

JAVA改变

[零、说明 4](#_Toc38561085)

[一、JDK8 4](#_Toc38561086)

[1、函数式接口 4](#_Toc38561087)

[2、Lambda 表达式 4](#_Toc38561088)

[3、接口的增强 4](#_Toc38561089)

[4、集合的流式操作 5](#_Toc38561090)

[5、注解的更新 5](#_Toc38561091)

[6、安全性 6](#_Toc38561092)

[7、IO/NIO 的改进 6](#_Toc38561093)

[8、全球化功能 6](#_Toc38561094)

[二、JDK9 6](#_Toc38561095)

[1、模块系统 6](#_Toc38561096)

[2、Jshell 7](#_Toc38561097)

[3、集合、Stream 和 Optional 7](#_Toc38561098)

[4、进程 API 7](#_Toc38561099)

[5、平台日志 API 和 服务 8](#_Toc38561100)

[6、反应式流 （Reactive Streams） 8](#_Toc38561101)

[7、变量句柄 8](#_Toc38561102)

[8、改进方法句柄（Method Handle） 8](#_Toc38561103)

[9、并发 8](#_Toc38561104)

[10、Nashorn 9](#_Toc38561105)

[11、I/O 流新特性 9](#_Toc38561106)

[12、改进应用安全性能 9](#_Toc38561107)

[13、JVM 方面 9](#_Toc38561108)

[14、其他 9](#_Toc38561109)

[三、JDK10 9](#_Toc38561110)

[1、局部变量类型推断 10](#_Toc38561111)

[2、整合 JDK 代码仓库 10](#_Toc38561112)

[3、统一的垃圾回收接口 10](#_Toc38561113)

[4、并行全垃圾回收器 G1 10](#_Toc38561114)

[5、应用程序类数据共享 10](#_Toc38561115)

[6、线程-局部管控 10](#_Toc38561116)

[7、移除 Native-Header 自动生成工具 11](#_Toc38561117)

[8、额外的 Unicode 语言标签扩展 11](#_Toc38561118)

[9备用存储装置上的堆分配 11](#_Toc38561119)

[10、基于 Java 的 实验性 JIT 编译器 11](#_Toc38561120)

[11、根证书认证 11](#_Toc38561121)

[12、基于时间的版本发布模式 11](#_Toc38561122)

[四、JDK11 12](#_Toc38561123)

[1、基于嵌套的访问控制 12](#_Toc38561124)

[2、标准 HTTP Client 升级 12](#_Toc38561125)

[3、低开销垃圾回收器：Epsilon 12](#_Toc38561126)

[4、简化启动单个源代码文件的方法 12](#_Toc38561127)

[5、用于 Lambda 参数的局部变量语法 12](#_Toc38561128)

[6、低开销的 Heap Profiling 13](#_Toc38561129)

[7、支持 TLS 1.3 协议 13](#_Toc38561130)

[8、可伸缩低延迟垃圾收集器：ZGC(实验阶段) 13](#_Toc38561131)

[9、飞行记录器 13](#_Toc38561132)

[10、动态类文件常量 13](#_Toc38561133)

[五、JDK12 14](#_Toc38561134)

[1、Shenandoah：一个低停顿垃圾收集器（实验阶段） 14](#_Toc38561135)

[2、增加一套微基准套件 14](#_Toc38561136)

[3、Switch 表达式扩展（预览功能） 14](#_Toc38561137)

[4、引入 JVM 常量 API 14](#_Toc38561138)

[5、改进 AArch64 实现 14](#_Toc38561139)

[6、使用默认类数据共享（CDS）存档 14](#_Toc38561140)

[7、改善 G1 垃圾收集器，使其能够中止混合集合 14](#_Toc38561141)

[8、增强 G1 垃圾收集器，使其能自动返回未用堆内存给操作系统 14](#_Toc38561142)

[六、JDK13 14](#_Toc38561143)

[1、动态应用程序类-数据共享 15](#_Toc38561144)

[2、增强 ZGC 15](#_Toc38561145)

[3、Socket API 重构 15](#_Toc38561146)

[4、Switch 表达式扩展（预览功能）：yield 15](#_Toc38561147)

[5、文本块（预览功能） 15](#_Toc38561148)

[七、JDK14 16](#_Toc38561149)

[1、ZGC支持Windows/macOS 16](#_Toc38561150)

[2、弃用ParallelScavenge + SerialOld GC组合 16](#_Toc38561151)

[3、instanceof 运算符引入模式匹配（预览） 16](#_Toc38561152)

[4、NullPointerExceptions增强 16](#_Toc38561153)

[5、switch表达式标准化 16](#_Toc38561154)

[6、文本块第二版预览 16](#_Toc38561155)

