Java9 新特性

翻译自：https://docs.oracle.com/javase/9/whatsnew/toc.htm#JSNEW-GUID-5B808B2F-E891-43CD-BF6E-78787E547071

Key Changes in JDK9

1. Java平台模块化系统

引入了一种新的Java编程组建 -- 一个被命名且自描述的代码和数据的集合 -- 模块.
在新的模块系统中:

1. 引入了一个新的可选的阶段, link-time. link-time是介于compile-time和run-time之间的,
   在link-time 期间, 一组module可以被打包和优化进一个自定义的运行时镜像(小型容器?).

详情见 jlink tool in Java Platform, Standard Edition Tools Reference

1. 在java的工具javac, jlink 和java中添加了新的option, 用来指定特定的module paths. Module path 定位了 Module 的定义.

2. 引入了模块化的JAR文件(在根目录中有module-info.class)

3. 引入了JMOD格式. 这个格式是一种和JAR相似的打包格式, 但是他可以包含native code和配置文件. 详情见 jmod tool

JDK模块本身已经被分成了一组模块. 变化如下:

1. 允许你把JDK的 module 和各种配置文件结合起来, 包括:

> 有关JRE和JDK的配置
> 类似于Java8中的Compact Profiles的配置
> 自定义的配置,这些配置中包含了一组特定的Module以及它们所依赖的module

1. 重构了JDK和JRE的运行时镜像从而容纳module并且提升了性能, 安全以及可靠性.

2. 定义了一种新的URI模式用来命名存储在运行时镜像中的模块,类和资源, 从而不用暴露运行时镜像的内部结构和格式

3. 去掉了endorsed-standards override 机制(覆盖系统api) 和 扩展机制(添加系统默认api).

4. 从Java运行时镜像中去掉了rt.jar和tools.jar

5. 大多数的JDK内部API都默认不可见了(inaccessible), 但是留下来的一些极其重要 并 广泛被使用的内部API. 可以运行jdeps -jdkinternals 来决定代码是否使用JDK内部API

2. 新的版本格式

提供了一个简化的版本字符串格式从而更加清楚的区分major, minor, security 和 patch 的更新新版本. 新的版本格式如下:

> $MAJOR.$MINOR.$SECURITY.$PATCH

1. $MAJOR是主版本号, 例如JDK9

2. #MINOR是每次小型更新的版本号,例如bug fix, 标准api的修订 或者是一些平台说明文档之外的特性的实现

3. $SECURITY是安全更新版本的版本号.

4. $PATCH是包含了安全和高优先级的修补的版本的版本号

What's New for the JDK 9 Installer

略

What's New for Tools in JDK 9

JShell

在Java平台中添加了REPL功能. JShell提供了一个交互的命令行接口用来在Java编程平台上运行并计算declarations, statements, 和 expressions. 这个功能可以立即获得代码的运算结果或者反馈, 从而促进代码的开发以及为Java学习提供帮助.

详情见 jshell in Java Platform, Standard Edition Tools Reference 以及 Introduction to JShell in Java Platform, Standard Edition Java Shell User's Guide.

Add More Diagnostic Commands

添加了更多的诊断命令从而提高诊断JDK和HotSpot的能力

详情见 jcmd in Java Platform, Standard Edition Tools Reference.

Remove Launch-Time JRE Version Selection

去除了请求一个和加载时jre版本不同的jre版本的功能.

现代的应用大多数通过Java Web Start, 本地OS 打包系统或者是安装包进行部署. 这些技术有他们自己的方法来管理所需要的JRE 版本, 例如查找, 下载或者更新等. 因此 Launch-Time JRE Version Selection 这个功能已经过时了.

