超实用的
Java14新特性

讲师：宋红康
新浪微博：尚硅谷-宋红康

目录

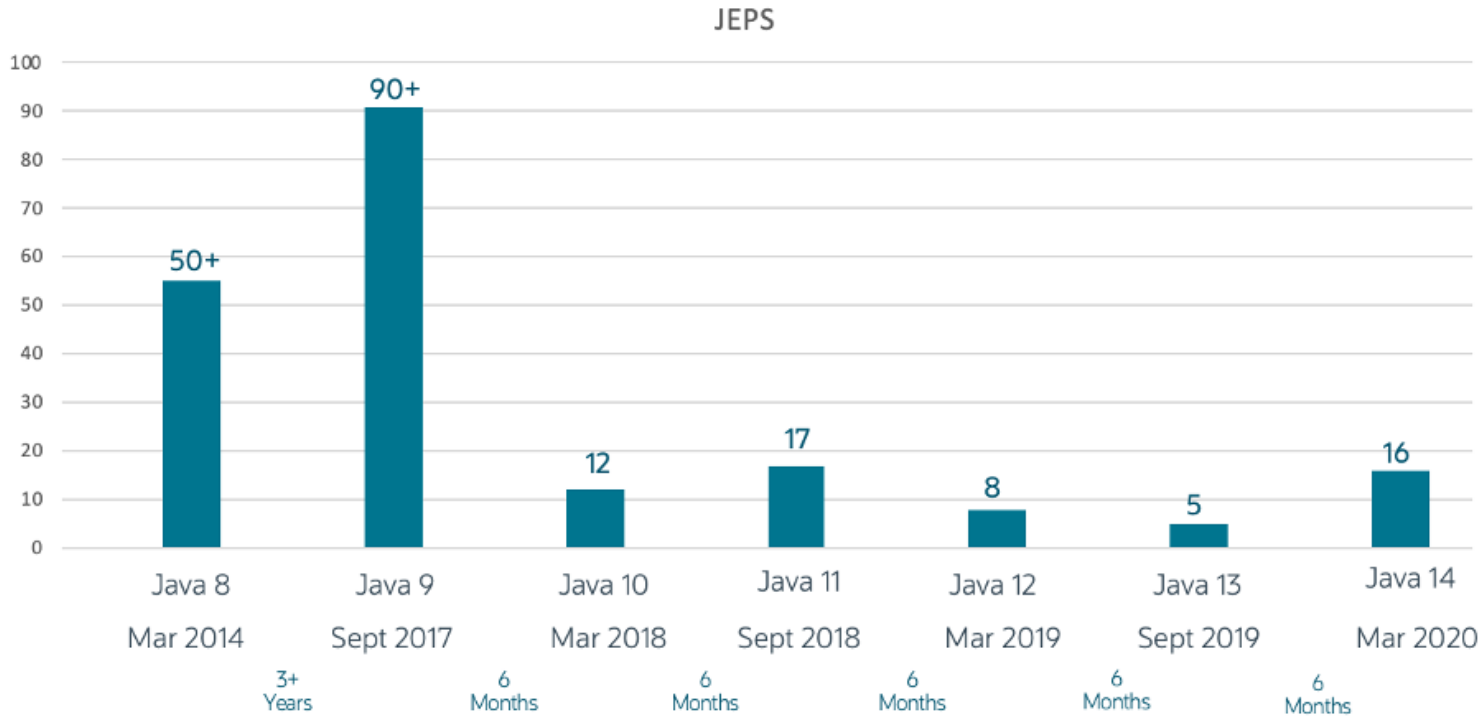
尚硅谷

从哪几个角度学习新特性

- 语法层面：lambda表达式，switch，自动装箱、自动拆箱，enum，<>，接口中的默认方法、静态方法、私有方法
- API层面：Stream API、新的日期时间、Optional、String、集合框架
- 底层优化：JVM的优化，元空间，GC，GC的参数，js的执行引擎

1- Java 14 新特性概述

2020年 3 月 17 日，JDK/Java 14 正式 GA(General Available)。这是自从Java采用六个月一次的发布周期之后的第五次发布。



Java14发布，16大新特性，代码更加简洁明快

此版本包含的 JEP（Java/JDK Enhancement Proposals，JDK 增强提案）比 Java 12 和 13 加起来的还要多。总共 16 个新特性，包括两个孵化器模块、三个预览特性、两个弃用的功能以及两个删除的功能。

- “孵化器模块”：将尚未定稿的API和工具先交给开发者使用，以获得反馈，并用这些反馈进一步改进Java平台的质量。

- “预览特性”：是规格已经成型、实现已经确定，但还未最终定稿的功能。它们出现在Java中的目的是收集在真实世界中使用后的反馈信息，促进这些功能的最终定稿。这些特性可能会随时改变，根据反馈结果，这些特性甚至可能会被移除，但通常所有预览特性最后都会在Java中固定下来。

JDK 14

JDK 14 is the open-source reference implementation of version 14 of the Java SE Platform as specified by by JSR 389 in the Java Community Process.

JDK 14 reached General Availability on 17 March 2020. Production-ready binaries under the GPL are available from Oracle; binaries from other vendors will follow shortly.

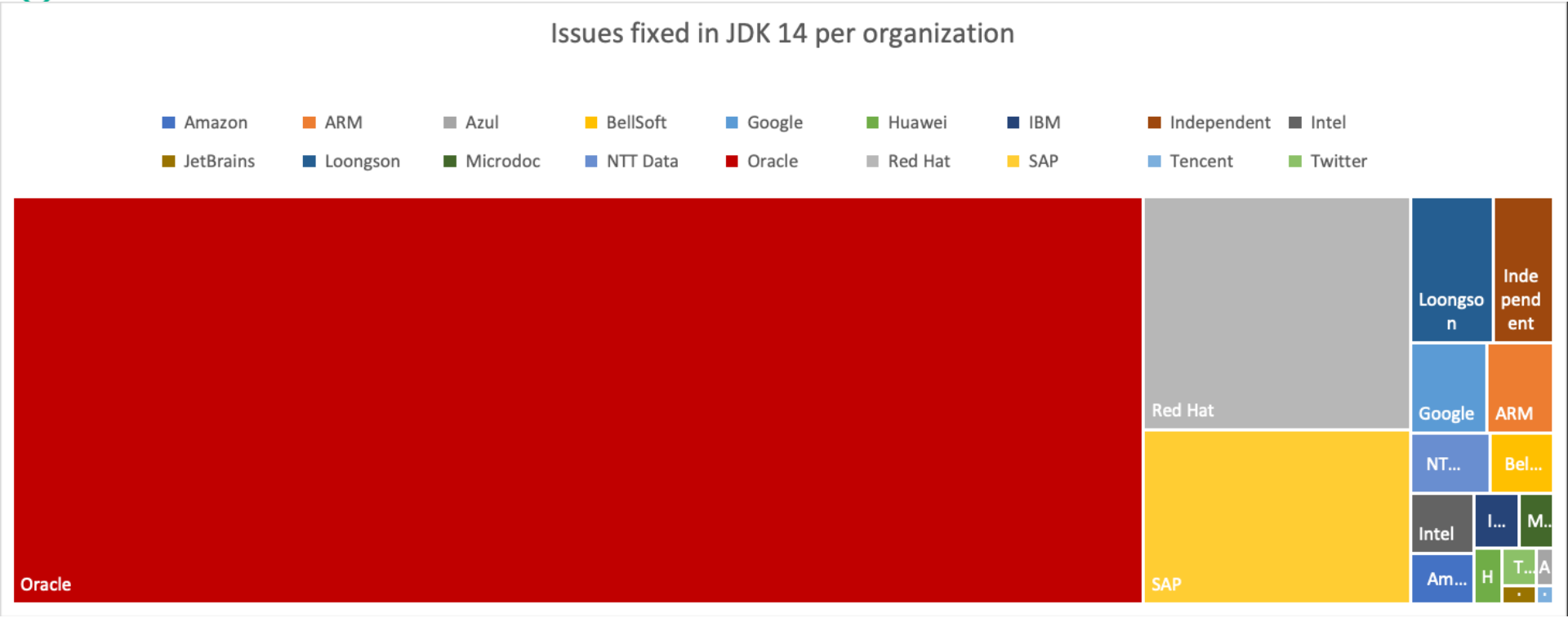
The features and schedule of this release were proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal. The release was produced using the JDK Release Process (JEP 3).