# 零、说明

1、这里只收集了主要的更新点

2、新增的功能的具体原理和使用方法这里不详细说明

3、8->14,重大变化：lambda、Jshell、模块系统、var保留字、默认采用G1、废弃CMS、HttpClint标准化、Epsilon收集器、ZGC收集器、Shenandoah收集器、switch表达式扩展、线程-局部管控、应用程序类数据共享、Socket API 重构、文本块

# 一、JDK8

https://www.ibm.com/developerworks/cn/java/j-lo-jdk8newfeature/

https://www.oracle.com/technetwork/java/javase/8-relnotes-2226341.html

## 1、函数式接口

通过在接口里面添加一个抽象方法，这些方法可以直接从接口中运行。如果一个接口定义个唯一一个抽象方法，那么这个接口就成为函数式接口。同时，引入了一个新的注解：@FunctionalInterface。可以把他它放在一个接口前，表示这个接口是一个函数式接口。这个注解是非必须的，只要接口只包含一个方法的接口，虚拟机会自动判断，不过最好在接口上使用注解 @FunctionalInterface 进行声明。在接口中添加了 @FunctionalInterface 的接口，只允许有一个抽象方法，否则编译器也会报错。

## 2、Lambda 表达式

a、函数式接口的重要属性是：我们能够使用 Lambda 实例化它们。

b、Lambda 表达式让你能够将函数作为方法参数，或者将代码作为数据对待。Lambda 表达式的应用则使代码变得更加紧凑，可读性增强；Lambda 表达式使并行操作大集合变得很方便，可以充分发挥多核 CPU 的优势，更易于为多核处理器编写代码；

c、Lambda 表达式由三个部分组成：

c.1第一部分为一个括号内用逗号分隔的形式参数，参数是函数式接口里面方法的参数；

c.2第二部分为一个箭头符号：->；

c.3第三部分为方法体，可以是表达式和代码块

d、要使用 Lambda 表达式，需要定义一个函数式接口，这样往往会让程序充斥着过量的仅为 Lambda 表达式服务的函数式接口。为了减少这样过量的函数式接口，Java 8 在 java.util.function 中增加了不少新的函数式通用接口：

d.1、Function<T, R>：将 T 作为输入，返回 R 作为输出，他还包含了和其他函数组合的默认方法。

d.2、Predicate<T> ：将 T 作为输入，返回一个布尔值作为输出，该接口包含多种默认方法来将 Predicate 组合成其他复杂的逻辑（与、或、非）。

d.3、Consumer<T> ：将 T 作为输入，不返回任何内容，表示在单个参数上的操作。

## 3、接口的增强

a、Java 8 对接口做了进一步的增强。在接口中可以添加使用 default 关键字修饰的非抽象方法。还可以在接口中定义静态方法。如今，接口看上去与抽象类的功能越来越类似了。

b、默认方法：Java 8 还允许我们给接口添加一个非抽象的方法实现，只需要使用 default 关键字即可，这个特征又叫做扩展方法。在实现该接口时，该默认扩展方法在子类上可以直接使用，它的使用方式类似于抽象类中非抽象成员方法。但扩展方法不能够重载 Object 中的方法。例如：toString、equals、 hashCode 不能在接口中被重载。

c、静态方法：在接口中，还允许定义静态的方法。接口中的静态方法可以直接用接口来调用。

## 4、集合的流式操作

a、引入了流式操作（Stream），通过该操作可以实现对集合（Collection）的并行处理和函数式操作。

b、根据操作返回的结果不同，流式操作分为中间操作和最终操作两种, 可以将多个操作依次串联起来：

b.1、最终操作返回一特定类型的结果，该操作必须是流的最后一个操作，一旦被调用，Stream 就到了一个终止状态，而且不能再使用了。常见的操作有：

b.1.1、forEach()：对每个元素做处理；

b.1.2、toArray()：把元素导出到数组；

b.1.3、findFirst()：返回第一个匹配的元素；

b.1.4、anyMatch()：是否有匹配的元素等。

b.2、中间操作返回流本身,该操作会保持 stream 处于中间状态，允许做进一步的操作。它返回的还是的 Stream，允许更多的链式操作。常见的中间操作有：

b.2.1、filter()：对元素进行过滤；

b.2.2、sorted()：对元素排序；

b.2.3、map()：元素的映射；

b.2.4、distinct()：去除重复元素；

b.2.5、subStream()：获取子 Stream 等。

c、根据流的并发性，流又可以分为串行和并行两种。

c.1、串行流上的操作是在一个线程中依次完成

c.2、并行流则是在多个线程上同时执行

c.3、并行与串行的流可以相互切换：通过 stream.sequential() 返回串行的流，通过 stream.parallel() 返回并行的流。

c.4、相比较串行的流，并行的流可以很大程度上提高程序的执行效率。

d、流式操作实现了集合的过滤、排序、映射等功能。

e、Stream 和 Collection 集合的区别：

e.1、Collection 是一种静态的内存数据结构，主要面向内存，存储在内存中

e.2、Stream 是有关计算的，主要是面向 CPU，通过 CPU 实现计算

## 5、注解的更新

a、Java 8 的类型注解扩展了注解使用的范围：局部变量、类与接口、方法的异常也能添加注解。新增的两个注释的程序元素类型 ElementType.TYPE\_USE 和 ElementType.TYPE\_PARAMETER 用来描述注解的新场合。

