk.java.net/jeps/122，原文截取：

[回到顶部](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_labelTop)

## Motivation

This is part of the JRockit and Hotspot convergence effort. JRockit customers do not need to configure the permanent generation (since JRockit does not have a permanent generation) and are accustomed to not configuring the permanent generation.

 即：移除永久代是为融合HotSpot JVM与 JRockit VM而做出的努力，因为JRockit没有永久代，不需要配置永久代。

### 2.2 现实使用中易出问题

由于永久代内存经常不够用或发生内存泄露，爆出异常java.lang.OutOfMemoryError: PermGen

[回到顶部](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_labelTop)

## 三、深入理解元空间（Metaspace）

### 3.1元空间的内存大小

元空间是方法区的在HotSpot jvm 中的实现，方法区主要用于存储类的信息、常量池、方法数据、方法代码等。方法区逻辑上属于堆的一部分，但是为了与堆进行区分，通常又叫“非堆”。

元空间的本质和永久代类似，都是对JVM规范中方法区的实现。不过**元空间与永久代之间最大的区别在于：元空间并不在虚拟机中，而是使用本地内存。**，理论上取决于32位/64位系统可虚拟的内存大小。可见也不是无限制的，需要配置参数。

### 3.2常用配置参数

1.MetaspaceSize

初始化的Metaspace大小，控制元空间发生GC的阈值。GC后，动态增加或降低MetaspaceSize。在默认情况下，这个值大小根据不同的平台在12M到20M浮动。使用[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "_blank) -XX:+PrintFlagsInitial命令查看本机的初始化参数

2.MaxMetaspaceSize

限制Metaspace增长的上限，防止因为某些情况导致Metaspace无限的使用本地内存，影响到其他程序。在本机上该参数的默认值为4294967295B（大约4096MB）。

3.MinMetaspaceFreeRatio

当进行过Metaspace GC之后，会计算当前Metaspace的空闲空间比，如果空闲比小于这个参数（即实际非空闲占比过大，内存不够用），那么虚拟机将增长Metaspace的大小。默认值为40，也就是40%。设置该参数可以控制Metaspace的增长的速度，太小的值会导致Metaspace增长的缓慢，Metaspace的使用逐渐趋于饱和，可能会影响之后类的加载。而太大的值会导致Metaspace增长的过快，浪费内存。

4.MaxMetasaceFreeRatio

当进行过Metaspace GC之后， 会计算当前Metaspace的空闲空间比，如果空闲比大于这个参数，那么虚拟机会释放Metaspace的部分空间。默认值为70，也就是70%。

5.MaxMetaspaceExpansion

Metaspace增长时的最大幅度。在本机上该参数的默认值为5452592B（大约为5MB）。

6.MinMetaspaceExpansion

Metaspace增长时的最小幅度。在本机上该参数的默认值为340784B（大约330KB为）。

### 3.3测试并追踪元空间大小

#### 3.3.1.测试字符串常量

[复制代码](javascript:void(0);)

1 public class StringOomMock {

2 static String base = "string";

3

4 public static void main(String[] args) {

5 List<String> list = new ArrayList<String>();

6 for (int i=0;i< Integer.MAX\_VALUE;i++){

7 String str = base + base;

8 base = str;

9 list.add(str.intern());

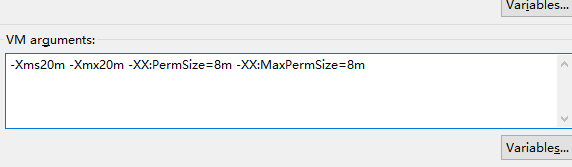
10 }

11 }

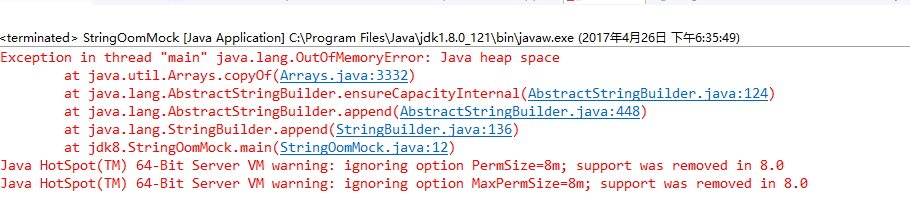
12 }

[复制代码](javascript:void(0);)

在eclipse中选中类--》run configuration-->java application--》new 参数如下：



 由于设定了最大内存20M，很快就溢出，如下图：



 可见在jdk8中：

1.字符串常量由永久代转移到堆中。

2.持久代已不存在，PermSize MaxPermSize参数已移除。（看图中最后两行）

#### 3.3.2.测试元空间溢出

根据定义，我们以加载类来测试元空间溢出，代码如下：

[复制代码](javascript:void(0);)

1 package jdk8;

2

3 import java.io.File;

4 import java.lang.management.ClassLoadingMXBean;

5 import java.lang.management.ManagementFactory;

6 import java.net.URL;

7 import java.net.URLClassLoader;

8 import java.util.ArrayList;

9 import java.util.List;

10

11 /\*\*

12 \*

13 \* @ClassName:OOMTest

14 \* @Description:模拟类加载溢出（元空间oom）

15 \* @author diandian.zhang

16 \* @date 2017年4月27日上午9:45:40

17 \*/

18 public class OOMTest {

19 public static void main(String[] args) {

20 try {

21 //准备url

22 URL url = new File("D:/58workplace/11study/src/main/java/jdk8").toURI().toURL();

23 URL[] urls = {url};

24 //获取有关类型加载的JMX接口

25 ClassLoadingMXBean loadingBean = ManagementFactory.getClassLoadingMXBean();

26 //用于缓存类加载器

27 List<ClassLoader> classLoaders = new ArrayList<ClassLoader>();

28 while (true) {

29 //加载类型并缓存类加载器实例

30 ClassLoader classLoader = new URLClassLoader(urls);

31 classLoaders.add(classLoader);

32 classLoader.loadClass("ClassA");

33 //显示数量信息（共加载过的类型数目，当前还有效的类型数目，已经被卸载的类型数目）

34 System.out.println("total: " + loadingBean.getTotalLoadedClassCount());

35 System.out.println("active: " + loadingBean.getLoadedClassCount());

36 System.out.println("unloaded: " + loadingBean.getUnloadedClassCount());

37 }

38 } catch (Exception e) {

39 e.printStackTrace();

40 }

41 }

42 }

[复制代码](javascript:void(0);)

为了快速溢出，设置参数：-XX:MetaspaceSize=8m -XX:MaxMetaspaceSize=80m，运行结果如下：



 上图证实了，我们的JDK8中类加载（方法区的功能）已经不在永久代PerGem中了，而是Metaspace中。可以配合JVisualVM来看，更直观一些。

[回到顶部](https://www.cnblogs.com/dennyzhangdd/p/6770188.html#_labelTop)

## 四、总结

本文讲解了元空间（Metaspace）的由来和本质，常用配置，以及监控测试。元空间的大小是动态变更的，但不是无限大的，最好也时常关注一下大小，以免影响服务器内存。

==================

参考：

[深入探究JVM（2） - 探秘Metaspace](http://blog.csdn.net/sczyh22/article/details/46662279)

[【理解JVM】JVM内存模型](http://blog.csdn.net/moshenglv/article/details/52027216)