Java 9 New Features Overview

Guide Java New Java Features About 2419 words About 8 minutes

Java 9 was released on September 21, 2017. As a new version released three and a half years after Java 8, Java 9 brought many significant changes, the most important of which was the introduction of the Java platform module system, as well as other changes such as collections and `Stream` streams.

You can download the JDK version you need from Archived OpenJDK General-Availability Releases ! The official new feature description document address is: https://openjdk.java.net/projects/jdk/ .

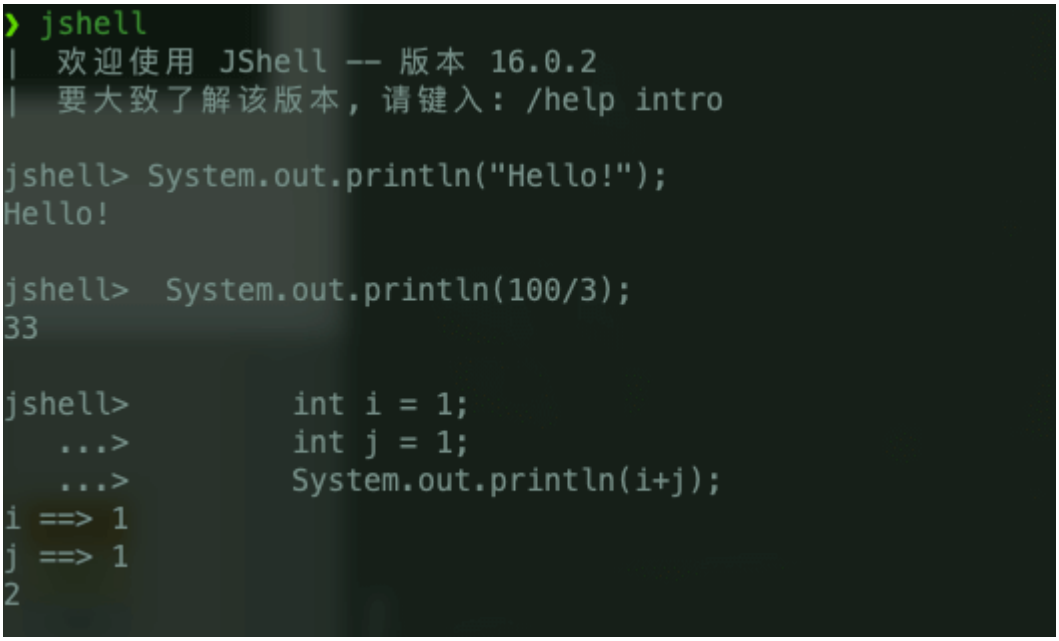
Overview (selected part) :

- JEP 222: Java Command-Line Tools
- JEP 261: Modular Systems
- JEP 248: G1 becomes the default garbage collector
- JEP 193: Variable Handles
- JEP 254: String Storage Structure Optimization

JShell

JShell is a new utility added to Java 9. It provides a real-time command line interactive tool for Java similar to Python.

In JShell, you can directly enter expressions and view their execution results.



What benefits does JShell bring us?

1. The threshold for outputting the first line of the Java version of "Hello World!" is lowered, which can increase the learning enthusiasm of novices.
2. It is more efficient than IDE when dealing with simple logic and verifying simple problems (it is not intended to replace IDE. For the verification of complex logic, IDE is more suitable and the two complement each other).
3. ...

What is the difference between JShell code and ordinary compilable code?

1. Once the statement is entered, JShell can immediately return the execution result without the need for an editor, compiler, or interpreter.
2. JShell supports repeated declaration of variables, and the later declaration will overwrite the earlier declaration.
3. JShell supports independent expressions such as ordinary addition operations `1 + 1`.
4. ...