Multi-Release JAR Files

扩展了JAR文件格式，使得针对不同Java发行版的class文件可以在一个JAR中共存。

一个多版本(multirelease JAR -- MRJAR)JAR文件包含了额外的版本化的目录结构，用来存放针对不同Java平台版本的类文件和资源文件。 可以通过jar工具的 --release选项来指定版本目录。

Remove the JVM TI hprof Agent

移除了JDK中的hprof agent代理。 Hprof agent 是作为JVM工具接口的实例代码的， 本来并没有被设计成一个产品级的工具。

Hprof agent中有用的功能已经被其他替代产品所替代了。

NOTE:

> 虽然hprof被替代了， 但是用jmap或者其他诊断工具依旧可以创建一份hprof格式的heap dump

Remove the jhat Tool

从JDK中移除了jhat命令。

jhat命令是在JDK6中加入的一个实验性的并且unsupported的工具。 它已经过时了;更高级的堆可视化工具和分析工具已经有很多年历史了。

Validate JVM Command-Line Flag Arguments

为了避免发生错误， 将会检查JVM中所有数值化的命令行参数。 如果参数不合法， 会输出一段对应的错误信息。

范围以及选项限制检查已经被应用到所有需要用户指定的数值化的命令行参数上了。

详情见java和 Validate Java Virtual Machine Flag Arguments

Compile for Older Platform Versions

改进了javac命令， 从而使得它可以编译出跑在9之前版本的Java平台上的程序。

当使用 -source 或者 -target命令的时候， 编译出的程序可能偶尔会调用先前版本不支持的API. 可以使用 --release选型来防止出现这个问题。

详情见 javac

jlink: The Java Linker

收集和优化一组模块以及它们的依赖， 并形成一个自定义的运行时镜像 （defined in JEP 220）。

jlink工具为装配阶段的转化和优化以及替代镜像格式的生成定义了一个插件机制。 它可以为一个特定的程序生成并优化出一个自定义的运行时。 JEP 261定义了一个 link-time(链接时？) 的概念, 作为一个介于编译时和运行时这两个阶段之间的一个可选的阶段。 Link-Time需要一个Linking工具来装配并优化一组模块以及他们相关的依赖， 从而生成一个运行时镜像或是可执行文件。

详情见jlink

What’s New for Security in JDK 9

Datagram Transport Layer Security (DTLS)

添加了Java Secure Socket Extension(JSSE) API 和 SunJSSE 的安全实现， 从而支持了DTLS Version 1.0 和 DTLS Version 1.2 这两个协议。

详情见 Datagram Transport Layer Security (DTLS) in Java Platform, Standard Edition Security Developer's Guide.

TLS Application-Layer Protocol Negotiation Extension

允许客户端和服务器端在一个TLS连接中协商所要用的应用层协议。 在应用层协议协商汇中(Application-Layer Protocol Negotiation, ALPN), 客户端在 TLS ClientHello中发送一个支持的应用协议的列表。 服务器端选择一个协议并把这个选择作为TLS ServerHello的一部分内容发送回去。 ALPN可以在TLS的握手阶段完成， 不会对网络造成负担。

详情见TLS Handshake and Application Layer Protocol Negotiation in Java Platform, Standard Edition Security Developer's Guide.

OCSP Stapling for TLS

(OCSP : Online Certificate Status Protocol 在线证书状态协议， 向一个CA颁发机构询问证书的合法性。 一般是由Client发起的。 OCSP Stapling ： 由Server向OCSP URL询问证书的合法性并发给客户端。 主要是为了提速)

允许TLS链接中的Server检查一个X.509证书是否已经废弃。 在TLS的握手阶段中， Server会联系(contact)一个OSCP的响应方来查询一个正在协商中的证书。 之后Server会把有关该证书的废弃信息附加进返回给客户端的证书中， 这样客户端就可以进行相应的操作。

允许客户端向一个TLS Server发起 OCSP Stapling的请求。客户端会检查从支持这个功能的Server所返回的信息。

详情见See OCSP Stapling in Java Platform, Standard Edition Security Developer's Guide.

Leverage CPU Instructions for GHASH and RSA

使用HotSpot的 GHASH 的原语把 AES/GSM/NoPadding算法的效率提升了34倍到150倍。 在Intel x64 CPU 上的PCLMULQDQ指令和SPARC上的xmull/xmulhi指令都提高了GHASH原语的速度。

使用HotSpot的RSA原语把 BigInteger.squareToLen 和 BigInteger.mulAdd 方法的性能提升了50%. RSA原语适用于Intel X64上的java.math.BigInteger类。

引入了一个新的安全属性jdk.security.provider.preferred来配置针对特定的算法进行显著的性能提升的provider

详情见See Configuring the Preferred Provider for Specific Algorithms in Java Platform, Standard Edition Security Developer's Guide.