Schedule

2019/12/12	Rampdown Phase One (fork from main line)
2020/01/16	Rampdown Phase Two
2020/02/06	Initial Release Candidate
2020/02/20	Final Release Candidate
2020/03/17	General Availability

Features

305: Pattern Matching for instanceof (Preview)
343: Packaging Tool (Incubator)
345: NUMA-Aware Memory Allocation for G1
349: JFR Event Streaming
352: Non-Volatile Mapped Byte Buffers
358: Helpful NullPointerExceptions
359: Records (Preview)
361: Switch Expressions (Standard)
362: Deprecate the Solaris and SPARC Ports
363: Remove the Concurrent Mark Sweep (CMS) Garbage Collector
364: ZGC on macOS
365: ZGC on Windows
366: Deprecate the ParallelScavenge + SerialOld GC Combination
367: Remove the Pack200 Tools and API
368: Text Blocks (Second Preview)
370: Foreign-Memory Access API (Incubator)



此次的发布与之前的Java 11、12和13一样，离不开OpenJDK社区无数个人和组织的无私奉献。JDK 14修正了1986个JIRA问题，其中1458个来自Oracle的员工，另外528个来自独立开发者和其他公司的开发者的提交。

305: Pattern Matching for instanceof (Preview)
343: Packaging Tool (Incubator)
345: NUMA-Aware Memory Allocation for G1
349: JFR Event Streaming
352: Non-Volatile Mapped Byte Buffers
358: Helpful NullPointerExceptions
359: Records (Preview)
361: Switch Expressions (Standard)
362: Deprecate the Solaris and SPARC Ports
363: Remove the Concurrent Mark Sweep (CMS) Garbage Collector
364: ZGC on macOS
365: ZGC on Windows
366: Deprecate the ParallelScavenge + SerialOld GC Combination
367: Remove the Pack200 Tools and API
368: Text Blocks (Second Preview)
370: Foreign-Memory Access API (Incubator)

Mark Reinhold

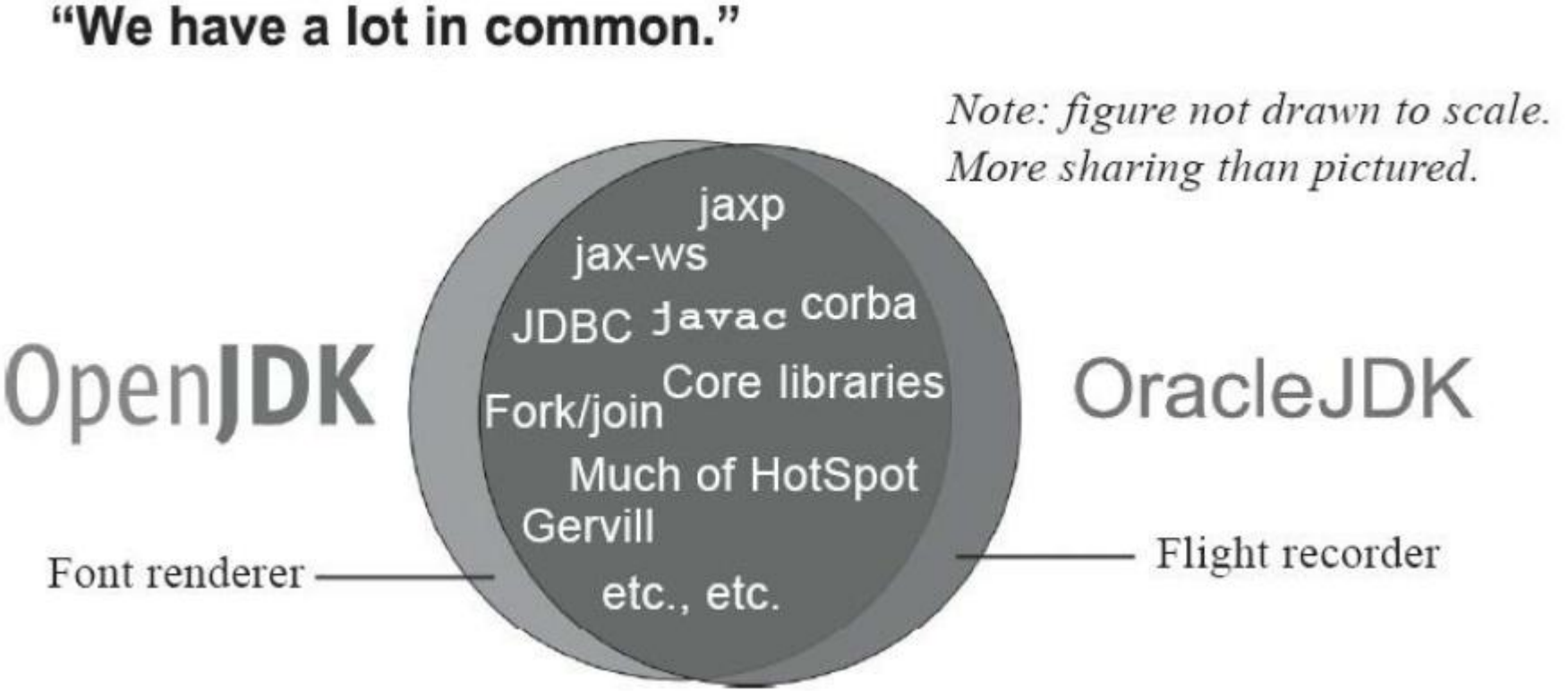
他是JDK 1.2和5.0版本的首席工程师，Java SE 6的规范制定的负责人，还是JDK 7，JDK 8和JDK 9 项目和规范的负责人，目前，他主要在OpenJDK社区领导JDK项目

