你还在使用 JDK 7？JDK 13 都已经来了！五大新特性你最喜欢哪个 ？

就在昨天(2019年09月17日)，JDK 13 已经处于 General Availability 状态，已经正式可用了。General Availability（简称 GA）是一种正式版本的命名，也就是官方开始推荐广泛使用了，我们熟悉的 MySQL 就用 GA 来命令其正式版本。

JDK 13

JDK 13 is the open-source reference implementation of version 13 of the Java SE Platform as specified by by JSR 388 in the Java Community Process.

JDK 13 reached General Availability on 17 September 2019. 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).

Features

350: Dynamic CDS Archives
351: ZGC: Uncommit Unused Memory
353: Reimplement the Legacy Socket API
354: Switch Expressions (Preview)
355: Text Blocks (Preview)

Schedule

2019/06/13	Rampdown Phase One (fork from main line)
2019/07/18	Rampdown Phase Two
2019/08/08	Initial Release Candidate
2019/08/22	Final Release Candidate
2019/09/17	General Availability

Last update: 2019/9/17 17:20 UTC

如果想及时了解Spark、Hadoop或者HBase相关的文章，欢迎关注微信公众号：iteblog_hadoop

从上图我们可以看到 JDK 13 带来了五个大的 Features：

- 350:Dynamic CDS Archives
- 351:ZGC: Uncommit Unused Memory
- 353:Reimplement the Legacy Socket API
- 354:Switch Expressions (Preview)
- 355:Text Blocks (Preview)

本文将带领大家来看看这五大特性。

350: Dynamic CDS Archives

这个特性是扩展 JEP310：Application Class-Data Sharing（简称AppCDS，参见 http://openjdk.java.net/jeps/310）的，以允许在 Java 应用程序执行结束时动态归档类，归档类扩展了默认的基础层 CDS （Class-Data Sharing）存档，此特性允许应用的类也可以被放置在共享的归档类（archived classes）文件中。
JEP 350 的主要目标有两个：

- 提高 AppCDS 的可用性，不需要用户进行试运行来为每个应用程序创建类列表。
- 由 -Xshare:dump 选项启用的静态归档（Static archiving）应该继续工作，这包括用于内置类加载器和用户定义类加载器的类。

用户可以指定动态存档名称的文件名作为 -XX:ArchiveClassesAtExit 选项的参数。例如，下面的命令创建hello.jsa:

```
% bin/java -XX:ArchiveClassesAtExit=hello.jsa -cp hello.jar Hello
```

要使用此动态存档运行相同的应用程序:

```
% bin/java -XX:SharedArchiveFile=hello.jsa -cp hello.jar Hello
```

351: ZGC: Uncommit Unused Memory

ZGC 是在 JDK 11 中引入的一个全新的垃圾收集器，它由 Oracle 开发，承诺在数 TB 的堆上具有非常低的暂停时间。但到目前为止，它还没有像 G1 垃圾收集器那样将未使用的堆内存返回到操作系统。这个JEP解决了这个问题，默认情况下启用了这个功能。由于网上有一大堆 ZGC 的文章，所以本文不准备详细介绍 ZGC了。

353: Reimplement the Legacy Socket API

目前 JDK 的 java.net.Socket 和 java.net.ServerSocket 的实现非常古老，这个 JEP为它们引入了一个全新的实现。Java 13 中的默认使用这个实现，但是旧的实现还没有删除，如果还需要，可以通过设置系统属性 jdk.net.usePlainSocketImpl 来使用它们。需要注意的是，并没有为 java.net.DatagramSocket 引入新的实现。

如果使用了新的 java.net.Socket 和 java.net.ServerSocket，在类中初始化 Socket 和 ServerSocket 将会显示出下面的调试信息：

```
java -XX:+TraceClassLoading JEP353  | grep Socket
[0.033s][info   ][class,load] java.net.Socket source: jrt:/java.base
[0.035s][info   ][class,load] java.net.SocketOptions source: jrt:/java.base
[0.035s][info   ][class,load] java.net.SocketImpl source: jrt:/java.base
[0.039s][info   ][class,load] java.net.SocketImpl$$Lambda$1/0x0000000800b50840 source: java.net.SocketImpl
[0.042s][info   ][class,load] sun.net.PlatformSocketImpl source: jrt:/java.base
[0.042s][info   ][class,load] sun.nio.ch.NioSocketImpl source: jrt:/java.base
[0.043s][info   ][class,load] sun.nio.ch.SocketDispatcher source: jrt:/java.base
[0.044s][info   ][class,load] java.net.DelegatingSocketImpl source: jrt:/java.base
[0.044s][info   ][class,load] java.net.SocksSocketImpl source: jrt:/java.base
[0.044s][info   ][class,load] java.net.ServerSocket source: jrt:/java.base
[0.045s][info   ][class,load] jdk.internal.access.JavaNetSocketAccess source: jrt:/java.base
[0.045s][info   ][class,load] java.net.ServerSocket$1 source: jrt:/java.base
```