DRBG-Based SecureRandom Implementations

提供了在 SecureRandom API中提到的NIST SP 800-90Ar1所说的伪随机数生成器机制的函数功能

DRGB机制使用了与SHA-512和AES-256一样可靠的现代算法。 这些机制可以被配置成不同的安全强度和特性， 从而满足用户的需求。

Disable SHA-1 Certificates

通过提供了一种更加灵活的机制来禁用了基于SHA-1签名的X.509证书链， 从而提升了JDK的安全配置。

禁用了JDK中默认包含的TLS Server中利用根证书定位的证书链中的SHA-1算法。 本地或者企业的证书认证不受影响。

在jdk.certpath.disabledAlgorithms中添加了一些新的安全限制从而提升其安全属性。 这些限制扩大了对可能被禁用的证书的控制。

Create PKCS12 Keystores by Default

默认的keystore类型从JKS修改成了PKCS12. PKCS12是一个可扩展的标准的并且被广泛支持的存储加密密钥的存储格式。PKCS12 keystore通过存储私钥可信公钥证书以及密钥来提高保密性。这个特性还提供了与例如Mozilla, IE, OpenSSL以及其他支持PKCS12的系统交互的机会。

SunJEES 提供了一个PKCS12的完整实现。 java.security.KeyStore可以用来读写PKCS12文件。

详情见Key Management in Java Platform, Standard Edition Security Developer's Guide.

可以用密钥和证书管理组件keytool来创建 PKCS12 keystores.

详情见Creating a Keystore in Java Platform, Standard Edition Security Developer's Guide 和 keytool in Java Platform, Standard Edition Tools Reference.

SHA-3 Hash Algorithms

支持了SHA-3加密哈希算法。
java.security.MessageDigest 还支持了这些额外的标准算法：SHA3-224, SHA3-256, SHA3-384 和 SHA3-512

下面的提供商支持SHA-3算法：

```
1. SUN provider: SHA3-224, SHA3-256, SHA3-384, and SHA3-512

2. OracleUcrypto provider: SHA-3 digests supported by Solaris 12.0
```

What’s New for Deployment in JDK 9

Deprecate the Java Plug-in

在Oracle JDK9中反对Java Plug-in 和相关的applet技术的应用。 虽然这些技术在 JDK9中依旧是可以使用的， 但是这些技术在未来

的版本中会被考虑删除。

网页中嵌入的Applet和JavaFX应用需要Java Plug-in来运行， 所以请考虑用 Java Web Start或者其他技术重写这些应用。

详情见 Migrating Java Applets to Java Web Start and JNLP 和 Self-Contained Application Packaging in Java Platform, Standard Edition Deployment Guide.

Enhanced Java Control Panel

提升了Java Control Panel中options的分类和展示。信息可以被更方便的定位， 提供了搜索框， 并且 Modal Dialog Boxes 被废弃掉了。 注意一些option的位置和先前版本的 Java Control Panel中不一致。

详情见 Java Control Panel in Java Platform, Standard Edition Deployment Guide.

Modular Java Application Packaging

把Jigsaw 项目中的特性整合进了Java Packager, 包括模块化的意识和自定义运行时的创建。

可以利用jlink工具来创建更小的 package

只能创建只使用了JDK9运行时的应用。 不用用来打包使用早期版本的JRE的应用。

详情见Customization of the JRE和Packaging for Modular Applications in Java Platform, Standard Edition Deployment Guide

Deprecate the Applet API

移除了Applet API. 这玩意已经过时了。

What’s New for the Java Language in JDK 9

Java9中语法的改变很小。只有以下几点改变：

1. 允许在私有实例方法上使用@SafeVargs注解
2. 允许事实上的final变量在try-w ith-resource语句中作为resource来使用, 这种情况下不再需要指定新的资源
3. 允许在菱形括号内使用匿名类， 如果有关的参数类型可以被标识出来的话
4. 单独的下划线不再可以作为一个标识符
5. 添加了接口私有方法的支持

   详情见 (Java Language Changes for Java SE 9)[https://docs.oracle.com/javase/9/language/toc.htm#JSLAN-GUID-16A5183A-DC0D-4A96-B9D8-AAC9671222DD]

What’s New for Javadoc in JDK 9

Simplified Doclet API

用一种新的简化的API替代了旧的Doclet API. 标准doclet已经用新的Doclet API重写了.