版本	发布日期	最终免费公开更新时间[3][4]	最后延伸支持日期
JDK Beta	1995	?	?
JDK 1.0	1996 年 1 月	?	?
JDK 1.1	1997 年 2 月	?	?
J2SE 1.2	1998 年 12 月	?	?
J2SE 1.3	2000 年 5 月	?	?
J2SE 1.4	2002 年 2 月	2008 年 10 月	2013 年 2 月
J2SE 5.0	2004 年 9 月	2009 年 11 月	2015 年 4 月
Java SE 6	2006 年 12 月	2013 年 4 月	2018 年 12 月
Java SE 7	2011 年 7 月	2015 年 4 月	2022 年 7 月
Java SE 8 (LTS)	2014 年 3 月	Oracle 于 2019 年 1 月停止更新（商用） Oracle 于 2020 年 12 月停止更新（非商用） AdoptOpenJDK 于 2023 年 9 月或之前停止更新 Amazon Corretto 于 2023 年 6 月或之前停止更新	2030 年 12 月
Java SE 9	2017 年 9 月	OpenJDK 于 2018 年 3 月停止更新	不适用
Java SE 10	2018 年 3 月	OpenJDK 于 2018 年 9 月停止更新	不适用
Java SE 11 (LTS)	2018 年 9 月	Amazon Corretto 于 2024 年 8 月或之前停止更新 AdoptOpenJDK 于 2022 年 9 月停止更新	2026 年 9 月
Java SE 12	2019 年 3 月	OpenJDK 于 2019 年 9 月停止更新	不适用
Java SE 13	2019 年 9 月	OpenJDK 于 2020 年 3 月停止更新	不适用
Java SE 14	2020 年 3 月	OpenJDK 于 2020 年 9 月停止更新	不适用
Java SE 15	2020 年 9 月	OpenJDK 于 2021 年 3 月停止更新	不适用
Java SE 16	2021 年 3 月	OpenJDK 于 2021 年 9 月停止更新	不适用
Java SE 17 (LTS)	2021 年 9 月	待定	待定

格式： 旧版本 旧版本，仍被支持 当前版本 未来版本

回顾旧版本，展望新版本

Oracle JDK 与 Open JDK



Oracle JDK 不再免费提供。但是，你现在可以从包括 Oracle 在内的各种供应商获得免费的 OpenJDK 发行版。

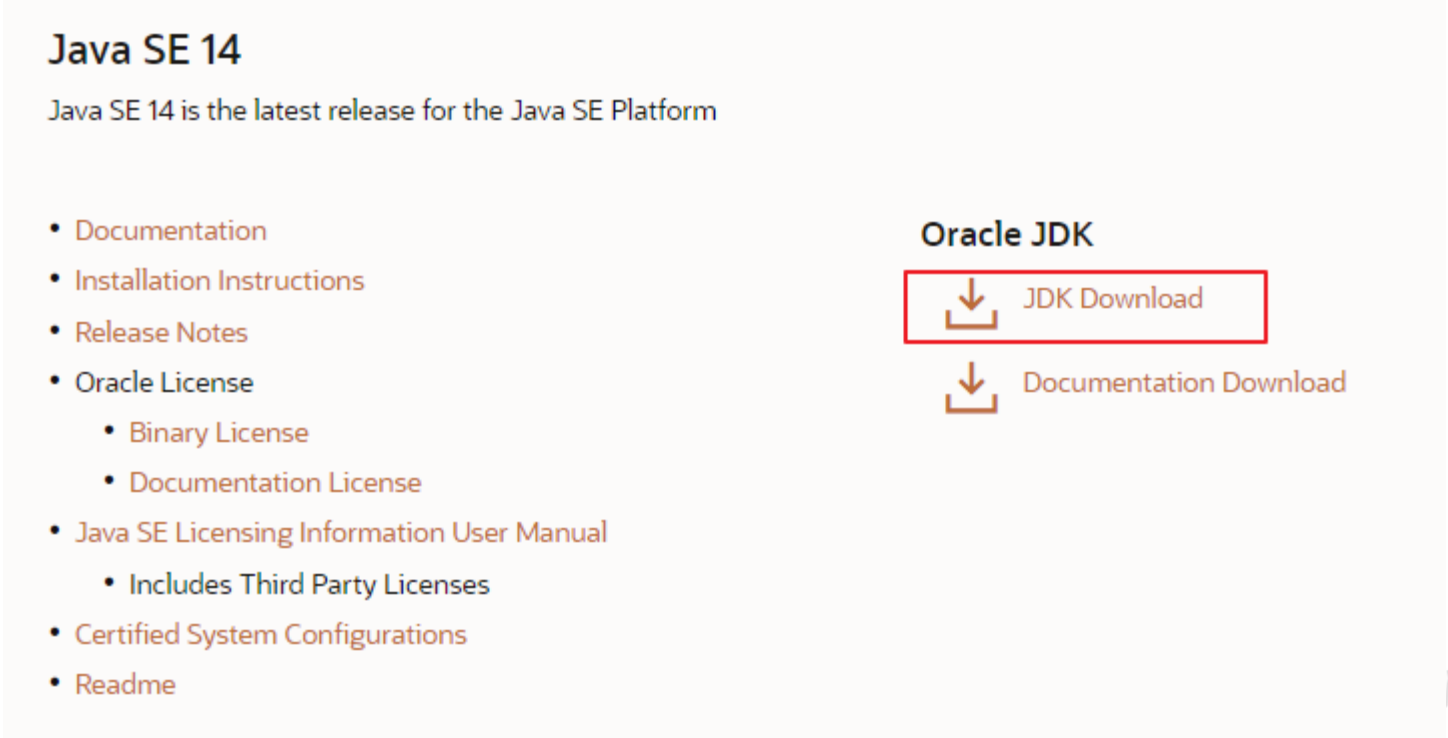
“语言必须发展，否则它们就有变得无关紧要的风险。” Brian Goetz (甲骨文公司)在2019年11月在 Devoxx 举行的“ Java 语言期货”演讲中说。

尽管 Java 已经发展了25年，但仍然远远没有过时。下面，我们将研究 JDK 14的创新。

小步快跑，快速迭代

2- 环境安装

- **JDK14的下载、安装**
  - https://www.oracle.com/java/technologies/javase-downloads.html



- **IDEA2020.1的下载、安装、配置**
  - https://www.jetbrains.com/idea/nextversion/#section=windows

3- 超实用新特性

1. JEP 305：instanceof的模式匹配（预览）

这个特性很有意思，因为它为更为通用的模式匹配打开了大门。模式匹配通过更为简便的语法基于一定的条件来抽取对象的组件，而 instanceof 刚好是这种情况，它先检查对象类型，然后再调用对象的方法或访问对象的字段。

有了该功能，可以减少Java程序中显式强制转换的数量，从而提高生产力，还能实现更精确、简洁的类型安全的代码。

```
if (obj instanceof String) {
    String str = (String) obj; // 需要强转
    .. str.contains(..) ..
}else{
     str = ....
}
```