Modular system

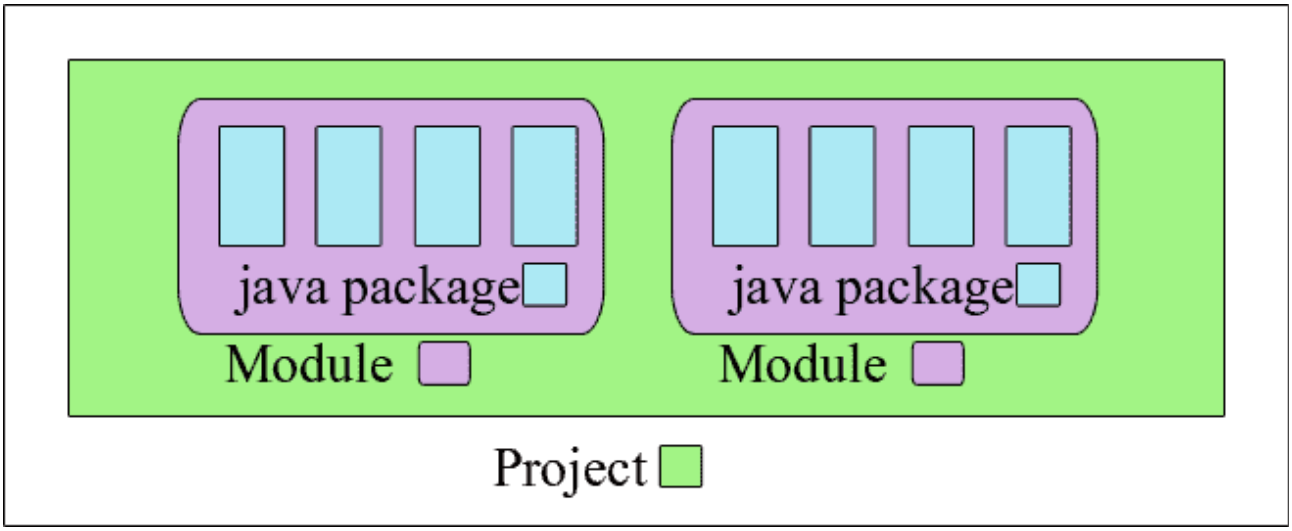
The module system is part of the Jigsaw Project , which introduces modular development practices into the Java platform, making our code more reusable!

What is a module system? The official definition is:

> A uniquely named, reusable group of related packages, as well as resources (such as images and XML files) and a module descriptor.

In simple terms, you can think of a module as a set of uniquely named, reusable packages, resources, and module description files (`module-info.java`).

Any jar file `module-info.java` can be upgraded to a module by adding a module description file ( ).



With the introduction of the module system, the JDK was reorganized into 94 modules. Java applications can use the newly added **jlink tool** (Jlink is a new command-line tool released with Java 9. It allows developers to create their own lightweight, customized JRE for module-based Java applications) to create custom runtime images that only contain the required JDK modules. This significantly reduces the size of the Java runtime environment.

We can use `exports` keywords to precisely control which classes can be opened to the public and which classes can only be used internally.

```java
module my.module {
    //exports 公开指定包的所有公共成员
    exports com.my.package.name;
}

module my.module {
     //exports…to 限制访问的成员范围
    export com.my.package.name to com.specific.package;
}
```

For a deeper understanding of Java 9 modularity, you can refer to the following articles:

- "Project Jigsaw: Module System Quick-Start Guide"
- Java 9 Modules: part 1
- Java 9 Demystified (Part 2. Module System)

G1 becomes the default garbage collector

In Java 8, the default garbage collector was Parallel Scavenge (new generation) + Parallel Old (old generation). In Java 9, the CMS garbage collector was deprecated and **the G1 (Garbage-First Garbage Collector)** became the default garbage collector.

G1 was introduced in Java 7 and became the default garbage collector after two versions of excellent performance.

Quickly create immutable collections

Added factory methods such as `List.of()`, `Set.of()`, `Map.of()` and `Map.ofEntries()` to create immutable collections (a bit like Guava):

```java
List.of("Java", "C++");
Set.of("Java", "C++");
Map.of("Java", 1, "C++", 2);
```

The collection created using `of()` is an immutable collection and cannot be added, deleted, replaced, or sorted. Otherwise, `java.lang.UnsupportedOperationException` an exception will be reported.

