大家好，我是Guide哥！这篇文章来自读者的投稿，经过了两次较大的改动，两周的完善终于完成。Java 8新特性见这里：Java8新特性最佳指南 。

Guide 哥：别人家的特性都用了几年了，我 Java 才出来，哈哈！真实！

Java9

发布于 2017 年 9 月 21 日 。作为 Java8 之后 3 年半才发布的新版本，Java 9 带 来了很多重大的变化其中最重要的改动是 Java 平台模块系统的引入,其他还有诸如集合、Stream 流

Java 平台模块系统

Java 平台模块系统，也就是 Project Jigsaw，把模块化开发实践引入到了 Java 平台中。在引入了模块系统之后，JDK 被重新组织成 94 个模块。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。这样可以极大的减少 Java 运行时环境的大小。

Java 9 模块的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。 工件的格式可以是传统的 JAR 文件或是 Java 9 新增的 JMOD 文件。

Jshell

jshell 是 Java 9 新增的一个实用工具。为 Java 提供了类似于 Python 的实时命令行交互工具。

在 Jshell 中可以直接输入表达式并查看其执行结果

集合、Stream 和 Optional

- 增加 了 `List.of()`、`Set.of()`、`Map.of()` 和 `Map.ofEntries()`等工厂方法来创建不可变集合，比如 `List.of("Java", "C++");`、`Map.of("Java", 1, "C++", 2)`;（这部分内容有点参考 Guava 的味道）
- `Stream` 中增加了新的方法 `ofNullable`、`dropWhile`、`takeWhile` 和 `iterate` 方法。`Collectors` 中增加了新的方法 `filtering` 和 `flatMapping`
- `Optional` 类中新增了 `ifPresentOrElse`、`or` 和 `stream` 等方法

进程 API

Java 9 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

平台日志 API 和服务

Java 9 允许为 JDK 和应用配置同样的日志实现。新增了 `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。

我们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。

反应式流 （ Reactive Streams ）

- 在 Java9 中的 `java.util.concurrent.Flow` 类中新增了反应式流规范的核心接口
- Flow 中包含了 `Flow.Publisher`、`Flow.Subscriber`、`Flow.Subscription` 和 `Flow.Processor` 等 4 个核心接口。Java 9 还提供了`SubmissionPublisher` 作为`Flow.Publisher` 的一个实现。

变量句柄

- 变量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
- 变量句柄的含义类似于已有的方法句柄`MethodHandle`
- 由 Java 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

改进方法句柄（Method Handle）

- 方法句柄从 Java7 开始引入，Java9 在类`java.lang.invoke.MethodHandles` 中新增了更多的静态方法来创建不同类型的方法句柄

其它新特性

- **接口私有方法**： Java 9 允许在接口中使用私有方法
- **try-with-resources 增强**： 在 try-with-resources 语句中可以使用 effectively-final 变量（什么是 effectively-final 变量，见这篇文章 http://ilkinulas.github.io/programming/java/2016/03/27/effectively-final-java.html）
- **类 `CompletableFuture` 中增加了几个新的方法（`completeAsync`，`orTimeout` 等）**
- **Nashorn 引擎的增强**： Nashorn 从 Java8 开始引入的 JavaScript 引擎，Java9 对 Nashorn 做了些增强，实现了一些 ES6 的新特性
- **I/O 流的新特性**： 增加了新的方法来读取和复制 InputStream 中包含的数据
- **改进应用的安全性能**： Java 9 新增了 4 个 SHA- 3 哈希算法，SHA3-224、SHA3-256、SHA3-384 和 SHA3-512
- ……

Java10

发布于 2018 年 3 月 20 日，最知名的特性应该是 var 关键字（局部变量类型推断）的引入了，其他还有垃圾收集器改善、GC 改进、性能提升、线程管控等一批新特性

var 关键字

- **介绍** :提供了 var 关键字声明局部变量：`var list = new ArrayList<String>(); // ArrayList<String>`
- **局限性**： 只能用于带有构造器的**局部变量**和 for 循环中

