# 毕昇JDK8和11首次同时发布Aarch64和X86_64两个版本

2021年9月30日，毕昇JDK update Q3版本正式发布，本次发布将包含X86_64版本。此前，毕昇JDK只发布Aarch64版本，这可能会对运维产生一定的影响，例如需要根据架构构建多个版本以包含不同架构的JDK，此次毕昇JDK同时发布X86_64版本以及Aarch64版本，将极大的方便用户进行构建，降低维护多个版本的开销。另外，X86_64版本和Aarch64版本共源，所以X86_64版本也包含此前毕昇JDK团队在Aarch64上的功能和大部分优化，在功能和性能方面，两者几乎无差异。欢迎用户安装使用，为产品带来核心竞争力。

此次版本在同步OpenJDK社区8u302/11.0.12的基础上，还包含如下更新，为用户提供高性能、可用于生产环境的OpenJDK发行版。

1. PS优化——Introduce UsePSRelaxedForwardee to enable using relaxed CAS in copy_to_survivor_space(毕昇JDK8，毕昇JDK11)
2. G1 GC 优化——Parallel Full GC for G1(毕昇JDK8)
3. 提供鲲鹏硬件加速的KAEProvider(毕昇JDK11)
4. 支持按进程id和时间戳生成 jfr 文件（毕昇JDK8，毕昇JDK11)
5. Bug fixes

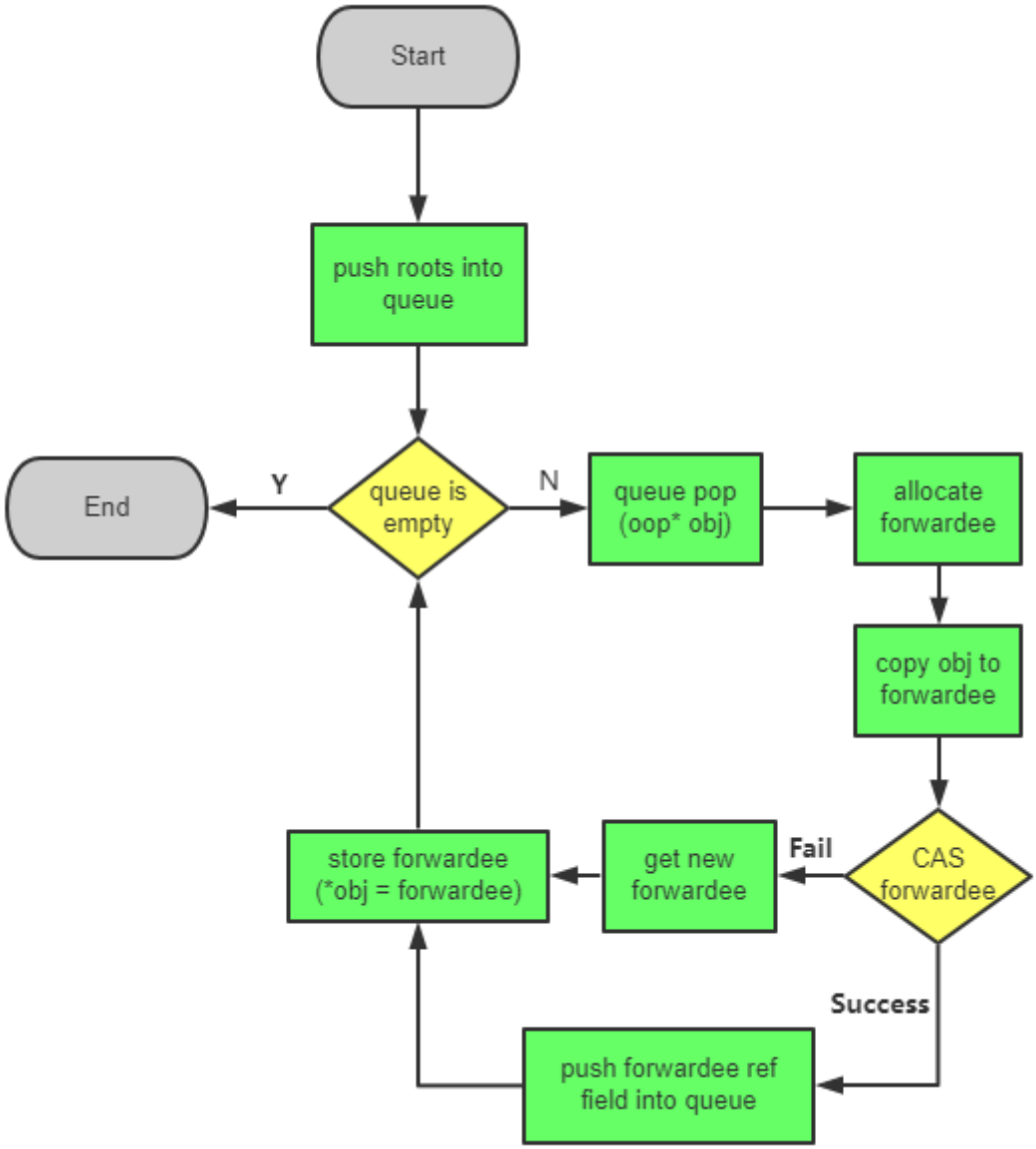
# PS:introduce UsePSRelaxedForwardee to enable using relaxed CAS in copy_to_survivor_space(毕昇JDK8，毕昇JDK11)

