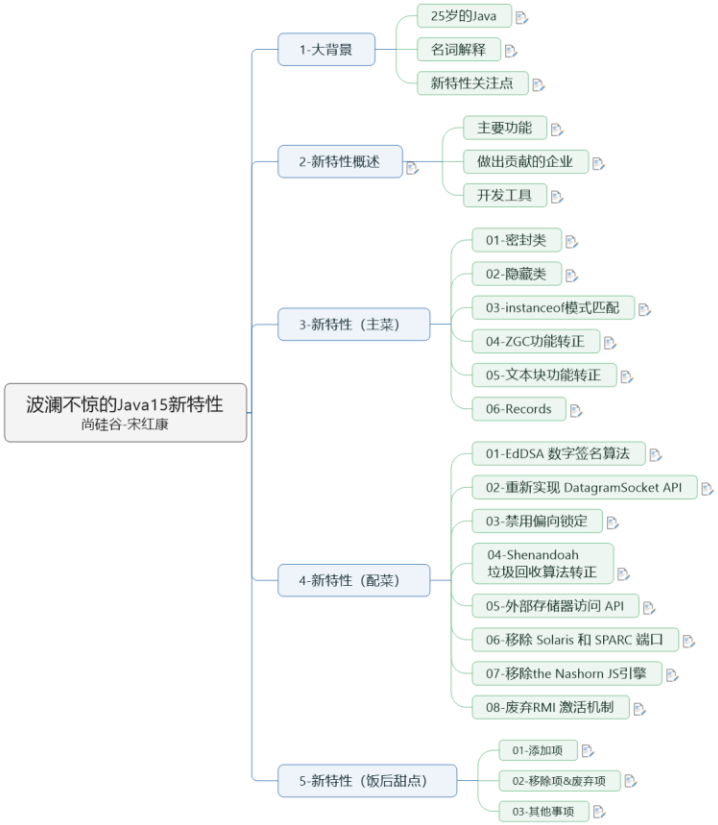
波澜不惊的 Java15 新特性 尚硅谷-宋红康



1. 1-大背景

1.1. 25 岁的 Java

1．Java 的现状

2020 年是 Java 诞生的第 25 个年头。Oracle 表示，目前 java 仍然流行编程语言，广泛使用在企业中。目前全世界有超过 69%的专职开发人员使用 Java，全世界有 510 亿台活动 Java 虚拟机（JVM）在部署中，目前 Java 主要被用于分析，数据管理，大数据，DevOps，移动，持续开发工具和聊天机器人等服务。

2．历史版本的主要新特性

在过去的这些年中，Java 在过去增强功能的推动下为用户提供了超过二十年的创新。例如：

JDK 5:enum、泛型、自动装箱与拆箱、可变参数、增强循环等

JDK 6:支持脚本语言、JDBC4.0API

JDK 7:支持 try-with-resources、switch 语句块增加 String 支持、NIO2.0 包

JDK 8:lambda 表达式、Stream API、新的日期时间的 API、方法引用、构造器引用

JDK 9:模块化系统、jshell

JDK 10:局部变量的类型推断

JDK 11:ZGC 的引入、Epsilon GC

JDK 12:switch 表达式、Shenandoah GC、增强 G1

JDK 13:switch 表达式引入 yield、文本块
JDK 14:instanceof 模式识别、Records、弃用 Parallel Scavenge+Serial GC 组合、删除 CMS GC

3. 近期版本发布及维护时间

OpenJDK 发版路线图：

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			

Oracle JDK 支持路线图：

Oracle Java SE支持路线图				
版本	GA日期	首要支持，直到	扩展支持，直到	持续支持
7	2011年7月	2019年7月	2022年7月	不定
8	2014年3月	2022年3月	2030年12月	不定
9（非LTS）	2017年9月	2018年3月	无法使用	不定
10（非LTS）	2018年3月	2018年9月	无法使用	不定
11（LTS）	2018年9月	2023年9月	2026年9月	不定
12（非LTS）	2019年3月	2019年9月	无法使用	不定
13（非LTS）	2019年9月	2020年3月	无法使用	不定
14（非LTS）	2020年3月	2020年9月	无法使用	不定
15（非LTS）	2020年9月	2021年3月	无法使用	不定