Guide 哥：实际上 Lombok 早就体用了一个类似的关键字，使用它可以简化代码，但是可能会降低程序的易读性、可维护性。一般情况下，我个人都不太推荐使用。

不可变集合

list，set，map 提供了静态方法`copyOf()`返回入参集合的一个不可变拷贝（以下为 JDK 的源码）

```
static <E> List<E> copyOf(Collection<? extends E> coll) {
    return ImmutableCollections.listCopy(coll);
}
```

`java.util.stream.Collectors`中新增了静态方法，用于将流中的元素收集为不可变的集合

Optional

- 新增了`orElseThrow()`方法来在没有值时抛出异常

并行全垃圾回收器 G1

从 Java9 开始 G1 就了默认的垃圾回收器，G1 是以一种低延时的垃圾回收器来设计的，旨在避免进行 Full GC，但是 Java9 的 G1 的 FullGC 依然是使用单线程去完成标记清除算法,这可能会导致垃圾回收期在无法回收内存的时候触发 Full GC。

为了最大限度地减少 Full GC 造成的应用停顿的影响，从 Java10 开始，G1 的 FullGC 改为并行的标记清除算法，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。

应用程序类数据共享

在 Java 5 中就已经引入了类数据共享机制 (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

Java 10 在现有的 CDS 功能基础上再次拓展，以允许应用类放置在共享存档中。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。其原理为：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

其他特性

- **线程-局部管控**：Java 10 中线程管控引入 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
- **备用存储装置上的堆分配**：Java 10 中将使得 JVM 能够使用适用于不同类型的存储机制的堆，在可选内存设备上进行堆内存分配
- **统一的垃圾回收接口**：Java 10 中，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。统一垃圾回收接口的主要原因是：让垃圾回收器（GC）这部分代码更加整洁，便于新人上手开发，便于后续排查相关问题。

Java11

Java11 于 2018 年 9 月 25 日正式发布，这是很重要的一个版本！Java 11 和 2017 年 9 月份发布的 Java 9 以及 2018 年 3 月份发布的 Java 10 相比，其最大的区别就是：在长期支持(Long-Term-Support)方面，**Oracle 表示会对 Java 11 提供大力支持，这一支持将会持续至 2026 年 9 月。这是据 Java 8 以后支持的首个长期版本。**

字符串加强

Java 11 增加了一系列的字符串处理方法，如以下所示。

Guide 哥：说白点就是多了层封装，JDK 开发组的人没少看市面上常见的工具类框架啊!

```
//判断字符串是否为空
" ".isBlank();//true
//去除字符串首尾空格
" Java ".strip();// "Java"
//去除字符串首部空格
" Java ".stripLeading();   // "Java "
//去除字符串尾部空格
" Java ".stripTrailing();  // " Java"
//重复字符串多少次
"Java".repeat(3);             // "JavaJavaJava"

//返回由行终止符分隔的字符串集合。
"A\nB\nC".lines().count();    // 3
"A\nB\nC".lines().collect(Collectors.toList());
```

ZGC：可伸缩低延迟垃圾收集器

ZGC 即 Z Garbage Collector，是一个可伸缩的、低延迟的垃圾收集器。

ZGC 主要为了满足如下目标进行设计：

- GC 停顿时间不超过 10ms
- 即能处理几百 MB 的小堆，也能处理几个 TB 的大堆
- 应用吞吐能力不会下降超过 15%（与 G1 回收算法相比）
- 方便在此基础上引入新的 GC 特性和利用 colord
- 针以及 Load barriers 优化奠定基础
- 当前只支持 Linux/x64 位平台

ZGC 目前 **处在实验阶段**，只支持 Linux/x64 平台

标准 HTTP Client 升级

Java 11 对 Java 9 中引入并在 Java 10 中进行了更新的 Http Client API 进行了标准化，在前两个版本中进行孵化的同时，Http Client 几乎被完全重写，并且现在完全支持异步非阻塞。