Java 14之前

```
if (!(obj instanceof String str)) {
    .. str.contains(..) .. // 不再需要转换代码，实际发生了转换
} else {
    .. str....
}
```

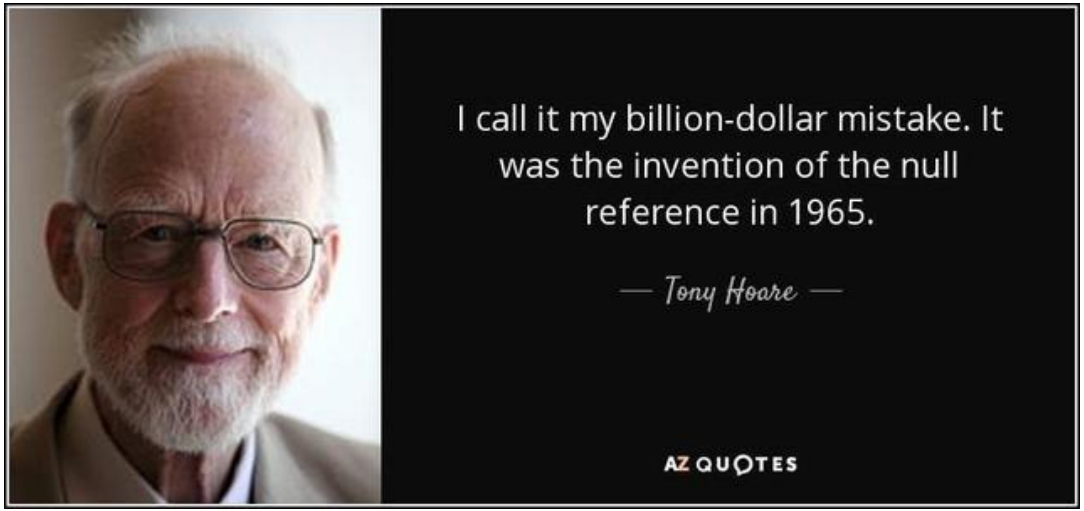
Java 14新特性

2. JEP 358：非常实用的NullPointerException

该特性改进了NullPointerException的可读性，能更准确地给出null变量的信息。

该特性可以帮助开发者提高生产力，以及改进各种开发工具和调试工具的质量。一个目标是减少开发人员的困惑和担忧 。

null何错之有？



2. JEP 358：非常实用的NullPointerException

相信很多Java程序员都一样对null和NPE深恶痛绝，因为他确实会带来各种各样的问题（来自《Java 8 实战》）。如：

- 它是错误之源。 NullPointerException是目前Java程序开发中最典型的异常。它会使你的代码膨胀。
- 它让你的代码充斥着深度嵌套的null检查，代码的可读性糟糕透顶。
- 它自身是毫无意义的。 null自身没有任何的语义，尤其是它代表的是在静态类型语言中以一种错误的方式对缺失变量值的建模。
- 它破坏了Java的哲学。 Java一直试图避免让程序员意识到指针的存在，唯一的例外是:null指针。
- 它在Java的类型系统上开了个口子。 null并不属于任何类型，这意味着它可以被赋值给任意引用类型的变量。这会导致问题， 原因是当这个变量被传递到系统中的另一个部分后，你将无法获知这个null变量最初赋值到底是什么类型。

- 在Groovy中使用安全导航操作符（Safe Navigation Operator）可以访问可能为null的变量：

```
def carInsuranceName = person?.car?.insurance?.name
```

- 在Haskell和Scala也有类似的替代品，如Haskell中的Maybe类型、Scala中的Option[A]。Option[A] 是一个类型为 A 的可选值的容器

- 在 Kotlin 中，其类型系统严格区分一个引用可以容纳 null 还是不能容纳。也就是说，一个变量是否可空必须显示声明，对于可空变量，在访问其成员时必须做空处理，否则无法编译通过：

```
var a: String = "abc"
a = null // 编译错误

如果允许为空，可以声明一个可空字符串，写作 String?
var b: String? = "abc" //String? 表示该 String 类型变量可为空
b = null // 编译通过
```

- 首先在Java 8中提供了Optional。
  - Optional在可能为null的对象上做了一层封装，强制你思考值不存在的情况，这样就能避免潜在的空指针异常。
  - 关于Optional的用法，不是本文的重点。在日常开发中经常结合Stream一起使用Optional，还是比较好用的。

- 另外一个值得一提的就是最近（2020年03月17日）发布的JDK 14中对于NPE有了一个增强。那就是JEP 358: Helpful NullPointerExceptions
  - 该特性可以更好地提示哪个地方出现的空指针，需要通过

    -XX:+ShowCodeDetailsInExceptionMessages开启
  - 在未来的版本中，这个特性可能会默认启用。
  - 这个增强特性不仅适用于方法调用，只要会导致 NullPointerException 的地方也都适用，包括字段的访问、数组的访问和赋值。

官方吐槽最为致命

早在2019年2月份，Java 语言架构师 Brian Goetz，曾经写过一篇文章，详尽的说明了并吐槽了Java语言，他和很多程序员一样抱怨“Java太啰嗦”或有太多的“繁文缛节”，他提到：开发人员想要创建纯数据载体类（plain data carriers）通常都必须编写大量低价值、重复的、容易出错的代码。如：构造函数、getter/setter、equals()、hashCode()以及toString()等。

以至于很多人选择使用IDE的功能来自动生成这些代码。还有一些开发会选择使用一些第三方类库，如Lombok等来生成这些方法，从而会导致了令人吃惊的表现（surprising behavior）和糟糕的可调试性（poor debuggability）。

神说要用record，于是就有了

- 我们有时候需要编写许多低价值的重复代码来实现一个简单的数据载体类：构造函数，访问器，equals()，hashCode ()，toString ()等。为了避免这种重复代码，Java 14推出record。

- **Java14也许最令人兴奋，同时也是最令人惊讶的创新就是：Record类型的引入 。**

- 使用record来减少类声明语法，效果类似 lombok 的 @Data 注解，Kotlin中的data class。**它们的共同点是类的部分或全部状态可以直接在类头中描述**，并且这个类中只包含了纯数据而已。

- 该预览特性提供了一种更为紧凑的语法来声明类。值得一提的是，该特性可以大幅减少定义类似数据类型时所需的样板代码

- 当你用**record** 声明一个类时，该类将自动拥有以下功能：
  - 获取成员变量的简单方法，以上面代码为例 name() 和 partner() 。注意区别于我们平常getter的写法。
  - 一个 equals 方法的实现，执行比较时会比较该类的所有成员属性
  - 重写 equals 当然要重写 hashCode
  - 一个可以打印该类所有成员属性的 toString 方法。
  - 请注意只会有一个构造方法。

- 和枚举类型一样，记录也是类的一种受限形式。作为回报，记录对象在简洁性方面提供了显著的好处。
  - Enum　interface

- 还可以在Record声明的类中定义静态字段、静态方法、构造器或实例方法。
- 不能在Record声明的类中定义实例字段；类不能声明为abstract；不能声明显式的父类等。
- 为了在Java 14中引入这种新类型，需要在Java.lang.Class对象中添加如下两个新方法：
  - RecordComponent[] getRecordComponents()
  - boolean isRecord()