1.2. 名词解释

因为是小步快跑，快速迭代，因此此处解释下这两个词：孵化器模块（Incubator）和预览特性（Preview）。

孵化器模块（Incubator/孵化版/实验版）
尚未定稿的 API/工具，主要用于从 Java 社区收集使用反馈，稳定性无保障，后期有较大可能性移除

预览特性（Preview/预览版）
规格已成型，实现已确定，但还未最终定稿。这些特性还是存在被移除的可能性，但一般来说最后都会被固定下来。

1.3. 新特性关注点

角度一：语法层面
lambda 表达式，switch，自动装箱和拆箱、enum、接口中的静态方法、默认方法、私有方法

角度二：API 层面
Stream API、新的日期时间的 API、Optional、String、集合框架

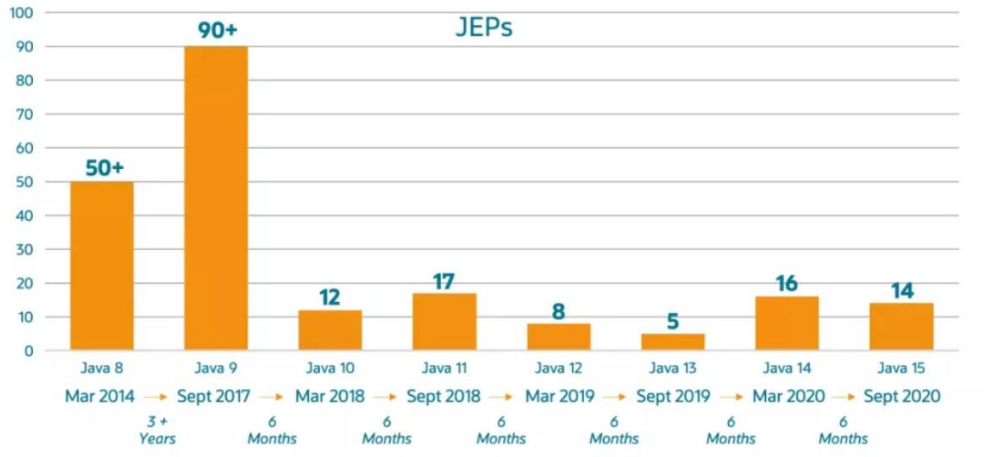
角度三：底层优化 Java C/C++
JVM 的优化、元空间、GC、GC 的组合、GC 的参数、js 的执行引擎、集合底层的实现

2. 2-新特性概述

1．JDK15 的发布
2020-9-15 当天，Oracle 宣布 Java/JDK15 正式发布。这是 Java 迭代周期缩短为六个月以后的第六个版本。随着迭代

速度的加快，让 Java 的设计、开发、测试人员能够更加轻松的面对迭代中的变化。

这些年发布的版本对应的 JEPs 数量如下图所示：



发布版本说明

根据发布的规划，这次发布的 JDK 15 将是一个短期的过度版，只会被 Oracle 支持（维护）6 个月，直到明年 3 月的 JDK 16 发布此版本将停止维护。而 Oracle 下一个长期支持版（LTS 版）会在明年的 9 月份后发布（Java 17），LTS 版每 3 年发布一个，上一次长期支持版是 18 年 9 月发布的 JDK 11。 而下一个长期支持 （LTS） 版本将是 JDK 17，该版本预计于 2021 年 9 月发布。

2.1. 主要功能

这次发布的主要功能有：
Java 15 为用户提供了 14 项主要的增强/更改，包括一个孵化器模块，三个预览功能，两个不推荐使用的功能以及两个删除功能。
http://openjdk.java.net/projects/jdk/15/

Features

339: Edwards-Curve Digital Signature Algorithm (EdDSA)
360: Sealed Classes (Preview)
371: Hidden Classes
372: Remove the Nashorn JavaScript Engine
373: Reimplement the Legacy DatagramSocket API
374: Disable and Deprecate Biased Locking
375: Pattern Matching for instanceof (Second Preview)
377: ZGC: A Scalable Low-Latency Garbage Collector
378: Text Blocks
379: Shenandoah: A Low-Pause-Time Garbage Collector
381: Remove the Solaris and SPARC Ports
383: Foreign-Memory Access API (Second Incubator)
384: Records (Second Preview)
385: Deprecate RMI Activation for Removal

对应中文特性：(JEP：JDK Enhancement Proposals，JDK 增强建议，也就是 JDK 的特性新增和改进提案。)