上面调试信息输出的 sun.nio.ch.NioSocketImpl 就是新的实现。如果我们需要使用旧的 Socket 实现，可以使用 jdk.net.usePlainSocketImpl 系统属性，这时候调试信息将输出如下信息：

```
$ java -Djdk.net.usePlainSocketImpl -XX:+TraceClassLoading JEP353  | grep Socket
[0.037s][info   ][class,load] java.net.Socket source: jrt:/java.base
[0.039s][info   ][class,load] java.net.SocketOptions source: jrt:/java.base
[0.039s][info   ][class,load] java.net.SocketImpl source: jrt:/java.base
[0.043s][info   ][class,load] java.net.SocketImpl$$Lambda$1/0x0000000800b50840 source: java.net.SocketImpl
[0.046s][info   ][class,load] sun.net.PlatformSocketImpl source: jrt:/java.base
[0.047s][info   ][class,load] java.net.AbstractPlainSocketImpl source: jrt:/java.base
[0.047s][info   ][class,load] java.net.PlainSocketImpl source: jrt:/java.base
[0.047s][info   ][class,load] java.net.AbstractPlainSocketImpl$1 source: jrt:/java.base
[0.047s][info   ][class,load] sun.net.ext.ExtendedSocketOptions source: jrt:/java.base
[0.047s][info   ][class,load] jdk.net.ExtendedSocketOptions source: jrt:/jdk.net
[0.047s][info   ][class,load] java.net.SocketOption source: jrt:/java.base
[0.047s][info   ][class,load] jdk.net.ExtendedSocketOptions$ExtSocketOption source: jrt:/jdk.net
[0.047s][info   ][class,load] jdk.net.SocketFlow source: jrt:/jdk.net
[0.047s][info   ][class,load] jdk.net.ExtendedSocketOptions$PlatformSocketOptions source: jrt:/jdk.net
[0.047s][info   ][class,load] jdk.net.ExtendedSocketOptions$PlatformSocketOptions$1 source: jrt:/jdk.net
[0.048s][info   ][class,load] jdk.net.LinuxSocketOptions source: jrt:/jdk.net
[0.048s][info   ][class,load] jdk.net.LinuxSocketOptions$$Lambda$2/0x0000000800b51040 source: jdk.net.LinuxSocketOptions
```

```
[0.049s][info   ][class,load] jdk.net.ExtendedSocketOptions$1 source: jrt:/jdk.net
[0.049s][info   ][class,load] java.net.StandardSocketOptions source: jrt:/java.base
[0.049s][info   ][class,load] java.net.StandardSocketOptions$StdSocketOption source: jrt:/java.b
ase
[0.051s][info   ][class,load] sun.net.ext.ExtendedSocketOptions$$Lambda$3/0x0000000800b51
440 source: sun.net.ext.ExtendedSocketOptions
[0.057s][info   ][class,load] java.net.DelegatingSocketImpl source: jrt:/java.base
[0.057s][info   ][class,load] java.net.SocksSocketImpl source: jrt:/java.base
[0.058s][info   ][class,load] java.net.ServerSocket source: jrt:/java.base
[0.058s][info   ][class,load] jdk.internal.access.JavaNetSocketAccess source: jrt:/java.base
[0.058s][info   ][class,load] java.net.ServerSocket$1 source: jrt:/java.base
```

从上面信息可以看出，已经是用了旧的 java.net.PlainSocketImpl 实现。

354: Switch Expressions (Preview)

JDK 12 中引入了 Switch 表达式，此时还属于预览特性。JEP 354 修改了这个特性，它引入了 yield 语句，用于从块返回值，而不是使用
break。这意味着，switch表达式需要返回值的时候应该使用 yield,
而需要不返回值应该使用break。

```
System.out.println(switch (args[1]) {
 case "1" -> 1;
 case "2" -> 2;
 default -> {
  int len = args[1].length();
  yield len;
 }
});
```

或者

```
System.out.println(switch (args[1]) {
 case "1": yield 1;
 case "2": yield 2;
 default: {
  int len = args[1].length();
  yield len;
 }
});
```

注意，这个特性在早期版本是不能使用的。

355: Text Blocks (Preview)

JDK 12 中引入了原始字符串文字特性（http://openjdk.java.net/jeps/326），但在发布之前却被放弃了。JEP 355 在引入多行字符串文字(一个文本块)的意义上是类似的。

与 Python 类似，可以定义多行文本块(而不是使用单行连接)，如下所示：

```
String lines = """
        hello iteblog
         hello word
        """;
```

上面例子的 lines 变量里面内容第一个字符是 h，最后一个是字符 d，但是字符 d 后面还跟着一个换行符，所以上面 lines 的定义就相当于 String lines="hello iteblog₩nhello word₩n"，如果 lines 字符串最后你不想要 ₩n，那么你需要这么写：

```
String lines = """
        hello iteblog
         hello word""";
```

这时候上面的定义就相当于 String lines="hello iteblog₩nhello word"，大家也可以看出，hello iteblog 和 hello word 字符串前后的空格全部被删除了。

最后，附上 Oracle Java SE Support Roadmap（https://www.oracle.com/technetwork/java/java-se-support-roadmap.html）可以看出 JDK 13 已经从 2019年09月开始可用了。

Oracle Java SE Support Roadmap*†

Release	GA Date	Premier Support Until	Extended Support Until	Sustaining Support
6	December 2006	December 2015	December 2018	Indefinite
7	July 2011	July 2019	July 2022*****	Indefinite
8 □ **	March 2014	March 2022	March 2025	Indefinite

9 (non □ LTS)	September 2017	March 2018	Not Available	Indefinite
10 (non □ LTS)	March 2018	September 2018	Not Available	Indefinite
11 (LTS)	September 2018	September 2023	September 2026	Indefinite
12 (non □ LTS)	March 2019	September 2019	Not Available	Indefinite
13 (non □ LTS)	September 2019***	March 2020	Not Available	Indefinite

本博客文章除特别声明，全部都是原创！
原创文章版权归过往记忆大数据（过往记忆）所有，未经许可不得转载。
本文链接: 【】（）