a.1、ElementType.TYPE\_PARAMETER 表示该注解能写在类型变量的声明语句中。

a.2、ElementType.TYPE\_USE 表示该注解能写在使用类型的任何语句中（例如声明语句、泛型和强制转换语句中的类型）。

b、对类型注解的支持，增强了通过静态分析工具发现错误的能力。原先只能在运行时发现的问题可以提前在编译的时候被排查出来。Java 8 本身虽然没有自带类型检测的框架，但可以通过使用 Checker Framework 这样的第三方工具，自动检查和确认软件的缺陷，提高生产效率。

c、引入了重复注解机制，这样相同的注解可以在同一地方声明多次。重复注解机制本身必须用 @Repeatable 注解。

## 6、安全性

a、支持更强的基于密码的加密算法。基于 AES 的加密算法，例如 PBEWithSHA256AndAES\_128 和 PBEWithSHA512AndAES\_256，已经被加入进来。

b、在客户端，TLS1.1 和 TLS1.2 被设为默认启动。并且可以通过新的系统属性包 jdk.tls.client.protocols 来对它进行配置。

c、Keystore 的增强，包含新的 Keystore 类型 java.security.DomainLoadStoreParameter 和为 Keytool 这个安全钥匙和证书的管理工具添加新的命令行选项-importpassword。同时，添加和更新了一些关于安全性的 API 来支持 KeyStore 的更新。

d、支持安全的随机数发生器。SecureRandom 这个类的 getInstanceStrong 方法如今可以获取各个平台最强的随机数对象实例，通过这个实例生成像 RSA 私钥和公钥这样具有较高熵的随机数。

e、JSSE（Java(TM) Secure Socket Extension）服务器端开始支持 SSL/TLS 服务器名字识别 SNI（Server Name Indication）扩展。SNI 扩展目的是 SSL/TLS 协议可以通过 SNI 扩展来识别客户端试图通过握手协议连接的服务器名字。

f、安全性比较差的加密方法被默认禁用。默认不支持 DES 相关的 Kerberos 5 加密方法。如果一定要使用这类弱加密方法需要在 krb5.conf 文件中添加 allow\_weak\_crypto=true。考虑到这类加密方法安全性极差，开发者应该尽量避免使用它。

## 7、IO/NIO 的改进

a、改进了 java.nio.charset.Charset 的实现，使编码和解码的效率得以提升，也精简了 jre/lib/charsets.jar 包

b、优化了 String(byte[],\*) 构造方法和 String.getBytes() 方法的性能

c、还增加了一些新的 IO/NIO 方法，使用这些方法可以从文件或者输入流中获取流（java.util.stream.Stream），通过对流的操作，可以简化文本行处理、目录遍历和文件查找。

## 8、全球化功能

a、支持新的 Unicode 6.2.0 标准

b、新增了日历和本地化的 API，改进了日期时间的管理

c、新的 java.time 中包含了所有关于时钟（Clock），本地日期（LocalDate）、本地时间（LocalTime）、本地日期时间（LocalDateTime）、时区（ZonedDateTime）和持续时间（Duration）的类

d、Date 类新增了 toInstant() 方法

# 二、JDK9

https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-9/index.html

https://www.oracle.com/java/technologies/javase/9-relnotes.html

## 1、模块系统

a、在引入了模块系统之后，JDK 被重新组织成 94 个模块

b、Java 应用可以通过新增的 jlink 工具，创建出只包含所依赖的 JDK 模块的自定义运行时镜像

c、这样可以极大的减少 Java 运行时环境的大小，镜像的大小的减少可以节省很多存储空间和带宽资源

d、重要特征是在其工件（artifact）的根目录中包含了一个描述模块的 module-info.class文件，由根目录中的源代码文件 module-info.java 编译而来，包含如下内容：

d.1模块导出的包：使用 exports 可以声明模块对其他模块所导出的包。包中的 public 和 protected 类型，以及这些类型的 public 和 protected 成员可以被其他模块所访问。没有声明为导出的包相当于模块中的私有成员，不能被其他模块使用。