- JEP 339：EdDSA 数字签名算法
- JEP 360：密封类（预览）
- JEP 371：隐藏类
- JEP 372：移除 Nashorn JavaScript 引擎
- JEP 373：重新实现 Legacy DatagramSocket API
- JEP 374：禁用偏向锁定
- JEP 375：instanceof 模式匹配（第二次预览）
- JEP 377：ZGC：一个可扩展的低延迟垃圾收集器
- JEP 378：文本块
- JEP 379：Shenandoah:低暂停时间垃圾收集器
- JEP 381：移除 Solaris 和 SPARC 端口

- JEP 383：外部存储器访问 API（第二次孵化版）
- JEP 384：Records（第二次预览）
- JEP 385：废弃 RMI 激活机制

总结：

- JDK15 整体来看新特性方面并不算很亮眼，它主要是对之前版本**预览特性**的功能做了确定，如文本块、ZGC 等，这么一来我们就可以放心大胆的使用了。
- **你发任你发，我用 Java8。**

2.2. 做出贡献的企业

通过 Oracle 的官方博文看到，虽然主力军 Oracle 干了 79.68%的工作量，其次是开源巨头红帽。但还是有不少企业参与其中。国内企业贡献最多的居然是腾讯公司，被 Oracle 点名感谢。阿里巴巴和华为也做出了不可磨灭的贡献。Java 有中国科技力量的贡献能大大提升中国 IT 的硬实力。

2.3. 开发工具

支持 Java15 的开发工具

目前，官方博文说支持 Java15 的工具，就下面这三款：

- JetBrains IDEA
- Apache NetBeans
- Eclipse Marketplace

运行 JDK15 需要 IDEA 2020.2 才能支持。（JDK14 要求 IDEA 2020.1）

JDK15 下载路径：
https://www.oracle.com/java/technologies/javase-jdk15-downloads.html

IDEA 2020.02 版本下载地址：

https://www.jetbrains.com/idea/download/#section=windows

3. 3-新特性（主菜）

3.1. 01-密封类

JEP 360:Sealed Classes(Preview)密封的类和接口（预览）
通过密封的类和接口来增强 Java 编程语言，这是新的预览特性。
用于限制超类的使用，密封的类和接口限制其它可能继承或实现它们的其它类或接口。

这个特性的目标包括—允许类或接口的开发者来控制哪些代码负责实现，提供了比限制使用超类的访问修饰符声明方式更多选择,并通过支持对模式的详尽分析而支持模式匹配的未来发展。

在 Java 中，类层次结构通过继承实现代码的重用，父类的方法可以被许多子类继承。
但是，类层次结构的目的并不总是重用代码。有时，其目的是对域中存在的各种可能性进行建模，例如图形库支持的形状类型或金融应用程序支持的贷款类型。当以这种方式使用类层次结构时，我们可能需要限制子类集从而来简化建模。

具体使用：
因为我们引入了 sealed class 或 interfaces，这些 class 或者 interfaces 只允许被指定的类或者 interface 进行扩展和实现。

使用修饰符 sealed，您可以将一个类声明为密封类。密封的类使用 reserved 关键字 permits 列出可以直接扩展它的类。子类可以是最终的，非密封的或密封的。

示例：

```
package com.atguigu.java;

public abstract sealed class Shape permits
Circle, Rectangle, Square {...}
public final class Circle extends Shape {...}
public sealed class Rectangle extends Shape
permits TransparentRectangle, FilledRectangle
{...}
public final class TransparentRectangle
extends Rectangle {...}
public final class FilledRectangle extends
Rectangle {...}
public non-sealed class Square extends Shape
{...}
```

3.2. 02-隐藏类

JEP 371:Hidden Classes(隐藏类)

该提案通过启用标准 API 来定义无法发现且具有有限生命周期的隐藏类，从而提高 JVM 上所有语言的效率。JDK 内部和外部的框架将能够动态生成类，而这些类可以定义隐藏类。通常来说基于 JVM 的很多语言都有动态生成类的机制，这样可以提高语言的灵活性和效率。

- 隐藏类天生为框架设计的，在运行时生成内部的 class。
- 隐藏类只能通过反射访问，不能直接被其他类的字节码访问。
- 隐藏类可以独立于其他类加载、卸载，这可以减少框架的内存占用。