HTML5 Javadoc

支持HTML5格式的输出。 为了确保和HTML5兼容， 请确保文档注释中的内容和H5兼容.

Javadoc Search

提供了一个搜索框来搜索API文档。 可以使用这个搜索框去查找程序元素， 关键词以及文档中的短语。

Module System

支持模块声明中的文档注释。 添加了新的命令行选项，用于配置哪些模块需要被纳入文档以及这些模块的总结。

What's New for the JVM in JDK 9

Compiler Control

通过编译指令选项提供了一种控制JVM编译的方式。 控制的级别有 runtime-managable 和 method-specific. Compiler Control 替代了并且向后兼容CompileCommand.(CompileCommand是之前的选项， 这个选项向后兼容Compiler Control)

详情见[Compiler Control in Java Platform, Standard Edition Java Virtual Machine Guide.](Compiler Control in Java Platform, Standard Edition Java Virtual Machine Guide.)

Segment Code Cache

将代码缓存划分到不同的独立的段中。 每个段都包含了一个特定类型的已编译好的代码， 从而提升了性能， 为未来的扩展提供可能。

详情见java in Java Platform, Standard Edition Tools Reference.

Dynamic Linking of Language-Defined Object Models

在运行时动态链接高级别的对象操作(比如读写属性、 调用函数) 到合适的目标方法处理器(target method handle). 在链接时是基于传递的参数的实际类型。这些对象操作是由 invokedynamic (java字节码指令)来实现的。

当java.lang.invoke提供了一个相对于invokedynamic的调用点低级别的API的时候， 并不提供有关的高级别的对象操作的语句， 也不提供实现的方法。

通过jdk.dylnalink, 你可以实现包含动态类型和利用invokedynamic进行调用的方法的编程语言。

What's New for JVM Tuning in JDK 9

Improve G1 Usability, Determinism, and Performance

优化了G1垃圾收集器， 从而可以是的它自动决定一些重要的内存回收设置。 之前这些设置必须手动设置从而获得优化的结果。除此之外， 修复了G1垃圾收集器的一些可用性， 确定性和性能上的问题。

Unified JVM Logging

引入了一个在JVM各个组件中通用的日志系统。

详情见 -Xloggc java option in Java Platform, Standard Edition Tools Reference.

Remove GC Combinations Deprecated in JDK 8

去掉了JDK8中过时的垃圾收集器的组合。

以下垃圾收集器的组合不再存在了:

1. DefNew + CMS

2. ParNew + SerialOld
3. Incremental CMS

   CMS(Concurrent Mark Sweep)的"前台"模式也被去掉了。

   下列命令参数也被去掉了:

4. -Xincgc
5. -XX:+CMSIncrementalMode
6. -XX:+UseCMSCompactAtFullCollection
7. -XX:+CMSFullGCsBeforeCompaction
8. -XX:+UseCMSCollectionPassing

   命令行参数 -XX:+UserParNew GC 也不再起作用了。 ParNew 现在只可以和CMS一起使用， 并且CMS也必须要求ParNew. 因此 -XX:+UseParNew GC 选项已经过时了， 在未来的版本中考虑去掉这个命令。

Make G1 the Default Garbage Collector

把G1垃圾收集器设置为32位和64位Server虚拟机上的默认垃圾收集器。 使用低停顿的垃圾收集器(比如G1)可以比面向吞吐量的垃圾收集器(比如Parallel GC)提供更好的性能和体验.

详情见Garbage-First Garbage Collector in Java Platform, Standard Edition HotSpot Virtual Machine Garbage Collection Tuning Guide

Unified GC Logging

使用统一的JVM日志框架重新实现了GC的日志系统。 GC日志的重新实现和当前的GC日志的风格保持了一致， 但是依旧有一些不同。

详情见Enable Logging with the JVM Unified Logging Framework in Java Platform, Standard Edition Tools Reference.