- 这是JDK 12和JDK 13中的预览特性，现在是正式特性了。
- 该特性规定，switch可以当作语句使用，也可以当作表达式使用。
- 这可以简化日常的编码方式，也为本版本中预览的模式匹配（JEP 305）特性打下了基础。
- 具体情况：

  使用->来替代以前的:+break；另外还提供了yield来在block中返回值

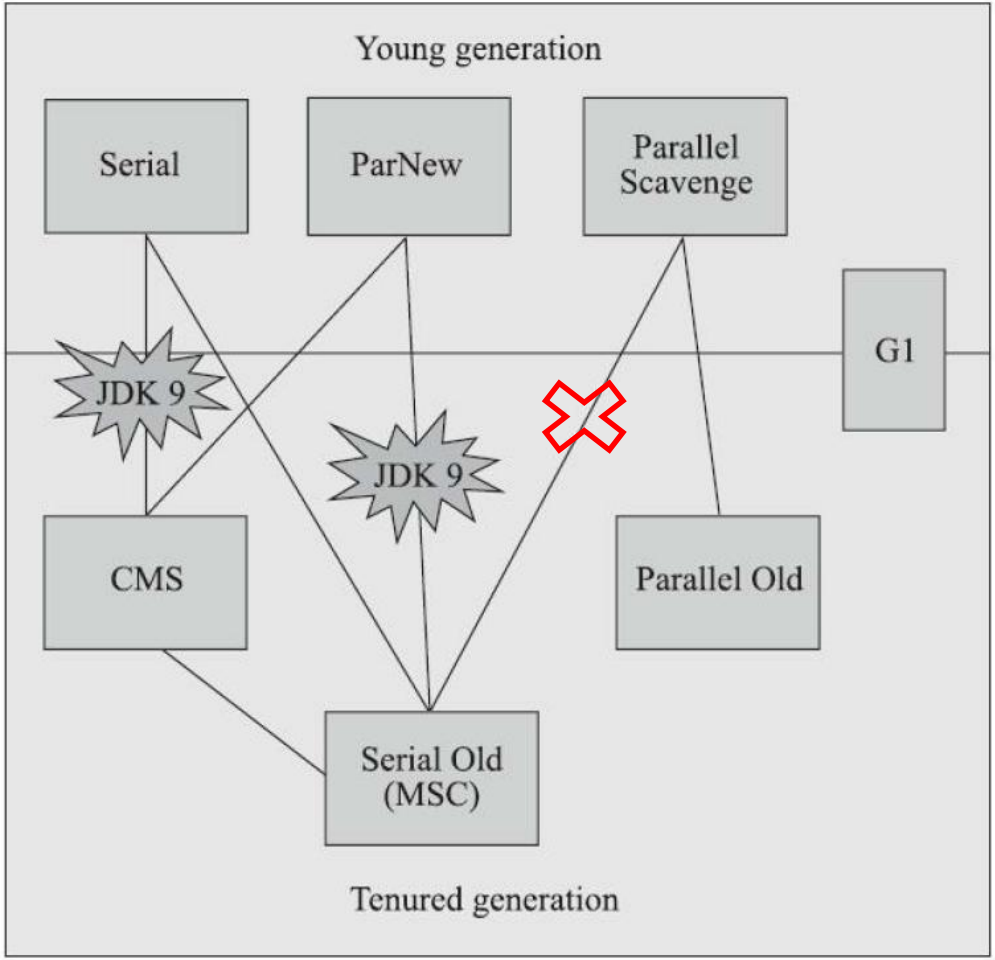
- JDK13引入的text blocks进行第二轮preview，JDK14的版本主要增加了两个escape sequences，分别是 \ <line-terminator>与\s escape sequence

- 现实问题

在Java中，通常需要使用String类型表达HTML，XML，SQL或JSON等格式的字符串，在进行字符串赋值时需要进行转义和连接操作，然后才能编译该代码，这种表达方式难以阅读并且难以维护。

- 目标
  - 简化跨越多行的字符串，避免对换行等特殊字符进行转义，简化编写Java程序。
  - 增强Java程序中用字符串表示的其他语言的代码的可读性
  - 解析新的转义序列

6.JEP 366：弃用ParallelScavenge和SerialOld GC组合



- 由于维护和兼容性测试的成本，在JDK 8时将Serial+CMS、ParNew+Serial Old这两个组合声明为废弃（JEP 173），并在JDK 9中完全取消了这些组合的支持（JEP214）

- ParallelScavenge + SerialOld GC的GC组合要被标记为Deprecate了。

6.JEP 366：弃用ParallelScavenge和SerialOld GC组合

- JDK官方给出将这个GC组合标记为Deprecate的理由是：这个GC组合需要大量的代码维护工作，并且，这个GC组合很少被使用。因为它的使用场景应该是一个很大的Young区配合一个很小的Old区，这样的话，Old区用SerialOldGC去收集时停顿时间我们才能勉强接受。

- 废弃了parallel young generation GC与SerialOld GC的组合(-XX:+UseParallelGC与-XX:-UseParallelOldGC配合开启)，现在使用-XX:+UseParallelGC -XX:-UseParallelOldGC或者-XX:-UseParallelOldGC都会出现告警如下：

```
Java HotSpot(TM) 64-Bit Server VM warning: Option
UseParallelOldGC was deprecated in version 14.0 and will likely
be removed in a future release.
```

7. JEP 363：删除CMS垃圾回收器

- 该来的总会来，自从G1（基于Region分代）横空出世后，CMS在JDK9中就被标记为Deprecate了（JEP 291: Deprecate the Concurrent Mark Sweep (CMS) Garbage Collector）

- CMS的弊端 ：
  1. 会产生内存碎片，导致并发清除后，用户线程可用的空间不足。
  2. 既然强调了并发（Concurrent），CMS收集器对CPU资源非常敏感
  3. CMS 收集器无法处理浮动垃圾

- 上述的这些问题，尤其是碎片化问题，给你的JVM实例就像埋了一颗炸弹。说不定哪次就在你的业务高峰期来一次FGC。当CMS停止工作时，会把Serial Old GC 作为备选方案，而Serial Old GC 是JVM中性能最差的垃圾回收方式，停顿个几秒钟，上十秒都有可能。

- 移除了CMS垃圾收集器，如果在JDK14中使用-XX:+UseConcMarkSweepGC的话，JVM不会报错，只是给出一个warning信息。

- 现在G1回收器已成为默认回收器好几年了。
- 我们还看到了引入了两个新的收集器：ZGC（JDK11出现）和Shenandoah（open jdk12）。
  - 主打特点：低停顿时间