Hidden Classes 是什么呢？

Hidden Classes 就是不能直接被其他 class 的二进制代码使用的 class。Hidden Classes 主要被一些框架用来生成运行时类，但是这些类不是被用来直接使用的，而是通过反射机制来调用。
比如在 JDK8 中引入的 lambda 表达式，JVM 并不会在编译的时候将 lambda 表达式转换成为专门的类，而是在运行时将相应的字节码动态生成相应的类对象。
另外使用动态代理也可以为某些类生成新的动态类。

那么我们希望这些动态生成的类需要具有什么特性呢？

1. **不可发现性。**因为我们是为某些静态的类动态生成的动态类，所以我们希望把这个动态生成的类看做是静态类的一部分。所以我们不希望除了该静态类之外的其他机制发现。
2. **访问控制。**我们希望在访问控制静态类的同时，也能控制到动态生成的类。

3. **生命周期。**动态生成类的生命周期一般都比较短，我们并不需要将其保存和静态类的生命周期一致。

API 的支持

所以我们需要一些 API 来定义无法发现的且具有有限生命周期的隐藏类。这将提高所有基于 JVM 的语言实现的效率。

比如：

java.lang.reflect.Proxy 可以定义隐藏类作为实现代理接口的代理类。

java.lang.invoke.StringConcatFactory 可以生成隐藏类来保存常量连接方法；

java.lang.invoke.LambdaMetaFactory 可以生成隐藏的 nestmate 类，以容纳访问封闭变量的 lambda 主体；

普通类是通过调用 ClassLoader::defineClass 创建的，而隐藏类是通过调用 Lookup::defineHiddenClass 创建的。这使 JVM 从提供的字节中派生一个隐藏类，链接该隐藏类，并返回提供对隐藏类的反射访问的查找对象。调用程序可以通过返回的查找对象来获取隐藏类的 Class 对象。

3.3. 03-instanceof 模式匹配

JEP 375：Pattern Matching for instanceof (Second Preview) instanceof 自动匹配模式

在 Java 14 中作为预览语言功能引入的 instanceof 模式匹配，在 Java 15 中处于第二次预览，而没有任何更改。

模式匹配允许程序中的通用逻辑（主要是从对象中的条件提取组件）可以更简洁地表达。Haskell 和 C# 等语言已采用模式匹配来实现简洁和安全性。

旧写法：

```
// 先判断类型
```

```
if (obj instanceof String) {
    // 然后转换
    String s = (String) obj;
    // 然后才能使用
}
```

新写法：（自动匹配模式）

```
if (obj instanceof String s) {
    // 如果类型匹配 直接使用
} else {
    // 如果类型不匹配则不能直接使用
}
```

3.4. 04-ZGC 功能转正

JEP 377:ZGC: A Scalable Low-Latency Garbage Collector (Production) ZGC 功能转正

ZGC 是 Java 11 引入的新的垃圾收集器（JDK9 以后默认的垃圾回收器是 G1），经过了多个实验阶段，自此终于成为正式特性。
自 2018 年以来，ZGC 已增加了许多改进，从并发类卸载、取消使用未使用的内存、对类数据共享的支持到改进的 NUMA 感知。此外，最大堆大小从 4 TB 增加到 16 TB。支持的平台包括 Linux、Windows 和 MacOS。

ZGC 是一个重新设计的并发的垃圾回收器，通过减少 GC 停顿时间来提高性能。

但是这并不是替换默认的 GC，默认的 GC 仍然还是 G1；之前需要通过-XX:+UnlockExperimentalVMOptions -XX:+UseZGC 来启用 ZGC，现在只需要-XX:+UseZGC 就可以。相信不久的将来它必将成为默认的垃圾回收器。

相关的参数有 ZAllocationSpikeTolerance、ZCollectionInterval、ZFragmentationLimit、ZMarkStackSpaceLimit、ZProactive、ZUncommit、ZUncommitDelay ZGC-specific JFR events(ZAllocationStall、ZPageAllocation、ZPageCacheFlush、ZRelocationSet、ZRelocationSetGroup、ZUncommit)也从 experimental 变为 product

3.5. 05-文本块功能转正

JEP 378：文本块功能转正

Text Blocks 首次是在 JDK 13 中以预览功能出现的，然后在 JDK 14 中又预览了一次，终于在 JDK 15 中被确定下来，可放心使用了。

文本块是一种多行字符串文字，它避免了大多数转义序列的需要，以一种可预测的方式自动设置字符串的格式，并在需要时使开发人员可以控制格式，简化编写 Java 程序的任务。