并且，Java11 中，Http Client 的包名由 `jdk.incubator.http` 改为`java.net.http`，该 API 通过 `CompleteableFuture` 提供非阻塞请求和响应语义。

使用起来也很简单，如下：

```
var request = HttpRequest.newBuilder()

    .uri(URI.create("https://javastack.cn"))

    .GET()
```

```
    .build();

var client = HttpClient.newHttpClient();

// 同步

HttpResponse<String> response = client.send(request,
HttpResponse.BodyHandlers.ofString());

System.out.println(response.body());

// 异步

client.sendAsync(request, HttpResponse.BodyHandlers.ofString())

    .thenApply(HttpResponse::body)

    .thenAccept(System.out::println);
```

简化启动单个源代码文件的方法

- 增强了 Java 启动器，使其能够运行单一文件的 Java 源代码。此功能允许使用 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
- 对于 Java 初学者并希望尝试简单程序的人特别有用，并且能和 jshell 一起使用
- 一定能程度上增强了使用 Java 来写脚本程序的能力

用于 Lambda 参数的局部变量语法

- 从 Java 10 开始，便引入了局部变量类型推断这一关键特性。类型推断允许使用关键字 var 作为局部变量的类型而不是实际类型，编译器根据分配给变量的值推断出类型
- Java 10 中对 var 关键字存在几个限制
    - 只能用于局部变量上
    - 声明时必须初始化
    - 不能用作方法参数
    - 不能在 Lambda 表达式中使用
- Java11 开始允许开发者在 Lambda 表达式中使用 var 进行参数声明

其他特性

- 新的垃圾回收器 Epsilon，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
- 低开销的 Heap Profiling：Java 11 中提供一种低开销的 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
- TLS1.3 协议：Java 11 中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

- 飞行记录器：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，这样所有人都能使用该功能了

Java12

增强 Switch

- 传统的 switch 语法存在容易漏写 break 的问题，而且从代码整洁性层面来看，多个 break 本质也是一种重复
- Java12 提供了 swtich 表达式，使用类似 lambda 语法条件匹配成功后的执行块，不需要多写 break
- 作为预览特性加入，需要在`javac`编译和`java`运行时增加参数`--enable-preview`

```
switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> System.out.println(6);
    case TUESDAY                -> System.out.println(7);
    case THURSDAY, SATURDAY     -> System.out.println(8);
    case WEDNESDAY              -> System.out.println(9);
}
```

数字格式化工具类

- `NumberFormat` 新增了对复杂的数字进行格式化的支持

```
NumberFormat fmt = NumberFormat.getCompactNumberInstance(Locale.US,
NumberFormat.Style.SHORT);
String result = fmt.format(1000);
 System.out.println(result); // 输出为 1K，计算工资是多少K更方便了。。。
```

Shenandoah GC

- Redhat 主导开发的 Pauseless GC 实现，主要目标是 99.9% 的暂停小于 10ms，暂停与堆大小无关等
- 和 Java11 开源的 ZGC 相比（需要升级到 JDK11 才能使用），Shenandoah GC 有稳定的 JDK8u 版本，在 Java8 占据主要市场份额的今天有更大的可落地性

G1 收集器提升

- **Java12 为默认的垃圾收集器 G1 带来了两项更新:**
  - 可中止的混合收集集合：JEP344 的实现，为了达到用户提供的停顿时间目标，JEP 344 通过把要被回收的区域集（混合收集集合）拆分为强制和可选部分，使 G1 垃圾回收器能中止垃圾回收过程。 G1 可以中止可选部分的回收以达到停顿时间目标
  - 及时返回未使用的已分配内存：JEP346 的实现，增强 G1 GC，以便在空闲时自动将 Java 堆内存返回给操作系统