## 背景

在JDK中Parallel Scavenge是一个高吞吐量GC，使用非常广泛。在specjbb测试中，PSPromotionManager::copy_to_survivor_space中的CAS指令CPU占比非常高，主要为release barrier导致，分析PS逻辑后，CAS没必要使用memory barrier，使用relaxed可以提高弱内存模型架构上PS的性能。

## 实现原理

PS的主要逻辑如下：




（1）由上述流程图可以看到，CAS Fail的线程不会去读forwardee内容，因此在弱内存模型的CPU架构上，即使copy obj和CAS乱序，也不会影响CAS Fail线程的正确性。

（2）关于work steal场景，其他线程steal到的obj能否看到其内容，这个是由CAS成功的push操作保证的，由于push操作底层实现有release语义，所以无正确性问题。

```
template<class E, MEMFLAGS F, unsigned int N> inline bool
GenericTaskQueue<E, F, N>::push(E t) {
  uint localBot = _bottom;
  assert(localBot < N, "_bottom out of range.");
  idx_t top = _age.top();
  uint dirty_n_elems = dirty_size(localBot, top);
  assert(dirty_n_elems < N, "n_elems out of range.");
  if (dirty_n_elems < max_elems()) {
    // g++ complains if the volatile result of the assignment is
    // unused, so we cast the volatile away.  We cannot cast directly
    // to void, because gcc treats that as not using the result of the
    // assignment.  However, casting to E& means that we trigger an
    // unused-value warning.  So, we cast the E& to void.
    (void) const_cast<E&>(_elems[localBot] = t);
    OrderAccess::release_store(&_bottom, increment_index(localBot));
    TASKQUEUE_STATS_ONLY(stats.record_push());
    return true;
  } else {
    return push_slow(t, dirty_n_elems);
  }
}
```

```
template<class E, MEMFLAGS F, unsigned int N>
bool GenericTaskQueue<E, F, N>::push_slow(E t, uint dirty_n_elems) {
  if (dirty_n_elems == N - 1) {
    // Actually means 0, so do the push.
    uint localBot = _bottom;
    // g++ complains if the volatile result of the assignment is
    // unused, so we cast the volatile away.  We cannot cast directly
    // to void, because gcc treats that as not using the result of the
    // assignment.  However, casting to E& means that we trigger an
    // unused-value warning.  So, we cast the E& to void.
    (void)const_cast<E&>(_elems[localBot] = t);
    OrderAccess::release_store(&_bottom, increment_index(localBot));
    TASKQUEUE_STATS_ONLY(stats.record_push());
    return true;
  }
  return false;
}
```

使用参数：

UsePSRelaxedForwardee： 试验特性开关，默认为false，表示PSPromotionManager::copy_to_survivor_space中CAS forwardee使用release语义；打开则表示CAS forwardee的时候使用relaxed（无任何memory barrier），以在弱内存模型CPU架构上获取更好性能。

## 性能测试

测试环境：

Architecture:
aarch64
Byte Order: Little Endian
CPU(s): 128
On-line CPU(s) list: 0-127
Thread(s) per core: 1
Core(s) per socket: 64

Socket(s): 2
NUMA node(s): 4
Vendor ID: 0x48
Model: 0
Stepping: 0x1
BogoMIPS: 200.00
L1d cache: 64K
L1i cache: 64K
L2 cache: 512K
L3 cache: 65536K
NUMA node0 CPU(s): 0-31
NUMA node1 CPU(s): 32-63
NUMA node2 CPU(s): 64-95
NUMA node3 CPU(s): 96-127