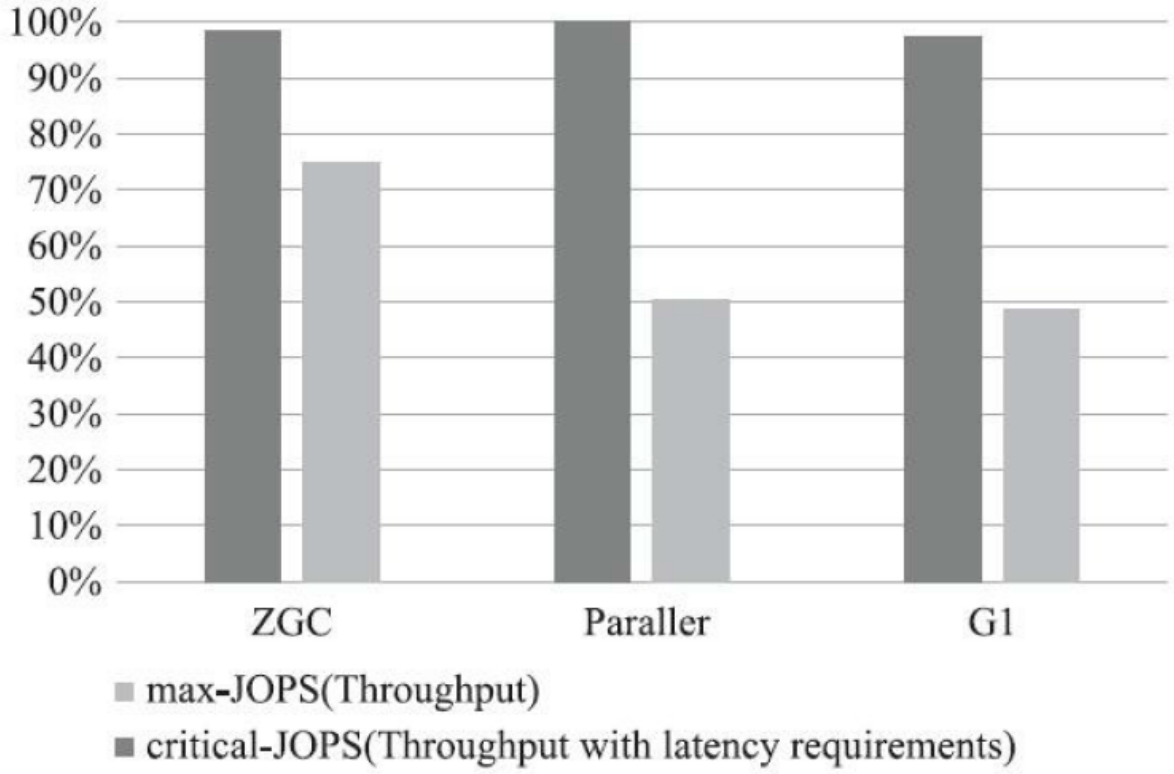
Shenandoah开发团队在实际应用中的测试数据

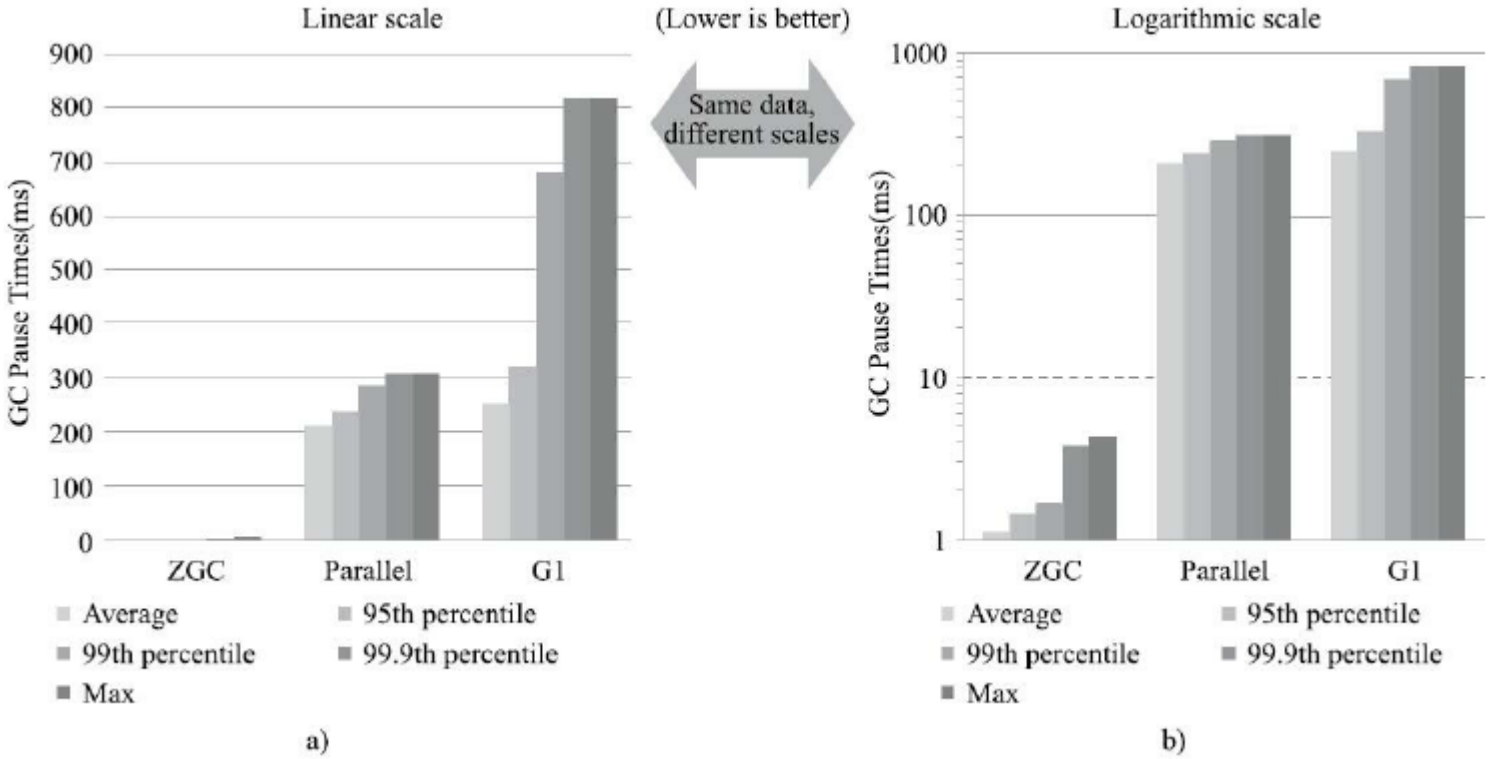
收 集 器	运行时间	总 停 顿	最大停顿	平均停顿
Shenandoah	387.602s	320ms	89.79ms	53.01ms
G1	312.052s	11.7s	1.24s	450.12ms
CMS	285.264s	12.78s	4.39s	852.26ms
Parallel Scavenge	260.092s	6.59s	3.04s	823.75ms