文本块建议的目标是提高 Java 程序中的字符串的可读性，这些字符串表示以非 Java 语言编写的代码。
另一个目标是支持从字符串文本迁移，规定任何新构造都可以表达与字符串文本相同的字符串集，解释相同的转义序列，并且以与字符串文本相同的方式进行操作。
OpenJDK 开发人员希望添加转义序列来管理显式空白和换行控件。

3.6. 06-Records

JEP 384：Records Class（预览）

Records Class 也是第二次出现的预览功能，它在 JDK 14 中也出现过一次了，使用 Record 可以更方便的创建一个常量类，使用的前后代码对比如下。

- 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。

旧写法：

```
class Point {
    private final int x;
    private final int y;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
    int x() { return x; }
    int y() { return y; }
    public boolean equals(Object o) {
        if (!(o instanceof Point)) return
false;
        Point other = (Point) o;        return
other.x == x && other.y = y;
    }    public int hashCode() {
        return Objects.hash(x, y);
    }    public String toString() {
        return String.format("Point[x=%d,
y=%d]", x, y);
    }
}
```

新写法：

```
record Point(int x, int y) { }
```

4. 4-新特性（配菜）

4.1. 01-EdDSA 数字签名算法

JEP 339：Edwards-Curve Digital Signature Algorithm（EdDSA 数字签名算法)

这是一个新的功能。
新加入基于 Edwards-Curve 数字签名算法(EdDSA-Edwards-Curve Digital Signature Algorithm)的加密签名，即爱德华兹曲线数字签名算法。

在许多其它加密库（如 OpenSSL 和 BoringSSL）中得到支持。

与 JDK 中的现有签名方案相比，EdDSA 具有更高的安全性和性能，因此备受关注。它已经在 OpenSSL 和 BoringSSL 等加密库中得到支持，在区块链领域用的比较多。

EdDSA 是一种现代的椭圆曲线方案，具有 JDK 中现有签名方案的优点。EdDSA 将只在 SunEC 提供商中实现。

4.2. 02-重新实现 DatagramSocket API

JEP 373:Reimplement the Legacy DatagramSocket API(重新实现 DatagramSocket API)

新的计划是 JEP 353 的后续，该方案重新实现了遗留的套接字 API。
java.net.datagram.Socket 和 java.net.MulticastSocket 的当前实现可以追溯到 JDK 1.0，那时 IPv6 还在开发中。因此，当前的多播套接字实现尝试调和 IPv4 和 IPv6 难以维护的方式。

- 通过替换 java.net.datagram 的基础实现，重新实现旧版 DatagramSocket API。
- 更改 java.net.DatagramSocket 和 java.net.MulticastSocket 为更加简单、现代化的底层实现。提高了 JDK 的可维护性和稳定性。

通过将 java.net.datagram.Socket 和 java.net.MulticastSocket API 的底层实现替换为更简单、更现代的实现来重新实现遗留的 DatagramSocket API。
新的实现：1.易于调试和维护;2.与 Project Loom 中正在探索的虚拟线程协同。

4.3. 03-禁用偏向锁定

JEP 374: Disable and Deprecate Biased Locking
禁用偏向锁定

在默认情况下禁用偏向锁定，并弃用所有相关命令行选项。目标是确定是否需要继续支持偏置锁定的高维护成本的遗留同步优化，HotSpot 虚拟机使用该优化来减少非竞争锁定的开销。尽管某些 Java 应用程序在禁用偏向锁后可能会出现性能下降，但偏向锁的性能提高通常不像以前那么明显。

该特性默认禁用了 biased locking(-XX:+UseBiasedLocking)，并且废弃了所有相关的命令行选型(BiasedLockingStartupDelay, BiasedLockingBulkRebiasThreshold, BiasedLockingBulkRevokeThreshold, BiasedLockingDecayTime, UseOptoBiasInlining, PrintBiasedLockingStatistics and PrintPreciseBiasedLockingStatistics)

4.4. 04-Shenandoah 垃圾回收算法转正

Shenandoah 垃圾回收算法终于从实验特性转变为产品特性，这是一个从 JDK 12 引入的回收算法，该算法通过与正在运行的 Java 线程同时进行疏散工作来减少 GC 暂停时间。Shenandoah 的暂停时间与堆大小无关，无论堆栈是 200 MB 还是 200 GB，都具有相同的一致暂停时间。