Java13

引入 yield 关键字到 Switch 中

- `Switch` 表达式中就多了一个关键字用于跳出 `Switch` 块的关键字 `yield`，主要用于返回一个值
- `yield`和 `return` 的区别在于：`return` 会直接跳出当前循环或者方法，而 `yield` 只会跳出当前 `Switch` 块，同时在使用 `yield` 时，需要有 `default` 条件

```
 private static String descLanguage(String name) {
        return switch (name) {
            case "Java": yield "object-oriented, platform independent and
secured";
            case "Ruby": yield "a programmer's best friend";
            default: yield name +" is a good language";
        };
 }
```

文本块

- 解决 Java 定义多行字符串时只能通过换行转义或者换行连接符来变通支持的问题，引入**三重双引号**来定义多行文本
- 两个`"""`中间的任何内容都会被解释为字符串的一部分，包括换行符

```
String json ="{\n" +
             "   \"name\":\"mkyong\",\n" +
             "   \"age\":38\n" +
             "}\n";   // 未支持文本块之前
```

```
 String json = """
               {
                   "name":"mkyong",
                   "age":38
               }
               """;
```

增强 ZGC 释放未使用内存

- 在 Java 11 中是实验性的引入的 ZGC 在实际的使用中存在未能主动将未使用的内存释放给操作系统的问题
- ZGC 堆由一组称为 ZPages 的堆区域组成。在 GC 周期中清空 ZPages 区域时，它们将被释放并返回到页面缓存 **ZPageCache** 中，此缓存中的 ZPages 按最近最少使用（LRU）的顺序，并按照大小进行组织
- 在 Java 13 中，ZGC 将向操作系统返回被标识为长时间未使用的页面，这样它们将可以被其他进程重用

SocketAPI 重构

- Java 13 为 Socket API 带来了新的底层实现方法，并且在 Java 13 中是默认使用新的 Socket 实现，使其易于发现并在排除问题同时增加可维护性

动态应用程序类-数据共享

- Java 13 中对 Java 10 中引入的 应用程序类数据共享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

Java14

record 关键字

- 简化数据类的定义方式，使用 record 代替 class 定义的类，只需要声明属性，就可以在获得属性的访问方法，以及 toString，hashCode,equals 方法
- 类似于使用 Class 定义类，同时使用了 lomobok 插件，并打上了 `@Getter,@ToString,@EqualsAndHashCode`注解
- 作为预览特性引入

```
/**
 * 这个类具有两个特征
 * 1. 所有成员属性都是final
 * 2. 全部方法由构造方法，和两个成员属性访问器组成（共三个）
 * 那么这种类就很适合使用record来声明
 */
final class Rectangle implements Shape {
    final double length;
    final double width;

    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    double length() { return length; }
    double width() { return width; }
}
/**
 * 1. 使用record声明的类会自动拥有上面类中的三个方法
 * 2. 在这基础上还附赠了equals(), hashCode()方法以及toString()方法
 * 3. toString方法中包括所有成员属性的字符串表示形式及其名称
 */
record Rectangle(float length, float width) { }
```

空指针异常精准提示

- 通过 JVM 参数中添加`-XX:+ShowCodeDetailsInExceptionMessages`，可以在空指针异常中获取更为详细的调用信息，更快的定位和解决问题

```
a.b.c.i = 99; // 假设这段代码会发生空指针
```

```
Exception in thread "main" java.lang.NullPointerException:
        Cannot read field 'c' because 'a.b' is null.
    at Prog.main(Prog.java:5) // 增加参数后提示的异常中很明确的告知了哪里为空导致
```

switch 的增强终于转正

- JDK12 引入的 switch（预览特性）在 JDK14 变为正式版本，不需要增加参数来启用，直接在 JDK14 中就能使用
- 主要是用`->`来替代以前的`:+break`；另外就是提供了 yield 来在 block 中返回值

