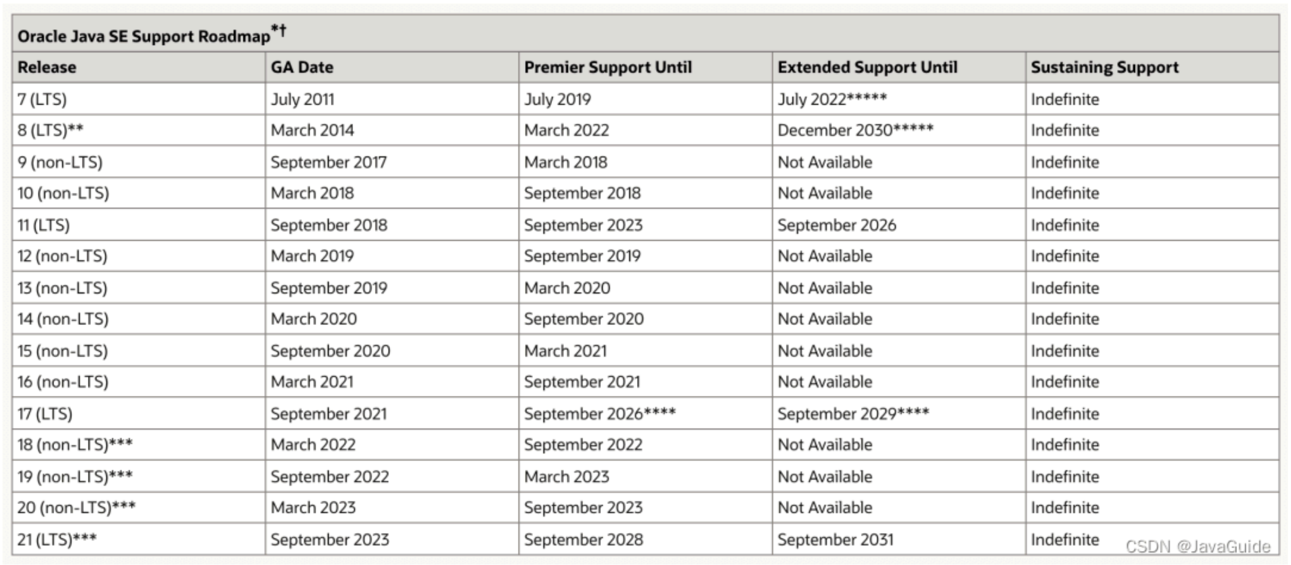
Java 17 New Features Overview (Important)

Guide Java New Java Features About 1864 words About 6 minutes

Java 17 was officially released on September 14, 2021 and is a long-term support (LTS) version.

The following figure is the timeline of Oracle JDK support given by Oracle.

17 can be supported until September 2029 at most.



Oracle Java SE Support Roadmap*†				
Release	**GA Date**	**Premier Support Until**	**Extended Support Until**	**Sustaining Support**
7 (LTS)	July 2011	July 2019	July 2022*****	Indefinite
8 (LTS)**	March 2014	March 2022	December 2030*****	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
14 (non-LTS)	March 2020	September 2020	Not Available	Indefinite
15 (non-LTS)	September 2020	March 2021	Not Available	Indefinite
16 (non-LTS)	March 2021	September 2021	Not Available	Indefinite
17 (LTS)	September 2021	September 2026****	September 2029****	Indefinite
18 (non-LTS)***	March 2022	September 2022	Not Available	Indefinite
19 (non-LTS)***	September 2022	March 2023	Not Available	Indefinite
20 (non-LTS)***	March 2023	September 2023	Not Available	Indefinite
21 (LTS)***	September 2023	September 2028	September 2031	Indefinite

Java 17 will be the most significant long-term support (LTS) release since Java 8, representing eight years of hard work by the Java community. Spring 6.x and Spring Boot 3.x support Java 17 at the very least.

This update brings a total of 14 new features:

- JEP 306: Restore Always-Strict Floating-Point Semantics
- JEP 356: Enhanced Pseudo-Random Number Generators
- JEP 382: New macOS Rendering Pipeline
- JEP 391: macOS/AArch64 Port (support for macOS AArch64)
- JEP 398: Deprecate the Applet API for Removal
- JEP 403: Strongly Encapsulate JDK Internals
- JEP 406: Pattern Matching for switch (Preview)
- JEP 407: Remove RMI Activation
- JEP 409: Sealed Classes (Regularization)
- JEP 410: Remove the Experimental AOT and JIT Compiler

- JEP 411: Deprecate the Security Manager for Removal
- JEP 412: Foreign Function & Memory API (Incubating)
- JEP 414: Vector API (Second Incubation)
- JEP 415:Context-Specific Deserialization Filters

Here I will only introduce in detail the new features 356, 398, 413, 406, 407, 409, 410, 411, 412, and 414 that I think are more important.

Related reading: OpenJDK Java 17 documentation .

JEP 356: Enhanced Pseudo-Random Number Generator

Before JDK 17, we could use `Random` , , `ThreadLocalRandom` and `SplittableRandom` to generate random numbers. However, these three classes have their own flaws and lack support for common pseudo-random algorithms.

Java 17 adds new interface types and implementations for pseudorandom number generators (PRNGs, also known as deterministic random bit generators), making it easier for developers to use various PRNG algorithms interchangeably in their applications.