d.2模块的依赖关系：使用requires可以声明模块对其他模块的依赖关系。使用 requires transitive 可 以把一个模块依赖声明为传递的。传递的模块依赖可以被依赖当前模块的其他模块所读取。如果一个模块所导出的类型的型构中包含了来自它所依赖的模块的类型，那么对该模块的依赖应该声明为传递的。

d.3服务的提供和使用：如果一个模块中包含了可以被 ServiceLocator 发现的服务接口的实现 ，需要使用 provides with 语句来声明具体的实现类 ；如果一个模块需要使用服务接口，可以使用 uses 语句来声明。

c、模块系统中增加了模块路径的概念。模块系统在解析模块时，会从模块路径中进行查找。为了保持与之前 Java 版本的兼容性，CLASSPATH 依然被保留。

d、所有的类型在运行时都属于某个特定的模块。对于从 CLASSPATH 中加载的类型，它们属于加载它们的类加载器对应的未命名模块。

e、可以通过 Class 的 getModule()方法来获取到表示其所在模块的 Module 对象。

f、在 JVM 启动时，会从应用的根模块开始，根据依赖关系递归的进行解析，直到得到一个表示依赖关系的图。

g、如果解析过程中出现找不到模块的情况，或是在模块路径的同一个地方找到了名称相同的模块，模块解析过程会终止，JVM 也会退出。

h、Java 也提供了相应的 API 与模块系统进行交互。

## 2、Jshell

a、jshell 为 Java 增加了类似 NodeJS 和 Python 中的读取-求值-打印循环（ Read-Evaluation-Print Loop

b、在 jshell 中 可以直接 输入表达式并查看其执行结果

## 3、集合、Stream 和 Optional

a、增加 了 List.of()、Set.of()、Map.of() 和 M ap.ofEntries()等工厂方法来创建不可变集合

b、Stream 中增加了新的方法 ofNullable、dropWhile、takeWhile 和 iterate

c、Collectors 中增加了新的方法 filtering 和 flatMapping

d、Optiona l 类中新增了 ifPresentOrElse、or 和 stream 等方法

## 4、进程 API

a、增加了 ProcessHandle 接口，可以对原生进程进行管理，尤其适合于管理长时间运行的进程

b、在使用 P rocessBuilder 来启动一个进程之后，可以通过 Process.toHandle()方法来得到一个 ProcessHandl e 对象的实例。

c、通过 ProcessHandle 可以获取到由 ProcessHandle.Info 表 示的进程的基本信息，如命令行参数、可执行文件路径和启动时间等。

d、ProcessHandle 的 onExit()方法返回一个 C ompletableFuture<ProcessHandle>对象，可以在进程结束时执行自定义的动作。

## 5、平台日志 API 和 服务

a、允许为 JDK 和应用配置同样的日志实现

b、新增的 System.LoggerFinder 用来管理 JDK 使 用的日志记录器实现。JVM 在运行时只有一个系统范围的 LoggerFinder 实例

c、LoggerFinder 通 过服务查找机制来加载日志记录器实现

d、我们也可以通过添加自己的 System.LoggerFinder 实现来让 JDK 和应用使用 SLF4J 等其他日志记录框架

## 6、反应式流 （Reactive Streams）

a、反应式流规范的出发点是提供一个带非阻塞负压（ non-blocking backpressure ） 的异步流处理规范

b、反应式流规范的核心接口已经添加到了 Java9 中的 java.util.concurrent.Flow 类中。

c、Flow 中包含了 Flow.Publisher、Flow.Subscriber、Flow.Subscription 和 F low.Processor 等 4 个核心接口。Java 9 还提供了 SubmissionPublisher 作为 Flow.Publisher 的一个实现

## 7、变量句柄

a、变量句柄含义类似于已有的方法句柄，是一个变量或一组变量的引用，包括静态域，非静态域，数组元素和堆外数据结构中的组成部分等。

b、变量句柄由 J ava 类 java.lang.invoke.VarHandle 来表示，可以使用类 java.lang.invoke.MethodHandles.Lookup 中的静态工厂方法来创建 VarHandle 对 象

c、访问模式：通过变量句柄，可以在变量上进行各种操作。

d、不同的访问模式尤其在内存排序上的不同语义。目前一共有 31 种 访问模式，而每种访问模式都 在 VarHandle 中 有对应的方法。这些方法可以对变量进行读取、写入、原子更新、数值原子更新和比特位原子操作等；还可以用来访问数组中的单个元素，以及把 byte[]数组 和 ByteBuffer 当成是不同原始类型的数组来访问。