使用specjbb2015进行测试，除UsePSRelaxedForwardee开关以外的测试参数如下：

-Xms50g -Xmx50g -XX:+UseParallelGC -XX:ParallelGCThreads=24 -XX:+UseLargePages -XX:LargePageSizeInBytes=2m -XX:+UseBiasedLocking -XX:+AlwaysPreTouch -XX:-UseAdaptiveSizePolicy

测试结果：

| | BiSheng JDK11.0.12 基线 | | BiSheng JDK11.0.12 PS优化 | |
|---|---|---|---|---|
| | max | critical | max | critical |
| 第一组数据 | 131580 | 25230 | 133225 | 32320 |
| 第二组数据 | 131580 | 24953 | 134869 | 32073 |
| 第三组数据 | 131580 | 24900 | 133225 | 32138 |
| 平均值 | 131580 | 25027.66667 | 133773 | 32177 |
| 提升 | 0 | 0 | 0.016666667 | 0.285657206 |
| | BiSheng JDK8u302 基线 | | BiSheng JDK8u302 PS优化 | |
| | max | critical | max | critical |
| 第一组数据 | 134869 | 25204 | 136514 | 29210 |
| 第二组数据 | 136514 | 25018 | 136514 | 29282 |
| 第三组数据 | 136514 | 25844 | 136514 | 29629 |
| 平均值 | 135965.6667 | 25355.33333 | 136514 | 29373.66667 |
| 提升 | 0 | 0 | 0.004032881 | 0.158480793 |

测试结果：从上图可以看到，针对SPECjbb的critical，毕昇JDK8可以提升15%，毕昇JDK11可以提升28%

# Parallel Full GC for G1. (毕昇JDK8)

## 概述

G1 Full GC是完全的STW，在此期间应用程序线程完全没有机会运行，长时间停顿会造成用户明显的感知。因此，使用G1过程中应尽量避免的Full GC的出现，如果出现最好能缩短其时间。当前JDK 8u中G1 Full GC完全采用串行，包括：

- 各阶段之间，包括标记存活对象、计算目标对象的位置、更新引用的位置、移动对象完成压缩阶段；
- 每个阶段内；

完全的串行导致即使是在多核机器上也无法利用机器的强大性能缩短Full GC的（停顿）时间。

由于G1 Full GC基本算法的约束，虽然上面提到的四个阶段之间无法并行化，但是各个阶段内却可以通过优化算法做到一定并行化，以达到缩短整体停顿时间的效果。本特性会将计算目标对象的位置、更新引用的位置、移动对象完成压缩三个阶段尽量做到阶段内的并行化。（标记存活对象阶段的并行化后续也会支持）

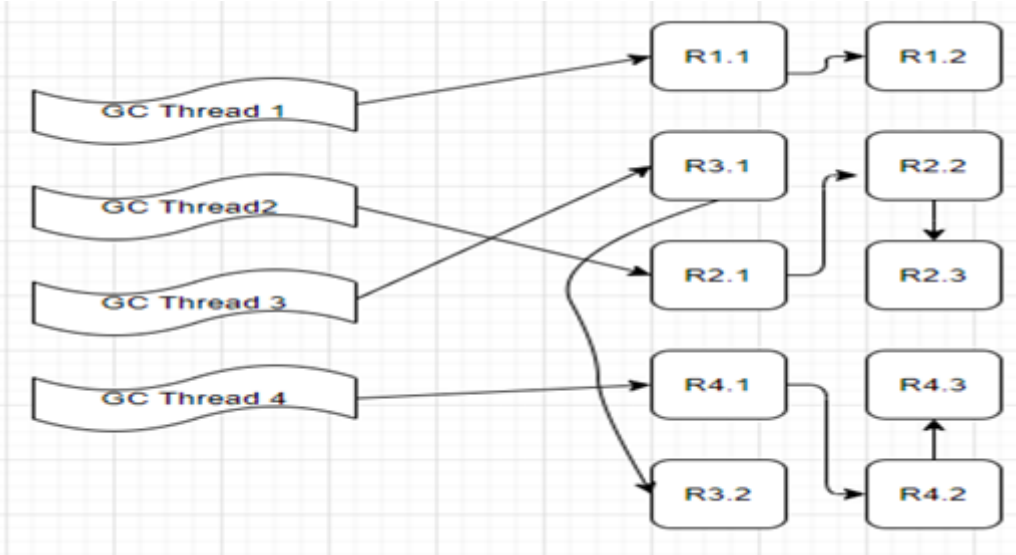
开启本特性后，可以明显降低G1 Full GC的平均停顿时间。本特性属于通用特性，适用于Aarch64、X86平台。