令人震惊、革命性的ZGC

- ZGC与Shenandoah目标高度相似，在尽可能对吞吐量影响不大的前提下，实现在任意堆内存大小下都可以把垃圾收集的停顿时间限制在十毫秒以内的低延迟。

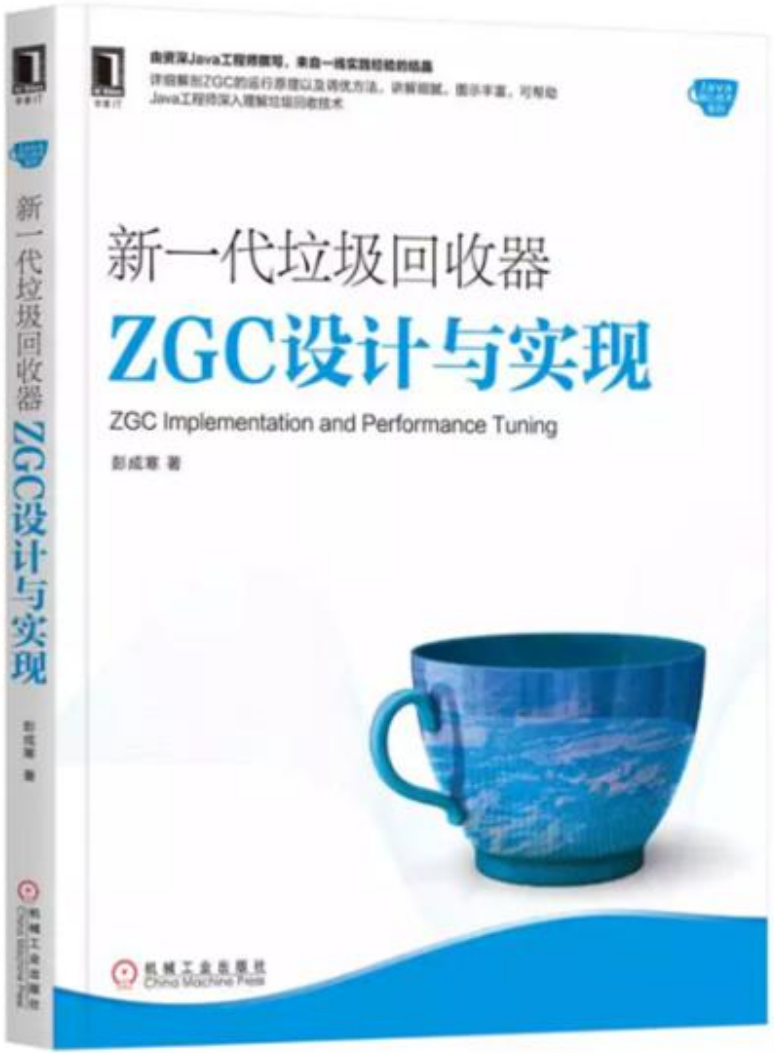
- 《深入理解Java虚拟机》一书中这样定义ZGC：ZGC收集器是一款基于Region内存布局的，（暂时）不设分代的，使用了读屏障、染色指针和内存多重映射等技术来实现可并发的标记-压缩算法的，以低延迟为首要目标的一款垃圾收集器。


100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
ZGC
Paraller
G1
max-JOPS(Throughput)
critical-JOPS(Throughput with latency requirements)


Linear scale
(Lower is better)
Logarithmic scale
GC Pause Times(ms)
900
800
700
600
500
400
300
200
100
0
Same data, different scales
1000
100
10
1
ZGC
Parallel
G1
Average
95th percentile
99th percentile
99.9th percentile
Max
a)
b)

虽然ZGC还在试验状态，没有完成所有特性，但此时性能已经相当亮眼，用“令人震惊、革命性”来形容，不为过。

未来将在服务端、大内存、低延迟应用的首选垃圾收集器。



JEP 364：ZGC应用在macOS上

JEP 365：ZGC应用在Windows上

- JDK14之前，ZGC仅Linux才支持。
- 尽管许多使用ZGC的用户都使用类Linux的环境，但在Windows和macOS上，人们也需要ZGC进行开发部署和测试。许多桌面应用也可以从ZGC中受益。因此，ZGC特性被移植到了Windows和macOS上。
- 现在mac或Windows上也能使用ZGC了，示例如下：
  -XX:+UnlockExperimentalVMOptions -XX:+UseZGC

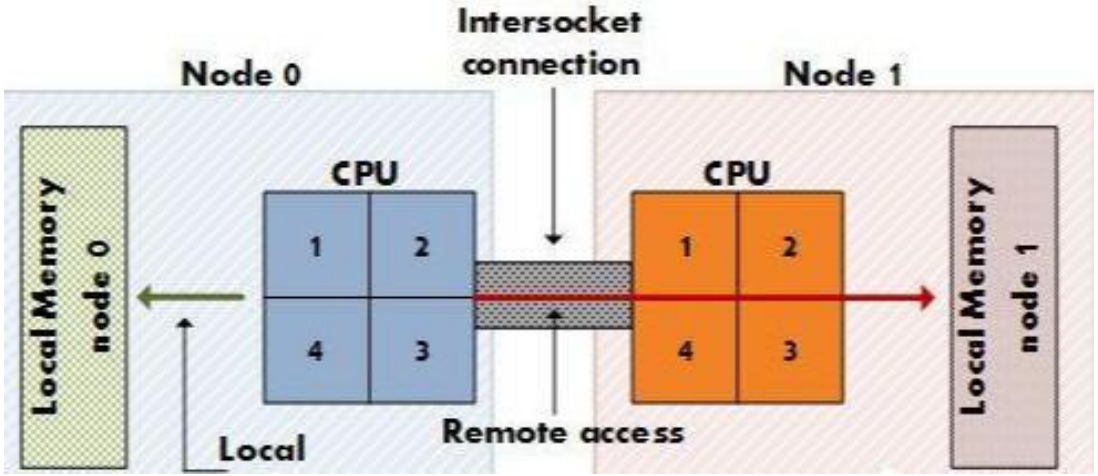
4- 其它新特性

- 这个孵化器工具为开发者带来了一种打包Java应用的方式，目的在于创建一个简单的打包工具，可以用于构建exe、pkg、dmg、deb、rpm格式的安装文件。

- JDK14引入了jdk.incubator.jpackage.jmod，它基于JavaFX javapackager tool构建。

该功能改进了G1垃圾回收器在非一致内存访问（NUMA）系统上的整体性能。

NUMA就是非统一内存访问架构（英语：non-uniform memory access，简称NUMA），是一种为多处理器的电脑设计的内存架构，内存访问时间取决于内存相对于处理器的位置。



Java为了更方便的了解运行的JVM情况，在之前的JDK11版本中引入了JFR特性，即JDK Flight Recorder。但是使用不太灵活。虽然JVM通过JFR暴露了超过500项数据，但是其中大部分数据只能通过解析JFR日志文件才能获取得到，而不是实时获取。用户想要使用JFR的数据的话，用户必须先开启JFR进行记录，然后停止记录，再将飞行记录的数据Dump到磁盘上，然后分析这个记录文件。