## 8、改进方法句柄（Method Handle）

a、类 java.lang.invoke.MethodHandles 增加了更多的静态方法来创建不同类型的方法句柄：

a.1、arrayConstructor：创建指定类型的数组。

a.2、arrayLength：获取指定类型的数组的大小。

a.3、varHandleInvoker 和 varHandleExactInvoker：调用 VarHandle 中的访问模式方法。

a.4、zero：返回一个类型的默认值。

a.5、empty：返 回 MethodType 的返回值类型的默认值。

a.6、loop、countedLoop、iteratedLoop、whileLoop 和 doWhileLoop：创建不同类型的循环，包括 for 循环、while 循环 和 do-while 循环。

a.7、tryFinally：把对方法句柄的调用封装在 try-finally 语句中。

## 9、并发

a、类 CompletableFuture 中增加了几个新的方法。

a.1、completeAsync 使用一个异步任务来获取结果并完成该 CompletableFuture。

a.2、orTimeout 在 CompletableFuture 没有在给定的超时时间之前完成，使用 TimeoutException 异常来完成 CompletableFuture。

a.3、completeOnTimeout 与 o rTimeout 类似，只不过它在超时时使用给定的值来完成 CompletableFuture。

a.4新的 Thread.onSpinWait 方法在当前线程需要使用忙循环来等待时，可以提高等待的效率。

## 10、Nashorn

a、已经实现了一些 ECMAScript 6 规范中的新特性，包括模板字符串、二进制和八进制字面量、迭代器 和 for..of 循环和箭头函数等。

b、Nashorn 还提供了 API 把 ECMAScript 源代码解析成抽象语法树（ Abstract Syntax Tree，AST ） ，可以用来对 ECMAScript 源代码进行分析。

## 11、I/O 流新特性

a、java.io.InputStream 中增加了新的方法来读取和复制 InputStream 中包含的数据：

a.1、readAllBytes：读取 InputStream 中的所有剩余字节。

a.2、readNBytes： 从 InputStream 中读取指定数量的字节到数组中。

a.3、transferTo：读取 InputStream 中的全部字节并写入到指定的 OutputStream 中 。

a.4、ObjectInputFilter 可以对 ObjectInputStream 中 包含的内容进行检查，来确保其中包含的数据是合法的。可以使用 ObjectInputStream 的方法 setObjectInputFilter 来设置。

a.5、ObjectInputFilter 在 进行检查时，可以检查如对象图的最大深度、对象引用的最大数量、输入流中的最大字节数和数组的最大长度等限制，也可以对包含的类的名称进行限制。

## 12、改进应用安全性能

a、新增了 4 个 SHA- 3 哈希算法，SHA3-224、SHA3-256、SHA3-384 和 S HA3-512。

b、增加了通过 java.security.SecureRandom 生成使用 DRBG 算法的强随机数。

## 13、JVM 方面

a、JVM 有了统一的日志记录系统，可以使用新的命令行选项-Xlog 来控制 JVM 上 所有组件的日志记录。该日志记录系统可以设置输出的日志消息的标签、级别、修饰符和输出目标等。

b、移除了在 Java 8 中 被废弃的垃圾回收器配置组合，同时 把 G1 设为默认的垃圾回收器实现。另外，**CMS 垃圾回收器已经被声明为废弃**。

c、增加了很多可以通过 jcmd 调用的诊断命令。

## 14、其他

a、Javac不支持jdk1.5以及以前的源码编译

b、String 存储结构变更，String底层存储数据改为byte数组存储数据private final byte[] value

c、新增HttpClientAPI

# 三、JDK10

https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-10/

https://www.oracle.com/java/technologies/10-relnotes.html

## 1、局部变量类型推断

a、局部变量类型推断主要是向 Java 语法中引入在其他语言（比如 C#、JavaScript）中很常见的保留类型名称 var

b、var 不是一个关键字，而是一个保留字

c、仅局限于具有初始化器的局部变量、增强型 for 循环中的索引变量以及在传统 for 循环中声明的局部变量，而不能用于推断方法的参数类型，不能用于构造函数参数类型推断，不能用于推断方法返回类型，也不能用于字段类型推断，同时还不能用于捕获表达式（或任何其他类型的变量声明）

## 2、整合 JDK 代码仓库

a、JDK 10 中将所有现有存储库(root、corba、hotspot、jaxp、jaxws、jdk、langtools、nashorn)合并到一个 Mercurial 存储库中

## 3、统一的垃圾回收接口

a、hotspot/gc 代码实现方面，引入一个干净的 GC 接口，改进不同 GC 源代码的隔离性，多个 GC 之间共享的实现细节代码应该存在于辅助类中

## 4、并行全垃圾回收器 G1

a、为 G1 引入多线程并行 GC，同时会使用与年轻代回收和混合回收相同的并行工作线程数量，从而减少了 Full GC 的发生，以带来更好的性能提升、更大的吞吐量。

b、Java 10 中将采用并行化 mark-sweep-compact 算法，并使用与年轻代回收和混合回收相同数量的线程。具体并行 GC 线程数量可以通过：-XX：ParallelGCThreads 参数来调节，但这也会影响用于年轻代和混合收集的工作线程数。