String storage structure optimization

Java 8 and earlier versions `String` have always used `char[]` storage. After Java 9, `String` the implementation of uses `byte[]` arrays to store strings, saving space.

```java
public final class String implements
java.io.Serializable,Comparable<String>, CharSequence {
    // @Stable 注解表示变量最多被修改一次，称为“稳定的”。
    @Stable
    private final byte[] value;
}
```

Interface private methods

Java 9 allows private methods in interfaces. This makes the use of interfaces more flexible and a bit like a simplified version of an abstract class.

```java
public interface MyInterface {
    private void methodPrivate(){
    }
}
```

try-with-resources enhancements

Before Java 9, we could only `try-with-resources` declare variables in a block:

```java
try (Scanner scanner = new Scanner(new File("testRead.txt"));
    PrintWriter writer = new PrintWriter(new
File("testWrite.txt"))) {
    // omitted
}
```

Starting from Java 9, `try-with-resources` you can use effectively-final variables in the statement.

```java
final Scanner scanner = new Scanner(new File("testRead.txt"));
PrintWriter writer = new PrintWriter(new File("testWrite.txt"))
try (scanner;writer) {
    // omitted
}
```

What is an effectively-final variable? Simply put, it is `final` a variable that is not modified but whose value is never changed after initialization.

As the above code demonstrates, even though `writer` the variable is not explicitly declared as final `final` , it will not change after the first assignment, so it is an effect
final variable.

Stream & Optional Enhancements

`Stream` New methods `ofNullable()` , `dropWhile()` , `takeWhile()` and `iterate()` overloaded methods of the method have been added.

The `` method in Java 9 `ofNullable()` allows us to create a singleton `Stream` , which can contain a non-null element, or create an empty one `Stream` . In Java 8, it is not possible to create an empty one `Stream` .

```java
Stream<String> stringStream = Stream.ofNullable("Java");
System.out.println(stringStream.count());// 1
Stream<String> nullStream = Stream.ofNullable(null);
System.out.println(nullStream.count());//0
```

`takeWhile()` The method can `Stream` sequentially obtain elements that meet the conditions from , and stop obtaining until the conditions are no longer met.

```java
List<Integer> integerList = List.of(11, 33, 66, 8, 9, 13);
integerList.stream().takeWhile(x -> x <
50).forEach(System.out::println);// 11 33
```

`dropWhile()` The effect of the method is `takeWhile()` the opposite.

```java
List<Integer> integerList2 = List.of(11, 33, 66, 8, 9, 13);
integerList2.stream().dropWhile(x -> x <
50).forEach(System.out::println);// 66 8 9 13
```

`iterate()` The new overloaded method provides a `Predicate` parameter (condition) to determine when to end the iteration

```java
public static<T> Stream<T> iterate(final T seed, final
UnaryOperator<T> f) {
}
// 新增加的重载方法
public static<T> Stream<T> iterate(T seed, Predicate<? super T>
hasNext, UnaryOperator<T> next) {

}
```

The usage comparison of the two is as follows. The new `iterate()` overloaded method is more flexible.

```java
// 使用原始 iterate() 方法输出数字 1~10
Stream.iterate(1, i -> i +
1).limit(10).forEach(System.out::println);
// 使用新的 iterate() 重载方法输出数字 1~10
Stream.iterate(1, i -> i <= 10, i -> i +
1).forEach(System.out::println);
```

`Optional` New methods such as `ifPresentOrElse()` , `or()` and are added to the class `stream()`

`ifPresentOrElse()` The method accepts two parameters `Consumer` and `Runnable` . If `Optional` is not empty `Consumer` , the parameter is called. If it is empty, the parameter is called `Runnable` .

```java
public void ifPresentOrElse(Consumer<? super T> action, Runnable
emptyAction)

Optional<Object> objectOptional = Optional.empty();
objectOptional.ifPresentOrElse(System.out::println, () ->
System.out.println("Empty!!!"));// Empty!!!
```

`or()` The method accepts a `Supplier` parameter and returns the value specified by the parameter if `Optional` it is empty . `Supplier` `Optional`