# 实现原理

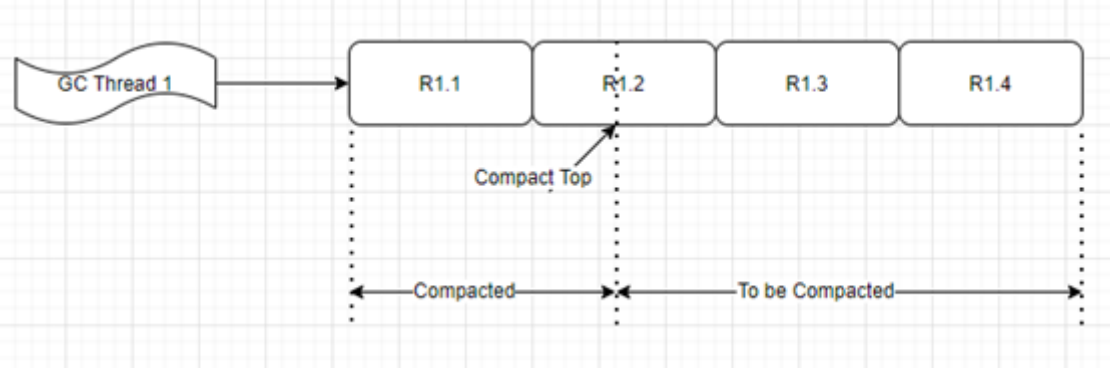
## 并行Full GC基本算法

如下列出了并行Full GC算法与串行Full GC算法的主要差异点：

1. 将整个堆分成不同的heap region set交给各个GC线程分别处理，尽量减少GC线程间同步、竞争；




2. G1 Full GC现有实现是将整个堆向一个方向（目标地址）压缩；要做到并行化，并减少并行GC线程间的交互、竞争，有效的方式是每个GC线程有自己压缩的方向（目标地址）。




3. 大对象的特殊处理：在计算目标对象位置并行阶段结束后，才能释放free的humongous region；

## 计算目标对象位置阶段的并行化

计算目标对象位置阶段主要负责

1）根据标记信息设置对象的forwardee。

2）释放没有被标记的humongous regions。

Forwardee的设置需要预先知道目标地址，该目标地址是通过Compaction Point维护着。在遍历heap region时每当发现一个新的标记的对象，就将Compaction Point里记录的目标地址设置为该对象的forwardee，然后再将Compaction Point里记录的目标地址加上对象的大小，作为下次forwardee设置的值。如此往复，直至每一个标记的对象都被forwarded。

并行地设置对象的Forwardee是通过1）隔离各个GC线程的遍历的heap region，2）隔离各个GC线程要为forwardee设置的目标地址来达成的。具体实现是，1）通过标记 region来隔离各个GC线程遍历的heap regions，2）通过为每个GC线程维护一个Compaction Point来隔离forwardee的设置。可以理解为将整个heap被分成了N份（GC线程个数为N），每一份由一个GC线程负责，各个线程尽量互不干扰地工作。

除此之外，每个GC线程的Compaction Point还负责收集属于该GC线程的regions、humongous regions，以便后续（压缩阶段）处理。

Free的大对象在计算目标对象位置阶段就会被释放。由于大对象的特殊性（可能包括多个heap region）加之多个GC线程在同时工作，需要对其进行一些特殊处理：如，在计算目标对象位置并行阶段结束后，才能释放free的humongous region，以避免多个GC线程访问同一个大对象的不同region时可能面临的数据不一致问题。

### 更新引用位置阶段的并行化

更新引用位置阶段主要负责根据对象的forwardee信息更新所有引用。

此阶段的并行化比较简单，因为需要的所有信息都只在对象头中（forwardee），并行化和串行化的算法差别很小，不同点只是每个GC线程要标记属于自己处理范围的heap region。

### 移动对象完成压缩阶段的并行化

移动对象完成压缩阶段主要负责根据对象的forwardee信息进行压缩。

每个GC线程都有属于自己的Compaction Point，在计算目标对象位置阶段Compaction Point中收集了需要该GC线程压缩的region的集合。对于单个GC线程来说，整个过程与串行差别不大，只是需要从自己的Compaction Point中取出regions，进行压缩。