## 5、应用程序类数据共享

a、类数据共享机制(Class Data Sharing，简称 CDS)，允许将一组类预处理为共享归档文件，以便在运行时能够进行内存映射以减少 Java 程序的启动时间，当多个 Java 虚拟机（JVM）共享相同的归档文件时，还可以减少动态内存的占用量，同时减少多个虚拟机在同一个物理或虚拟的机器上运行时的资源占用。

b、CDS 特性在原来的 bootstrap 类基础之上，扩展加入了应用类的 CDS (Application Class-Data Sharing) 支持：在启动时记录加载类的过程，写入到文本文件中，再次启动时直接读取此启动文本并加载。设想如果应用环境没有大的变化，启动速度就会得到提升。

c、可以想像为类似于操作系统的休眠过程，合上电脑时把当前应用环境写入磁盘，再次使用时就可以快速恢复环境。

## 6、线程-局部管控

a、线程管控引入 JVM 安全点的概念，将允许在不运行全局 JVM 安全点的情况下实现线程回调，由线程本身或者 JVM 线程来执行，同时保持线程处于阻塞状态，这种方式使得停止单个线程变成可能，而不是只能启用或停止所有线程。通过这种方式显著地提高了现有 JVM 功能的性能开销，并且改变了到达 JVM 全局安全点的现有时间语义。

b、增加的参数为：-XX:ThreadLocalHandshakes (默认为开启)，将允许用户在支持的平台上选择安全点。

## 7、移除 Native-Header 自动生成工具

a、当编译 JNI 代码时，已不再需要单独的 Native-Header 工具来生成头文件，因为这可以通过 Java 8（JDK-7150368）中添加的 javac 来完成。在未来的某一时刻，JNI 将会被 Panama 项目的结果取代，但是何时发生还没有具体时间表。

## 8、额外的 Unicode 语言标签扩展

a、自 Java 7 开始支持 BCP 47 语言标记以来， JDK 中便增加了与日历和数字相关的 Unicode 区域设置扩展

b、在 Java 9 中，新增支持 ca 和 nu 两种语言标签扩展。

c、在 Java 10 中将继续增加 Unicode 语言标签扩展，具体为：增强 java.util.Locale 类及其相关的 API，以更方便的获得所需要的语言地域环境信息。同时在这次升级中还带来了如下扩展支持：

A screenshot of a cell phone

Description automatically generated

## 9备用存储装置上的堆分配

a、要在备用设备上进行堆分配，可以使用堆分配参数 -XX:AllocateHeapAt = <path>，这个参数将指向文件系统的文件并使用内存映射来达到在备用存储设备上进行堆分配的预期结果。

## 10、基于 Java 的 实验性 JIT 编译器

a、开启了基于 Java 的 JIT 编译器 Graal, 并将其用作 Linux/x64 平台上的实验性 JIT 编译器开始进行测试和调试工作

b、Graal 是一个以 Java 为主要编程语言、面向 Java bytecode 的编译器。与用 C++实现的 C1 及 C2 相比，它的模块化更加明显，也更加容易维护。Graal 既可以作为动态编译器，在运行时编译热点方法；亦可以作为静态编译器，实现 AOT 编译。

c、默认情况下 HotSpot 仍使用的是 C2 编译器，要启用 Graal 作为 JIT 编译器，请在 Java 命令行上使用以下参数：-XX：+ UnlockExperimentalVMOptions -XX：+ UseJVMCICompiler

## 11、根证书认证

a、作为 JDK 一部分的 cacerts 密钥库旨在包含一组能够用于在各种安全协议的证书链中建立信任的根证书。

b、keytool 中加入参数 -cacerts，可以查看当前 JDK 管理的根证书。

c、从 Java 10 开始，将会在 JDK 中提供一套默认的 CA 根证书。

## 12、基于时间的版本发布模式

a、重新编写之前 JDK 版本中引入的版本号方案，将使用基于时间模型定义的版本号格式来定义新版本。保留与 JEP 223 版本字符串方案的兼容性，同时也允许除当前模型以外的基于时间的发布模型。

b、命名机制是：$FEATURE.$INTERIM.$UPDATE.$PATCH

b.1、$FEATURE，每次版本发布加 1，不考虑具体的版本内容。2018 年 3 月的版本是 JDK 10，9 月的版本是 JDK 11，依此类推。

b.2、$INTERIM，中间版本号，在大版本中间发布的，包含问题修复和增强的版本，不会引入非兼容性修改。

# 四、JDK11

https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-11/

https://www.oracle.com/technetwork/java/javase/11-relnote-issues-5012449.html

## 1、基于嵌套的访问控制

a、嵌套访问控制是一种控制上下文访问的策略，允许逻辑上属于同一代码实体，但被编译之后分为多个分散的 class 文件的类，无需编译器额外的创建可扩展的桥接访问方法，即可访问彼此的私有成员，并且这种改进是在 Java 字节码级别的。