Deprecate the Concurrent Mark Sweep (CMS) Garbage Collector

不再赞成使用CMS收集器。 当使用-XX:+UseConcMarkSweepGC选项请求使用CMS的时候会有一条警告message。 G1垃圾收集器在多数情况下已经是CMS的替代品了。

What’s New for Core Libraries in JDK 9

Process API Updates

优化了控制和管理操作系统本地进程的API。

ProcessHandle类提供了进程的本地ID， 参数， 命令， 开始时间， 占用CPU时间，用户， 父进程以及子进程。这个类也可以监控进程的活跃度以及摧毁进程。 有了ProcessHandle.onExit方法， ComputableFuture类的异步机制可以在进程结束吼采取相应的动作。

详情见Process API in Java Platform, Standard Edition Java Core Libraries Developer's Guide, java.lang.Process 和 java.lang.ProcessHandle

Variable Handles

定义了一个标准的方式， 用来在对象fields和数组元素上调用与 java.util.concurrent.atomic 和 sun.misc.Unsafe等价的操作.

定义了一些标准的界定操作(fence operation)， 这其中包含 VarHandle这个静态方法， 可以支持对内存组合(memory ordering?)的细粒度的控制。 这是sun.misc.Unsafe的一种替代， 前者提供了非标准的界定操作。

定义了一个标准的可达性界定操作，来确保一个被引用的对象保持强可达性。

Compact Strings

在String中采用了一种新的更加具有空间效率的内部实现。 先前String类使用char的数组来存储一个字符串的。，每个字符用两个Byte来存储(16bits)， 新的内部实现是用了byte的数组加上一个编码属性。

这仅仅是一个实现上的改变， 不会影响到公共的接口。

详情见CompactStrings option of the java command in Java Platform, Standard Edition Tools Reference.

Platform Logging API and Service

定义了一组最小化的API， 平台类可以用这些API来记录消息。 同时也为消息的消费者提供了一个服务接口。 一个库或者应用程序可以提供这些服务的实现， 从而把平台的日志信息发送至所选择的日志框架上。 如果没有自定义的实现， 那么就使用java.util.logging的默认实现。

More Concurrency Updates

添加了JEP 155: Concurrency Updates中提到的并发升级， 还在CompletableFuture API中提供了一个互操作的发布订阅的框架。

XML Catalogs

添加了一个支持Organization for the Advancement of Structured Information Standards (OASIS) XML Catalogs version 1.1标准的XML Catalog API。 这个API定义了catalog 和 catalog-resolver abstractions， 可以在接受解析器的JAXP processor中作为固有的或者扩展的解析器。

现有的库或者应用程序中使用的内部catalog API需要迁移到新的API从而使用这个新特性
。

详情见 XML Catalog API in Java Platform, Standard Edition Java Core Libraries Developer's Guide

Convenience Factory Methods for Collections

修改了collections 和 maps 使得创建包含少量元素的集合对象更加容易。List, Set 和 Map中 新的静态工厂方法可以更简单的创建不可变的集合的实例。

例如：

```
java Set<String> alphabet = Set.of("a", "b", "c");
```

详情见 Creating Immutable Lists, Sets, and Maps in Java Platform, Standard Edition Java Core Libraries Developer's Guide。

API文档见Immutable Set Static Factory Methods, Immutable Map Static Factory Methods 和 Immutable List Static Factory Methods

Enhanced Method Handles

增强了java.lang.invoke中的 MethodHandle, MethodHandles 和 MethodHandles.Lookup类， 从而简化了使用并且允许更好的编译优化。

增强了以下的内容：

1. 在java.lang.invoke中的MethodHandles类中， 为try/finally语句块提供了新的MethodHandle combinators。
2. 用新的用于处理参数的 MethodHandle combinators 来增强了 MethodHandle 和 MethodHandles
3. 再MethodHandles.Lookup 类中实现了新的接口方法查询以及可选的父类解析器。