怎么形容 Shenandoah 和 ZGC 的关系呢？异同点大概如下：

- 相同点：性能几乎可认为是相同的
- 不同点：ZGC 是 Oracle JDK 的，根正苗红。而 Shenandoah 只存在于 OpenJDK 中，因此使用时需注意你的 JDK 版本

打开方式：使用-XX:+UseShenandoahGC 命令行参数打开。

Shenandoah 在 JDK12 被作为 experimental 引入，在 JDK15 变为 Production；之前需要通过-XX:+UnlockExperimentalVMOptions -XX:+UseShenandoahGC 来启用，现在只需要-XX:+UseShenandoahGC 即可启用

4.5. 05-外部存储器访问 API

JEP 383:Foreign-Memory Access API (Second Incubator) 外部存储器访问 API（孵化器版）
目的是引入一个 API，以允许 Java 程序安全、有效地访问 Java 堆之外的外部存储器。如本机、持久和托管堆。

有许多 Java 程序是访问外部内存的，比如 Ignite 和 MapDB。该 API 将有助于避免与垃圾收集相关的成本以及与跨进程共享内存以及通过将文件映射到内存来序列化和

反序列化内存内容相关的不可预测性。该 Java API 目前没有为访问外部内存提供令人满意的解决方案。但是在新的提议中，API 不应该破坏 JVM 的安全性。

Foreign-Memory Access API 在 JDK14 被作为 incubating API 引入，在 JDK15 处于 Second Incubator，提供了改进。

4.6. 06-移除 Solaris 和 SPARC 端口

JEP 381：Remove the Solaris and SPARC Ports (移除 Solaris 和 SPARC 端口)

删除对 Solaris/SPARC、Solaris/x64 和 Linux/SPARC 端口的源代码和构建支持，在 JDK 14 中被标记为废弃，在 JDK15 版本正式移除。

许多正在开发的项目和功能（如 Valhalla、Loom 和 Panama）需要进行重大更改以适应 CPU 架构和操作系统特定代码。

近年来，Solaris 和 SPARC 都已被 Linux 操作系统和英特尔处理器取代。放弃对 Solaris 和 SPARC 端口的支持将使 OpenJDK 社区的贡献者能够加速开发新功能，从而推动平台向前发展。

4.7. 07-移除 the Nashorn JS 引擎

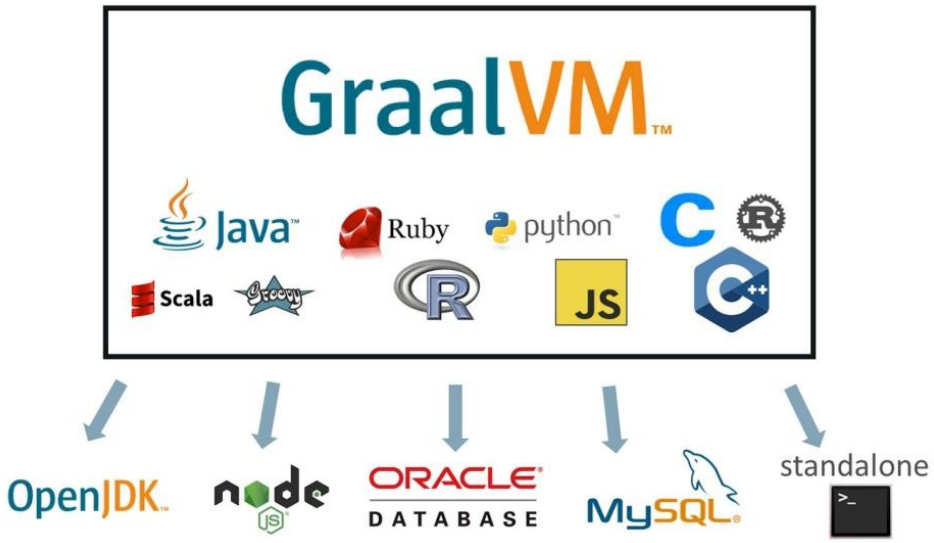
JEP 372：Remove the Nashorn JavaScript Engine

Nashorn 是在 JDK 提出的脚本执行引擎，该功能是 2014 年 3 月发布的 JDK 8 的新特性。在 JDK11 就已经把它标记为废弃了，JDK15 完全移除。