b、引入了两个新的属性：

b.1、NestMembers 的属性，用于标识其它已知的静态 nest 成员；

b.2、NestHost 属性，每个 nest 成员都包含，用于标识出它的 nest 宿主类。

## 2、标准 HTTP Client 升级

a、Http Client 的包名由 jdk.incubator.http 改为 java.net.http，该 API 通过 CompleteableFutures 提供非阻塞请求和响应语义，可以联合使用以触发相应的动作，并且 RX Flow 的概念也得到了实现。

b、提供了对 HTTP/2 等业界前沿标准的支持，同时也向下兼容 HTTP/1.1

## 3、低开销垃圾回收器：Epsilon

a、Epsilon 垃圾回收器的目标是开发一个控制内存分配，但是**不执行任何实际的垃圾回收工作**。它提供一个完全消极的 GC 实现，分配有限的内存资源，最大限度的降低内存占用和内存吞吐延迟时间。

b、通过参数 -XX:+UseEpsilonGC 开启

c、no-op GC 的几个使用场景：性能测试、内存压力测试、VM 接口测试、极度短暂 job 任务、延迟改进、吞吐改进

d、Epsilon 线性分配单个连续内存块。可复用现存 VM 代码中的 TLAB 部分的分配功能。非 TLAB 分配也是同一段代码，因为在此方案中，分配 TLAB 和分配大对象只有一点点的不同。Epsilon 用到的 barrier 是空的(或者说是无操作的)。因为该 GC执行任何的 GC 周期，不用关系对象图，对象标记，对象复制等。

## 4、简化启动单个源代码文件的方法

a、增强 Java 启动器，使之能够运行单一文件的 Java 源代码。允许使用 Java 解释器直接执行 Java 源代码。源代码在内存中编译，然后由解释器执行。唯一的约束在于所有相关的类必须定义在同一个 Java 文件中。

java HelloWorld.java

等同于

javac HelloWorld.java

java -cp . hello.World

## 5、用于 Lambda 参数的局部变量语法

a、允许开发者在 Lambda 表达式中使用 var 进行参数声明

## 6、低开销的 Heap Profiling

a、提供一种低开销的 Java 堆分配采样方法，能够得到堆分配的 Java 对象信息，并且能够通过 JVMTI 访问堆信息。

b、目的：足够低的开销，可以默认且一直开启、能通过定义好的程序接口访问、能够对所有堆分配区域进行采样、能给出正在和未被使用的 Java 对象信息

## 7、支持 TLS 1.3 协议

a、包含了传输层安全性（TLS）1.3 规范（RFC 8446）的实现，替换了之前版本中包含的 TLS，包括 TLS 1.2，同时还改进了其他 TLS 功能，例如 OCSP 装订扩展（RFC 6066，RFC 6961），以及会话散列和扩展主密钥扩展（RFC 7627），在安全性和性能方面也做了很多提升。

b、KRB5​​密码套件实现已从 Java 11 中删除

c、TLS 1.3 与以前的版本不直接兼容。

## 8、可伸缩低延迟垃圾收集器：ZGC(实验阶段)

a、ZGC 即 Z Garbage Collector（垃圾收集器或垃圾回收器），是一个可伸缩的、低延迟的垃圾收集器，主要为了满足如下目标进行设计：

a.1GC 停顿时间不超过 10ms

a.2即能处理几百 MB 的小堆，也能处理几个 TB 的大堆

a.3应用吞吐能力不会下降超过 15%（与 G1 回收算法相比）

a.4方便在此基础上引入新的 GC 特性和利用 colord

a.5针以及 Load barriers 优化奠定基础

a.6当前只支持 Linux/x64 位平台

b、作为实验性功能的 ZGC 将不会出现在 JDK 构建中，除非在编译时使用 configure 参数：--with-jvm-features=zgc显式启用。并配置JVM 参数:-XX:+UnlockExperimentalVMOptions和-XX:+UseZGC -Xmx10g

## 9、飞行记录器

a、飞行记录器之前是商业版 JDK 的一项分析工具，但在 Java 11 中，其代码被包含到公开代码库中

b、飞行记录器类似飞机上的黑盒子，是一种低开销的事件信息收集框架，主要用于对应用程序和 JVM 进行故障检查、分析。飞行记录器记录的主要数据源于应用程序、JVM 和 OS，这些事件信息保存在单独的事件记录文件中，故障发生后，能够从事件记录文件中提取出有用信息对故障进行分析。

c、启用飞行记录器参数：java -XX:StartFlightRecording=duration=1s, filename=recording.jfr；

d、也可以使用 bin/jcmd 工具启动和配置飞行记录器:

d.1、jcmd <pid> JFR.start

d.2、jcmd <pid> JFR.dump filename=recording.jfr

d.3、jcmd <pid> JFR.stop

## 10、动态类文件常量

a、支持新的常量池：CONSTANT\_Dynamic，它在初始化的时候，像 invokedynamic(这是个很重要的指令)