Before Java 14

```
switch (day) {
    case MONDAY:
    case FRIDAY:
    case SUNDAY:
        System.out.println(6);
        break;
    case TUESDAY:
        System.out.println(7);
        break;
    case THURSDAY:
    case SATURDAY:
        System.out.println(8);
        break;
    case WEDNESDAY:
        System.out.println(9);
        break;
}
```

Java 14 enhancements

```
switch (day) {
    case MONDAY, FRIDAY, SUNDAY -> System.out.println(6);
    case TUESDAY                -> System.out.println(7);
    case THURSDAY, SATURDAY     -> System.out.println(8);
    case WEDNESDAY              -> System.out.println(9);
}
```

instanceof 增强

- instanceof 主要在**类型强转前探测对象的具体类型**，然后执行具体的强转

- 新版的 instanceof 可以在判断的是否属于具体的类型同时完成转换

```
Object obj = "我是字符串";
if(obj instanceof String str){
    System.out.println(str);
}
```

其他特性

- 从 Java11 引入的 ZGC 作为继 G1 过后的下一代 GC 算法，从支持 Linux 平台到 Java14 开始支持 MacOS 和 Window（个人感觉是终于可以在日常开发工具中先体验下 ZGC 的效果了，虽然其实 G1 也够用）
- 移除了 CMS 垃圾收集器（功成而退）
- 新增了 jpackage 工具，标配将应用打成 jar 包外，还支持不同平台的特性包，比如 linux 下的`deb`和`rpm`，window 平台下的`msi`和`exe`

总结

关于预览特性

- 先贴一段 oracle 官网原文：`This is a preview feature, which is a feature whose design, specification, and implementation are complete, but is not permanent, which means that the feature may exist in a different form or not at all in future JDK releases. To compile and run code that contains preview features, you must specify additional command-line options.`
- 这是一个预览功能，该功能的设计，规格和实现是完整的，但不是永久性的，这意味着该功能可能以其他形式存在或在将来的 JDK 版本中根本不存在。 要编译和运行包含预览功能的代码，必须指定其他命令行选项。
- 就以`switch`的增强为例子，从 Java12 中推出，到 Java13 中将继续增强，直到 Java14 才正式转正进入 JDK 可以放心使用，不用考虑后续 JDK 版本对其的改动或修改
- 一方面可以看出 JDK 作为标准平台在增加新特性的严谨态度，另一方面个人认为是对于预览特性应该采取审慎使用的态度。特性的设计和实现容易，但是其实际价值依然需要在使用中去验证

JVM 虚拟机优化

- 每次 Java 版本的发布都伴随着对 JVM 虚拟机的优化，包括对现有垃圾回收算法的改进，引入新的垃圾回收算法，移除老旧的不再适用于今天的垃圾回收算法等
- 整体优化的方向是**高效，低时延的垃圾回收表现**
- 对于日常的应用开发者可能比较关注新的语法特性，但是从一个公司角度来说，在考虑是否升级 Java 平台时更加考虑的是**JVM 运行时的提升**

参考信息

- IBM Developer Java9 https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-9/
- Guide to Java10 https://www.baeldung.com/java-10-overview
- Java 10 新特性介绍https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-10/index.html

- IBM Devloper Java11 https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-11/index.html
- Java 11 – Features and Comparison： https://www.geeksforgeeks.org/java-11-features-and-comparison/
- Oracle Java12 ReleaseNote https://www.oracle.com/technetwork/java/javase/12all-relnotes-5211423.html#NewFeature
- Oracle Java13 ReleaseNote https://www.oracle.com/technetwork/java/javase/13all-relnotes-5461743.html#NewFeature
- New Java13 Features https://www.baeldung.com/java-13-new-features
- Java13 新特性概述 https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-13/index.html
- Oracle Java14 record https://docs.oracle.com/en/java/javase/14/language/records.html
- java14-features https://www.techgeeknext.com/java/java14-features