使用参数：

本特性需要通过VM option -XX:+G1ParallelFullGC显示打开，默认为关闭。

注意，本特性会带来如下JVM停顿时间上的收益：

1、 降低单次G1 Full GC的停顿时间；

2、 降低总的G1 Full GC的停顿时间；

但是，有可能会增加G1 Full GC的频率。所以，当降低JVM的停顿时间是应用程序的性能调优目标之一时，且G1 Full GC是停顿原因之一时，适用于打开G1ParallelFullGC VM Option，降低单次平均、总的停顿时间。

## 性能测试

测试套：Dacapo

测试参数：

- JVM：-Xmx1g -Xms1g -XX:ParallelGCThreads=$N
- Dacapo：-t 4 --iterations 5 --size huge --no-pre-iteration-gc h2

下面分别给出了并行GC线程数量分别为4、16时Full GC停顿时间的数据

- N == 4

| baseline | percentile | ParallelFullGC | improvement |
|---|---|---|---|
| 7.269909936 | 0.99 | 5.62585249 | 22.61% |
| 7.08191168 | 0.9 | 5.50645174 | 22.25% |
| 6.97446122 | 0.8 | 5.40685576 | 22.48% |
| 6.7997374 | 0.7 | 5.26532902 | 22.57% |
| 6.7104115 | 0.6 | 5.13041504 | 23.55% |
| 6.5305017 | 0.5 | 4.9761024 | 23.80% |
| 6.3764317 | 0.4 | 4.82261604 | 24.37% |
| 6.14229808 | 0.3 | 4.54988018 | 25.93% |
| 4.9476876 | 0.2 | 3.64964444 | 26.24% |
| 4.56323086 | 0.1 | 3.4157966 | 25.15% |

- N == 16

| baseline | percentile | ParallelFullGC | improvement |
|---|---|---|---|
| 6.556812319 | 0.99 | 5.496492769 | 16.17% |
| 6.06999569 | 0.9 | 4.75083408 | 21.73% |
| 5.88646958 | 0.8 | 3.59663782 | 38.90% |
| 5.83802396 | 0.7 | 3.55000364 | 39.19% |
| 5.78194332 | 0.6 | 3.52787632 | 38.98% |
| 5.7400595 | 0.5 | 3.5005536 | 39.02% |
| 5.52639954 | 0.4 | 3.25090818 | 41.17% |
| 4.9687052 | 0.3 | 3.18859845 | 35.83% |
| 4.264625 | 0.2 | 2.92987018 | 31.30% |
| 4.08533252 | 0.1 | 2.42765843 | 40.58% |

测试结果：受益（STW时间减少）基本在16%~40%。

# 提供鲲鹏硬件加速的KAEProvider(毕昇JDK11)

该特性已在早期的毕昇JDK 8u282中支持，详见2021年毕昇 JDK 的第一个重要更新来了，并在8u292版本中对其功能进行完善，详见毕昇 JDK 8u292、11.0.11 发布！，此次将在毕昇JDK11中对该特性进行支持。

## 实现原理和性能测试

实现原理和性能测试请参考鲲鹏硬件加解密特性详解. 但由于JDK11引入了模块系统，因此用户使用时需要将KAEProvider所在的模块(jdk.crypto.kaeprovider)进行导出，如下为毕昇JDK11中KAEProvider相关的文件：

```
[hedongbo@jvmtest-59 jdk]$ find . -name "*kae*"
./jmods/jdk.crypto.kaeprovider.jmod
./lib/libj2kae.so
./lib/libj2kae.debuginfo
./legal/jdk.crypto.kaeprovider
./conf/kaeprovider.conf
```

具体导出命令可参考如下格式：

```
编译：javac  --add-modules jdk.crypto.kaeprovider  --add-
exports=jdk.crypto.kaeprovider/org.openeuler.security.openssl=ALL-UNNAMED
 DHTest.java
运行：java  --add-modules jdk.crypto.kaeprovider  --add-
exports=jdk.crypto.kaeprovider/org.openeuler.security.openssl=ALL-UNNAMED
 DHTest
```