> PRNGs are used to generate sequences of numbers that are close to being completely random. Generally, PRNGs rely on an initial value, also known as a seed, to generate the corresponding pseudo-random number sequence. As long as the seed is fixed, the random numbers generated by the PRNG are completely deterministic, and therefore the random number sequences they generate are not truly random.

Example usage:

```java
RandomGeneratorFactory<RandomGenerator> l128X256MixRandom = RandomGeneratorFactory.of("L128X256MixRandom");
// 使用时间戳作为随机数种子
RandomGenerator randomGenerator = l128X256MixRandom.create(System.currentTimeMillis());
// 生成随机数
randomGenerator.nextInt(10);
```

JEP 398: Deprecate the Applet API for removal

The Applet API is used to write Java applets that run in web browsers. It was eliminated many years ago and there is no reason to use it anymore.

The Applet API was marked deprecated in Java 9 (JEP 289), but was not intended for removal.

JEP 406: Type Matching in Switch (Preview)

Just `instanceof` like , `switch` the type matching automatic conversion function was also added.

`instanceof` Code example:

```java
// Old code
if (o instanceof String) {
    String s = (String)o;
    ... use s ...
}

// New code
if (o instanceof String s) {
    ... use s ...
}
```

`switch` Code example:

```java
// Old code
static String formatter(Object o) {
    String formatted = "unknown";
    if (o instanceof Integer i) {
        formatted = String.format("int %d", i);
    } else if (o instanceof Long l) {
        formatted = String.format("long %d", l);
    } else if (o instanceof Double d) {
        formatted = String.format("double %f", d);
    } else if (o instanceof String s) {
        formatted = String.format("String %s", s);
    }
    return formatted;
}

// New code
static String formatterPatternSwitch(Object o) {
    return switch (o) {
        case Integer i -> String.format("int %d", i);
        case Long l    -> String.format("long %d", l);
        case Double d  -> String.format("double %f", d);
        case String s  -> String.format("String %s", s);
        default        -> o.toString();
    };
}
```

The judgment of `null` value has also been optimized.

```java
// Old code
static void testFooBar(String s) {
    if (s == null) {
        System.out.println("oops!");
        return;
    }
    switch (s) {
        case "Foo", "Bar" -> System.out.println("Great");
        default           -> System.out.println("Ok");
    }
}

// New code
static void testFooBar(String s) {
    switch (s) {
        case null         -> System.out.println("Oops");
        case "Foo", "Bar" -> System.out.println("Great");
        default           -> System.out.println("Ok");
    }
}
```

JEP 407: Remove Remote Method Invocation

Activation Mechanism

Removes the Remote Method Invocation (RMI) activation mechanism, while retaining the rest of RMI. The RMI activation mechanism is deprecated and no longer used.

JEP 409: Sealed Classes (Regularization)

Sealed classes were previewed by JEP 360 and integrated into Java 15. In JDK 16, sealed classes were improved (stricter reference checking and inheritance relationships of sealed classes) and previewed again by JEP 397 .

I covered sealed classes in detail in the Java 14 & 15 New Features Overview, so I won't go into further detail here .

JEP 410: Remove experimental AOT and JIT compilers

Java 9's JEP 295 introduced an experimental ahead-of-time (AOT) compiler that compiles Java classes into native code before starting the virtual machine.

Java 17 will remove the experimental ahead-of-time (AOT) and just-in-time (JIT) compilers, as they have been rarely used since their introduction and the effort required to maintain them is significant. The experimental Java-level JVM Compiler Interface (JVMCI) will remain so that developers can continue to use externally built versions of the compiler for JIT compilation.

JEP 411: Deprecate Security Manager for Removal

Deprecates the security manager for removal in a future release.

The security manager dates back to Java 1.0, and for many years it has not been the primary method for protecting client-side Java code, and is rarely used to protect server-side code. To move Java forward, Java 17 deprecates the security manager for removal along with the legacy Applet API (JEP 398).

JEP 412: Foreign Function and Memory API (Incubating)

This API enables Java programs to interoperate with code and data outside the Java runtime. By efficiently calling external functions (that is, code outside the JVM) and safely accessing external memory (that is, memory not managed by the JVM), this API enables Java programs to call native libraries and process native data without the risks and brittleness of JNI.

The foreign function and memory APIs had their first incubation phase in Java 17, as proposed by JEP 412. They were incubated in JEP 419 for the second time and integrated into Java 18, and previewed in JEP 424 for Java 19.

In the Java 19 New Features Overview , I introduced the external function and memory API in detail, so I will not give additional introductions here.

JEP 414: Vector API (Second Incubation)

The Vector API was originally proposed by JEP 338 and integrated into Java 16 as an incubating API . The second round of incubation was proposed by JEP 414 and integrated into Java 17, the third round was proposed by JEP 417 and integrated into Java 18, and the fourth round was proposed by JEP 426 and integrated into Java 19.

This incubator API provides an initial iteration of an API for expressing vector computations that reliably compile at runtime to the optimal vector hardware instructions for supported CPU architectures, resulting in superior performance over equivalent scalar computations, leveraging Single Instruction Multiple Data (SIMD) technology, a type of instruction available on most modern CPUs. While HotSpot supports automatic vectorization, the set of scalar operations that can be converted is limited and susceptible to code changes. This API will enable developers to easily write portable, high-performance vector algorithms in Java.

I have already introduced the vector API in detail in the Java 18 new features overview , so I will not give an additional introduction here.

Recently Updated2023/10/27 05:44
Contributors: guide , Guide , Mr.Hope , HunterChen , paigeman