Enhanced Deprecation

改造了@Deprecated注解从而为这个状态和API中所期望的处置操作提供了更好的信息。 添加了以下两个新的元素：

```
1. @Deprecated(forRemoval=true) 申明了这个API在未来的Java SE发行版本中被删除
2. @Deprecated(since="version") 包含了Java SE版本字符串， 申明了这个API从哪个版本以后开始过期
```

例如： @Deprecated(since="9", forRemoval=true)

在核心平台中的@Deprecated已经被升级了。

可以使用新工具 jdeprescan 来扫描一个类库中过时的API

详情见Enhanced Deprecation in Java Platform, Standard Edition Java Core Libraries Developer's Guide 和 jdperscan in Java Platform, Standard Edition Tools Reference

Spin-Wait Hints

定义了一个API， 允许Java代码提示一个自旋循环正在执行中。 一个自旋循环重复的检查一个条件是否是True, 比如是否可以获得一个锁， 然后可以安全的在锁释放之后做一些计算的操作。 这个API只是一个提示作用， 并且没有语法上的要求。

详情见the method Thread.onSpinWait

Filter Incoming Serialization Data

允许对序列化对象反序列化时的输入流进行过滤来提高安全性和鲁棒性。 对象序列化的客户端可以更好的验证它们的输入， 并且导出的RMI对象可以更好的验证调用参数

序列化客户端实现一个过滤器接口， 这个接口被设置到ObjectInputStream 上。 对RMI来说， 通过RemoteServerRef来导出的对象可以把过滤器设置在MarshalInputStream 上来验证调用参数。

Stack-Walking API

提供了一个 stack-w alking API，允许更加以简单的和懒加载的方式对堆栈中的数据进行访问。

这个API提供了两种遍历方式。 第一种是短遍历， 只停在符合条件的栈帧上。 另一种是长遍历， 会遍历整个堆栈。 短遍历避免了检查整个堆栈如果调用者只对堆栈顶层的几个帧感兴趣。 当Stack Walker 被相应配置后， API允许对类对象进行访问。

详情见java.lang.Stackw alker.

Merge Selected Xerces 2.11.0 Updates into JAXP

升级了JDK使得可以支持 2.11.0 版本的 Xerces parser。 这对public 的 JAXP API没有影响。

以下是对Xerces 2.11.0的改变： Datatypes, DOM L3 Serializer, XPointer, Catalog Resolver, 和 XML Schema Validation(包括修改 bug, 但是不包括XML Schema 1.1的开发代码)

What's New for Nashorn in JDK 9

Parser API for Nashorn

允许应用程序， 在特定的IDEs 和 服务端框架中， 解析和分析ECMAScript的代码。

可以利用Parser类中的方法解析来自字符串， URL 或者文件中的ECMAScript代码。 这些方法会返回一个CompilationUnitTree对象， 这个对象用抽象语法树的形式表达了一段ECMAScript的代码。

Implement Selected ECMAScript 6 Features in Nashorn

实现了ES6中许多新的特性， 包括：

1. Template Strings
2. let, const 和 block scope
3. 迭代器(iterator)和 for...of循环
4. Map, Set, WeakMap 和 WeakSet
5. Symbols
6. 二进制和八进制常量

What’s New for Client Technologies in JDK 9

略(这块没有接触过...)

What’s New for Internationalization in JDK 9

Unicode 8.0

支持了 Unicode 8.0.

Unicode6.3, 7.0, 8.0一共引入了10,555个character， 29个script和42个block. 所有的这些在JDK9中都支持。

CLDR Locale Data Enabled by Default

使用了 Common Locale Data Repository's (CLDR) 基于XML的本地时间。 这个特性在JDK8中被首次引入， 在JDK9中成为默认的本地时间。先前版本中， 默认的是JRE。

为了与JDK8兼容， 请在系统属性java.locale.providers中的CLDR前加上COMPAT

详情见CLDR Locale Data Enabled by Default in Java Platform, Standard Edition Internationalization Guide

UTF-8 Properties Files

可以加载UTF-8编码的属性文件。 在先前版本中， 使用ISO-8859-1编码加载属性资源。 UTF-8是一种更加广泛的方式用来表示非拉丁字符。

绝大多数现有的属性文件都不会受到影响。

详情见UTF-8 Properties Files in Java Platform, Standard Edition Internationalization Guide.