# 五、JDK12

https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-12/

https://www.oracle.com/technetwork/java/javase/12-relnote-issues-5211422.html

## 1、Shenandoah：一个低停顿垃圾收集器（实验阶段）

a、一种低停顿时间的垃圾收集器，原理是通过与 Java 应用程序中的执行线程同时运行，用以执行其垃圾收集、内存回收任务，通过这种运行方式，给虚拟机带来短暂的停顿时间。

b、需要使用参数：- XX:+UnlockExperimentalVMOptions 启用

## 2、增加一套微基准套件

a、基于 JMH（Java Microbenchmark Harness），使开发人员可以轻松运行现有的微基准测试并创建新的基准测试，其目标在于提供一个稳定且优化过的基准，其中包括将近 100 个基准测试的初始集合，并且能够轻松添加新基准、更新基准测试和提高查找已有基准测试的便利性。

b、要构建微基准套件，用户需要运行命令：make build-microbenchmark，类似的命令还有：make test TEST="micro:java.lang.invoke" 将使用默认设置运行 java.lang.invoke 相关的微基准测试。

## 3、Switch 表达式扩展（预览功能）

a、看图吧：

A close up of a logo

Description automatically generated

b、Switch 表达式支持下面类型： byte、char、short、int、Byte、Character、Short、Integer、enum、tring

## 4、引入 JVM 常量 API

a、引入 JVM 常量 API，用来更容易地对关键类文件 (key class-file) 和运行时构件（artefact）的名义描述 (nominal description) 进行建模，特别是对那些从常量池加载的常量

## 5、改进 AArch64 实现

## 6、使用默认类数据共享（CDS）存档

## 7、改善 G1 垃圾收集器，使其能够中止混合集合

## 8、增强 G1 垃圾收集器，使其能自动返回未用堆内存给操作系统

# 六、JDK13

https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-13/

https://www.oracle.com/technetwork/java/javase/13-relnote-issues-5460548.html

## 1、动态应用程序类-数据共享

a、进行了进一步的简化、改进和扩展，即：允许在 Java 应用程序执行结束时动态进行类归档，具体能够被归档的类包括：所有已被加载，但不属于默认基层 CDS 的应用程序类和引用类库中的类。

## 2、增强 ZGC

a、释放未使用内存给操作系统

b、支持最大堆大小为 16TB

c、添加参数：-XX:SoftMaxHeapSize 来软限制堆大小

d、ZGC 内存释放功能，默认情况下是开启的，不过可以使用参数：-XX：-ZUncommit 显式关闭，同时如果将最小堆大小 (-Xms) 配置为等于最大堆大小 (-Xmx)，则将隐式禁用此功能。

e、使用参数：-XX：ZUncommitDelay = <seconds>（默认值为 300 秒）来配置延迟释放，此延迟时间可以指定释放多长时间之前未使用的内存。

## 3、Socket API 重构

a、引入 NioSocketImpl 的实现用以替换 SocketImpl 的 PlainSocketImpl 实现，此实现与 NIO（新 I/O）实现共享相同的内部基础结构，并且与现有的缓冲区高速缓存机制集成在一起，因此不需要使用线程堆栈

b、在引入新实现方法的同时，之前版本的实现还未被移除，可以通过使用下列系统属性以重新使用原实现方法：-Djdk.net.usePlainSocketImpl = true

## 4、Switch 表达式扩展（预览功能）：yield

a、Switch 表达式（返回值）应该使用 yield，而 Switch 语句（不返回值）应该使用 break

b、看图吧：

A picture containing bird

Description automatically generated

## 5、文本块（预览功能）

a、启用预览功能：

a.1、javac --enable-preview --release 13 Example.java

a.2、java --enable-preview Example

b、看图

A picture containing bird, flower

Description automatically generated

# 七、JDK14

https://www.oracle.com/technetwork/java/javase/14-relnote-issues-5809570.html

instanceof的模式匹配（预览）

## 1、ZGC支持Windows/macOS

## 2、弃用ParallelScavenge + SerialOld GC组合

## 3、instanceof 运算符引入模式匹配（预览）

A screenshot of a cell phone

Description automatically generated

## 4、NullPointerExceptions增强

Exception in thread "main" java.lang.NullPointerException:

Cannot read field "c" because "a.b" is null

## 5、switch表达式标准化

## 6、文本块第二版预览