在 JDK11 中取以代之的是 GraalVM。GraalVM 是一个运行时平台，它支持 Java 和其他基于 Java 字节码的语言，但也支持其他语言，如 JavaScript，Ruby，Python 或 LLVM。性能是 Nashorn 的 2 倍以上。

JDK15 移除了 Nashorn JavaScript Engine 及 jjs 命令行工具。具体就是 jdk.scripting.nashorn 及 jdk.scripting.nashorn.shell 这两个模块被移除了。

补充：



Graal VM 在 HotSpot VM 基础上增强而成的跨语言全栈虚拟机，可以作为“任何语言”的运行平台使用。语言包括：Java、Scala、Groovy、Kotlin；C、C++、JavaScript、Ruby、Python、R 等

4.8. 08-废弃 RMI 激活机制

JEP 385:Deprecate RMI Activation for Removal

RMI Activation 被标记为 Deprecate,将会在未来的版本中删除。RMI 激活机制是 RMI 中一个过时的部分，自 Java 8 以来一直是可选的而非必选项。RMI 激活机制增加了持续的维护负担。RMI 的其他部分暂时不会被弃用。

RMI jdk1.2 引入，EJB

在 RMI 系统中，我们使用延迟激活。延迟激活将激活对象推迟到客户第一次使用（即第一次方法调用）之前。
既然 RMI Activation 这么好用，为什么要废弃呢？

因为对于现代应用程序来说，分布式系统大部分都是基于 Web 的，web 服务器已经解决了穿越防火墙，过滤请求，身份验证和安全性的问题，并且也提供了很多延迟加载的技术。
所以在现代应用程序中，RMI Activation 已经很少被使用到了。并且在各种开源的代码库中，也基本上找不到 RMI Activation 的使用代码了。
为了减少 RMI Activation 的维护成本，在 JDK8 中，RMI Activation 被置为可选的。现在在 JDK15 中，终于可以废弃了。

5. 5-新特性（饭后甜点）

5.1. 01-添加项

添加项

- Support for Unicode 13.0 (JDK-8239383)

升级了 Unicode，支持 Unicode 13.0

- Added isEmpty Default Method to CharSequence (JDK-8215401)

给 CharSequence 新增了 isEmpty 方法
java.base/java/lang/CharSequence.java

```
1 @since 15
2 default boolean isEmpty() {
3     return this.length() == 0;
4 }
```

- Specialized Implementations of TreeMap Methods (JDK-8176894)

JDK15 对 TreeMap 提供了 putIfAbsent, computeIfAbsent, computeIfPresent, compute, merge 方法提供了 overriding 实现

- New Option Added to jcmd for Writing a gzipped Heap Dump (JDK-8237354)

jcmd 的 GC.heap_dump 命令现在支持 gz 选型，以 dump 出 gzip 压缩版的 heap；compression level 从 1(fastest)到 9(slowest, but best compression)，默认为 1

- Added Support for SO_INCOMING_NAPI_ID Support (JDK-8243099)

jdk.net.ExtendedSocketOptions 新增了 SO_INCOMING_NAPI_ID 选型

- New System Properties to Configure the TLS Signature Schemes (JDK-8242141)

新增了 jdk.tls.client.SignatureSchemes 及 jdk.tls.server.SignatureSchemes 用于配置 TLS Signature Schemes

- Support for certificate_authorities Extension (JDK-8206925)

支持 certificate_authorities 的扩展

5.2. 02-移除项&废弃项

移除项

- Obsolete -XX:UseAdaptiveGCBoundary (JDK-8228991)

淘汰了-XX:UseAdaptiveGCBoundary

废弃项

- Deprecated -XX:ForceNUMA Option (JDK-8243628)

废弃了 ForceNUMA 选项

- Disable Native SunEC Implementation by Default (JDK-8237219)

默认禁用了 Native SunEC Implementation

5.3. 03-其他事项

已知问题

- java.net.HttpClient Does Not Override Protocols Specified in SSLContext Default Parameters (JDK-8239594)

HttpClient 现在没有覆盖在 SSLContext Default Parameters 中指定的 Protocols

其他事项

- DatagramPacket.getPort() Returns 0 When the Port Is Not Set (JDK-8237890)

当 DatagramPacket 没有设置 port 的时候，其 getPort 方法返回 0

- Improved Ergonomics for G1 Heap Region Size (JDK-8241670)

优化了默认 G1 Heap Region Size 的计算