举例：

jcmd <PID> JFR.start name=test duration=60s settings=template.jfc filename=output.jfr

新特性中，可以公开JDK Flight Recorder（JFR）的数据，用于持续监视，从而简化各种工具和应用程序对JFR数据的访问。

在JEP 352中，对FileChannel API进行了扩展，以允许创建MappedByteBuffer实例。

与易失性存储器（RAM）不同，它们在非易失性数据存储（NVM，非易失性存储器）上工作。但是，目标平台是Linux x64。

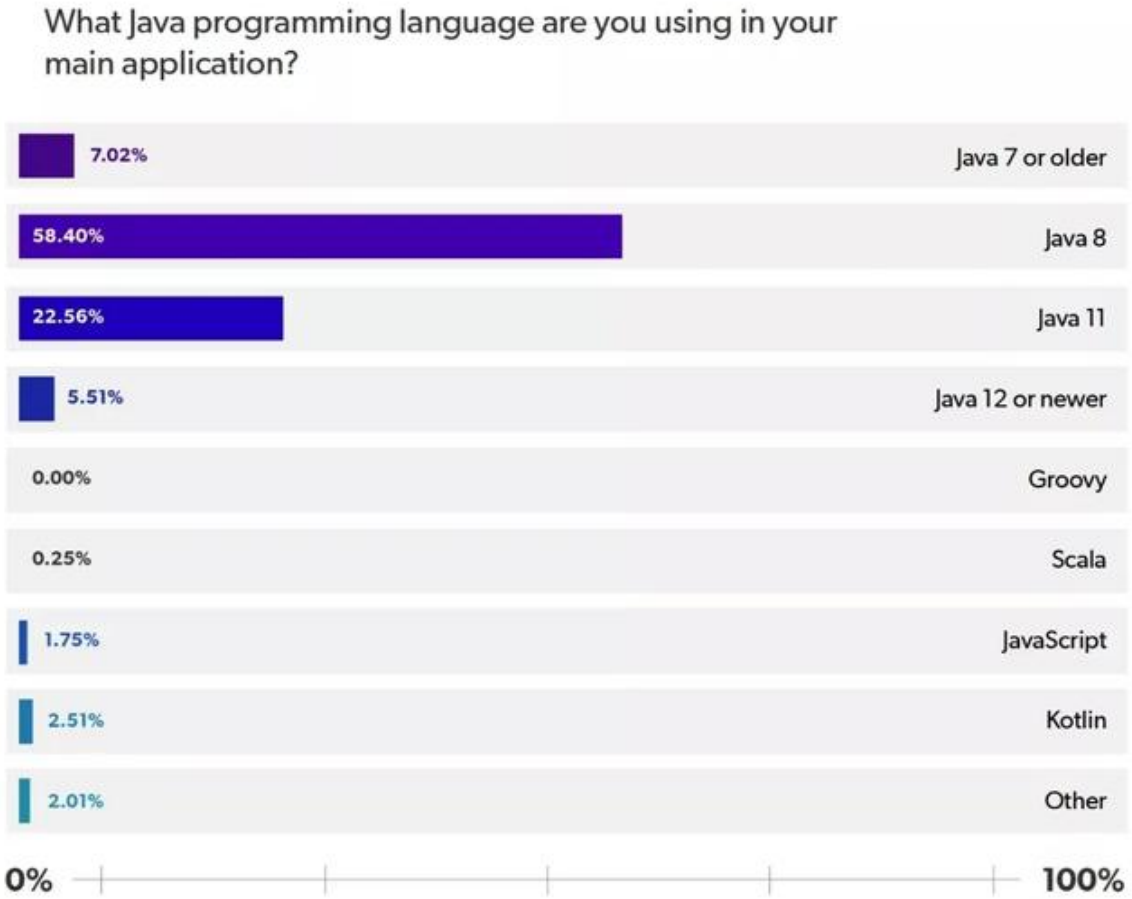
非易失性内存能够持久保持数据，因此可以利用该特性来改进性能。

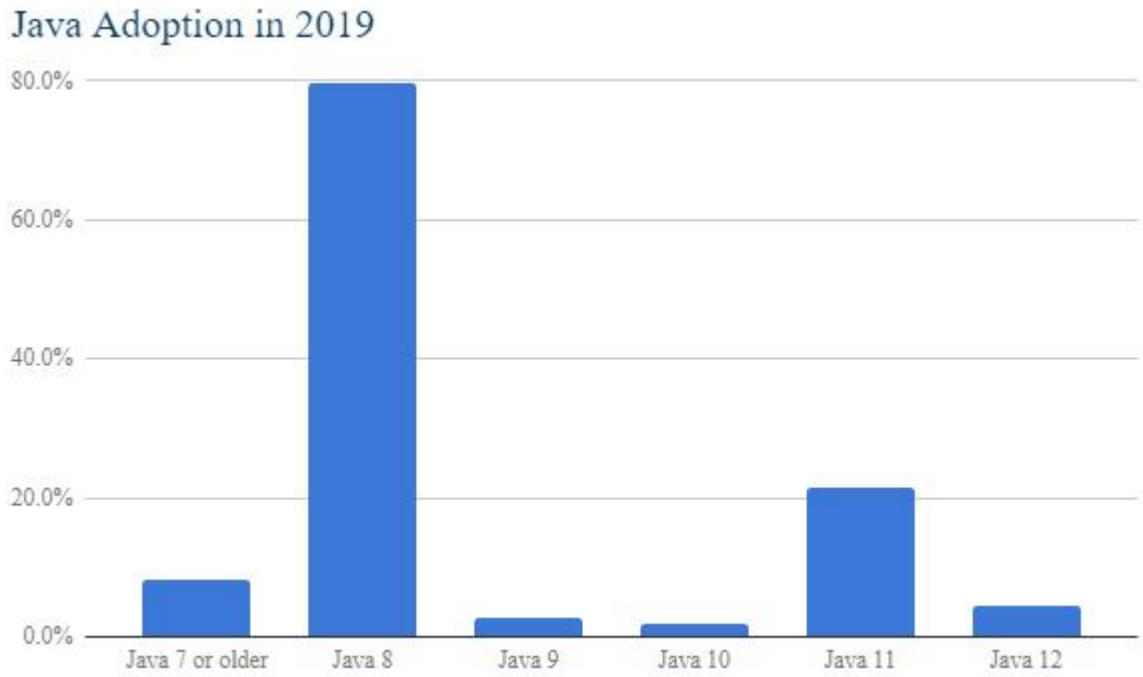
14. JEP 370：外部内存访问API
15. JEP 362：弃用Solaris和SPARC的移植
16. JEP 367：删除Pack200工具和API

上面列出的是大方面的特性，除此之外还有一些api的更新及废弃，主要见JDK 14 Release Notes。这里不再赘述。

5- 写在最后

JDK 14 性能提升，但 JDK 8 仍是最强王者！





在两个长期支持的版本 JDK 8 和 JDK 11中，相比之下肯定是JDK 11 的新特性更多一些，但是并没有真的非升不可的新特性。


尚硅谷
让天下没有难学的的技术