# 支持按进程id和时间戳生成 jfr 文件（毕昇JDK8，毕昇JDK11)

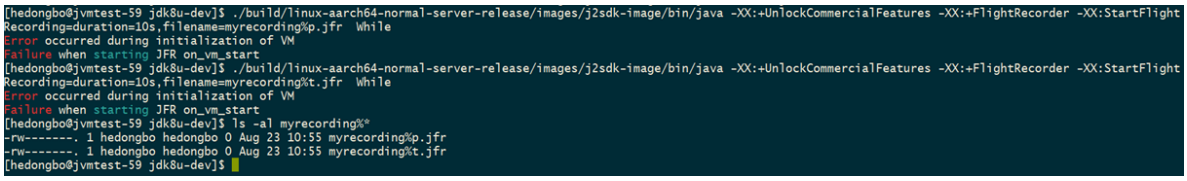
## 说明

该特性用来扩展JFR文件名，支持在文件名中加入进程号或时间戳或两者都加，当用户在环境上生成多个jfr文件时，该特性可以帮助用户根据需要快速定位到所需的文件。

## 功能测试

未合入此特性:

java -XX:+UnlockCommercialFeatures -XX:+FlightRecorder -XX:StartFlightRecording=duration=10s,filename=myrecording%t.jfr  While




合入此特性:

java -XX:+UnlockCommercialFeatures -XX:+FlightRecorder -XX:StartFlightRecording=duration=10s,filename=myrecording%t.jfr  While


# Bug fixes

除了上面介绍的一些特性外，毕昇JDK还合入了社区高版本中的一些bug fix和优化的patch，为用户提供稳定、高性能的毕昇JDK。具体回合patch如下：

- JDK8
  - 8197387：jcmd started by "root" must be allowed to access all VM processes  允许通过root启动的jcmd访问环境上任意的JVM进程，默认情况下，进程只能被启动该进程的用户通过jcmd访问。
  - 8069191：moving predicate out of loops may cause array accesses to bypass null check  修复c2 在aarch64上可能会crash的bug
  - 8167014: jdeps: Missing message: warn.skipped.entry  该修复可以解决通过jdeps解析特定的jar包出现的Missing message: warn.skipped.entry错误
  - 8268453: sun/security/pkcs12/EmptyPassword.java fails with Sequence tag error  该修复可以解决当对密码为空的KeyStore进行解析时，可能会出现的java.io.IOException: Sequence tag error问题

    - 8202142：jfr/event/io/TestInstrumentation is unstable JDK自带用例修复
    - 8143251：HeapRetentionTest.java Test is failing on jdk9/dev 该修复可以解决G1 GC在特定场景下导致进程假死的问题
    - 8183543：Aarch64: C2 compilation often fails with "failed spill-split-recycle sanity check" 修复C2编译器在某些场景下编译方法时报`failed spill-split-recycle sanity check`错误，导致方法被解释执行，进而造成应用程序性能下降的问题
- JDK11
    - 8268427: Improve AlgorithmConstraints:checkAlgorithm performance 该patch可以提升TLS的握手性能
    - 8257145: Performance regressionwith -XX:-ResizePLABafter JDK-8079555 该patch可以修复使用G1 GC后，HBase性能下降的问题，详细原理可参考毕昇JDK以前的文章JDK 从8升级到11，使用 G1 GC，HBase 性能下降近20%。JDK 到底干了什么？
    - 8247691：[aarch64] Incorrect handling of VM exceptions in C1 deopt stub/traps 该修复可以解决C1 编译器生成指令过程中使用错误的寄存器，进而导致进程Crash的问题

# 参考


[1] 毕昇JDK8 aarch64下载：https://mirrors.huaweicloud.com/kunpeng/archive/compiler/bisheng_jdk/bisheng-jdk-8u302-linux-aarch64.tar.gz

[2] 毕昇JDK8 x86_64下载：https://mirrors.huaweicloud.com/kunpeng/archive/compiler/bisheng_jdk/bisheng-jdk-8u302-linux-x64.tar.gz

[3] 毕昇JDK11 aarch64下载：https://mirrors.huaweicloud.com/kunpeng/archive/compiler/bisheng_jdk/bisheng-jdk-11.0.12-linux-aarch64.tar.gz

[4] 毕昇JDK11 x86_64下载：https://mirrors.huaweicloud.com/kunpeng/archive/compiler/bisheng_jdk/bisheng-jdk-11.0.12-linux-x64.tar.gz


# 交流群：

欢迎加入 Compiler SIG 交流群，一起交流编译器、虚拟机技术。或添加微信：pengchenghan99，回复"加群"，进入 Compiler SIG 交流群。

openEuler Compiler SIG

该二维码7天内(10月27日前)有效，重新进入将更新