



## Red Hat build of OpenJDK 21

在 Red Hat build of OpenJDK 21 中使用  
Shenandoah 垃圾收集器



Red Hat build of OpenJDK 21 在 Red Hat build of OpenJDK 21 中使用  
Shenandoah 垃圾收集器

---

## 法律通告

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 摘要

Red Hat build of OpenJDK 是 Red Hat Enterprise Linux 平台上的红帽产品。在 Red Hat build of OpenJDK 21 中使用 Shenandoah 垃圾收集器 提供了 Shenandoah 垃圾收集器的概述，并解释了如何使用红帽构建的 OpenJDK 21 配置它。

---

## 目录

提供有关红帽构建的 OPENJDK 文档的反馈 .....	3
使开源包含更多 .....	4
第 1 章 SHENANDOAH 垃圾收集器 .....	5
第 2 章 使用 SHENANDOAH 垃圾收集器运行 JAVA 应用程序 .....	6
第 3 章 SHENANDOAH 垃圾收集器模式 .....	7
第 4 章 SHENANDOAH 垃圾收集器的基本配置选项 .....	8



## 提供有关红帽构建的 OPENJDK 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

### 流程

1. 单击以下链接 [以创建 ticket](#)。
2. 在 **Summary** 中输入问题的简短描述。
3. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

## 使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 [CTO Chris Wright 的信息](#)。



## 第 1 章 SHENANDOAH 垃圾收集器

Shenandoah 是低暂停时间垃圾收集器(GC)，它通过与运行的 Java 程序同时执行更多的垃圾回收工作来减少 GC 暂停时间。并发 Mark Sweep 垃圾收集器(CMS)和 G1，红帽构建的 OpenJDK 21 的默认垃圾收集器执行实时对象的并发标记。

Shenandoah 添加并发压缩。Shenandoah 还通过在运行 Java 线程的同时压缩对象来减少 GC 暂停时间。带有 Shenandoah 的暂停时间独立于堆大小，这意味着您的堆是 200 MB 或 200 GB 的一致暂停时间。Shenandoah 是应用程序的一种算法，需要响应和可预测的短暂停。

### 其他资源

- 有关 Shenandoah 垃圾收集器的更多信息，请参阅 Oracle OpenJDK 文档中的 [Shenandoah GC](#)。

## 第 2 章 使用 SHENANDOAH 垃圾收集器运行 JAVA 应用程序

您可以使用 Shenandoah 垃圾收集器(GC)运行 Java 应用程序。

### 先决条件

- 安装了红帽构建的 OpenJDK。请参阅 [在 RHEL 上安装和使用红帽构建的 OpenJDK 17 中的在 Red Hat Enterprise Linux 上安装红帽构建的 OpenJDK 21。](#)

### 流程

- 使用 `-XX:+UseShenandoahGC` JVM 选项使用 Shenandoah GC 运行您的 Java 应用程序。

```
$ java <PATH_TO_YOUR_APPLICATION> -XX:+UseShenandoahGC
```

## 第 3 章 SHENANDOAH 垃圾收集器模式

您可以通过三种不同的模式运行 Shenandoah：使用 `-XX:ShenandoahGCMode=<name>` 选择特定的模式。以下列表描述了每个 Shenandoah 模式：

### normal/satb（产品，默认）

这个模式使用 Snapshot-At-The-Beginning (SATB) 标记运行并发垃圾收集器(GC)。此标记模式与 G1 类似，这是红帽构建的 OpenJDK 21 的默认垃圾收集器。

### IU（实验性）

这个模式会运行一个并发 GC with Incremental Update (IU) 标记。它可以更积极地回收内存不足。这个标记模式会镜像 SATB 模式。这可能会使标记不太保守，特别是在访问弱引用时。

### 被动(diagnostic)

这个模式运行停止 World 事件 GCs。此模式用于进行功能测试，但有时对于使用 GC 障碍的 bisecting 性能情况很有用，或者确定应用程序中的实际实时数据大小。

## 第 4 章 SHENANDOAH 垃圾收集器的基本配置选项

Shenandoah 垃圾收集器(GC)具有以下基本配置选项：

### **-Xlog:gc**

打印单个 GC 时间。

### **-Xlog:gc+ergo**

打印 Heuristics 决策，如果出现任何情况，则可能会发现问题。

### **-Xlog:gc+stats**

在运行结束时打印 Shenandoah 内部计时的摘要表。

最好在启用了日志记录后运行此操作。此概述表传达有关 GC 性能的重要信息。Heuristics 日志对找出 GC outliers 非常有用。

### **-XX:+AlwaysPreTouch**

将堆页面提交成内存，并有助于降低延迟 hiccups。

### **-Xms 和 -Xmx**

使用 **-Xms = -Xmx** 使堆不可调整大小，从而减少了堆管理困难。与 **AlwaysPreTouch** 一起，**-Xms = -Xmx** 会在启动时提交所有内存，这样可避免最终使用内存时比较困难。**-Xms** 还定义了内存 uncommit 的低边界，因此 **-Xms = -Xmx** 所有内存都保持提交。如果要为较小的空间配置 Shenandoah，则建议设置 lower **-Xms**。您需要决定设置它以平衡提交/未提交开销与内存占用量的低程度。在很多情况下，您可以设置 **-Xms** 任意低。

### **-XX:+UseLargePages**

启用 **hugetlbfs** Linux 支持。

### **-XX:+UseTransparentHugePages**

透明地启用巨页。使用透明大内存页时，建议将 **/sys/kernel/mm/transparent\_hugepage/enabled** 和 **/sys/kernel/mm/transparent\_hugepage/defrag** 设置为 **madvise**。使用 **AlwaysPreTouch** 运行时，它还将启动时支付有问题的工具成本。

### **-XX:+UseNUMA**

虽然 Shenandoah 尚不明确支持 NUMA，但最好启用对多套接字主机的 NUMA 交集。与 **AlwaysPreTouch** 相结合，它提供了优于默认的开箱即用配置的性能。

### **-XX:-UseBiasedLocking**

非连续（双插槽）锁定吞吐量之间有一个利弊，而安全点 JVM 则用于启用和禁用它们。对于面向延迟的工作负载，请关闭计费锁定。

### **-XX:+DisableExplicitGC**

从用户代码调用 **system.gc ()** 会强制 Shenandoah 执行额外的 GC 周期。它通常不会损害，因为 **-XX:+ExplicitGCInvokesConcurrent** 会被默认启用，这意味着会调用并发 GC 周期，而不是 STW Full GC。

更新于 2024-05-10