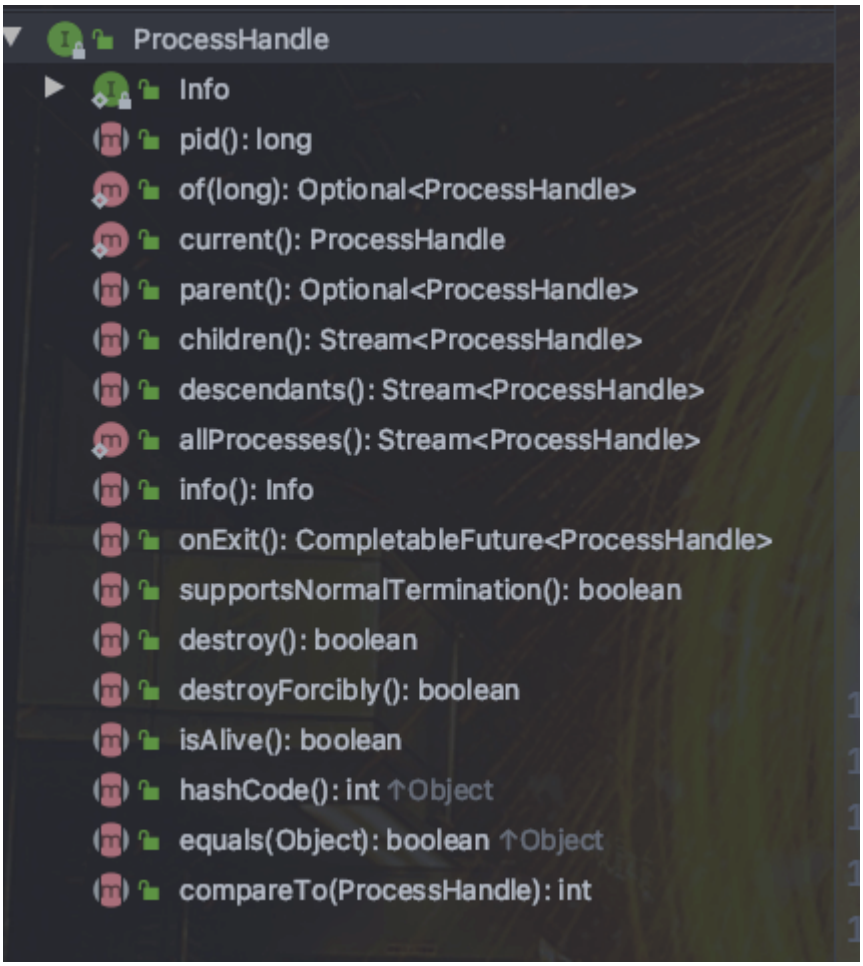
```java
public Optional<T> or(Supplier<? extends Optional<? extends T>>
supplier)

Optional<Object> objectOptional = Optional.empty();
objectOptional.or(() ->
Optional.of("java")).ifPresent(System.out::println);//java
```

Process API

Java 9 adds `java.lang.ProcessHandle` interfaces to manage native processes, which is particularly suitable for managing long-running processes.

```java
// 获取当前正在运行的 JVM 的进程
ProcessHandle currentProcess = ProcessHandle.current();
// 输出进程的 id
System.out.println(currentProcess.pid());
// 输出进程的信息
System.out.println(currentProcess.info());
```

`ProcessHandle` Interface Overview:



Reactive Streams

`java.util.concurrent.Flow` The core interface of the Reactive Streams specification was added in Java 9 with the class.

`Flow` It includes 4 core interfaces, including `Flow.Publisher` , `Flow.Subscriber` , `Flow.Subscription` and `Flow.Processor` . Java 9 also provides an implementatio `SubmissionPublisher` of `Flow.Publisher` .

For a more detailed explanation of Java 9 Reactive Streams, I recommend you read the article Java 9 Unveiled (17. Reactive Streams) by Lin Bento .

variable handle

A variable handle is a reference to a variable or a group of variables, including static fields, non-static fields, array elements, and components of off-heap data structures.

The meaning of variable handle is similar to the existing method handle `MethodHandle` , which is represented by a Java class . Static factory methods in `java.lang.invoke.VarHandle` the class can be used to create objects. `java.lang.invoke.MethodHandles.Lookup` `VarHandle`

`VarHandle` The emergence of replaces some operations in `java.util.concurrent.atomic` and `sun.misc.Unsafe` and provides a series of standard memory barrier operations for more fine-grained control of memory ordering. It outperforms existing APIs in terms of security, usability, and performance.

other

- **Improvements to the Platform Logging API** : Java 9 allows you to configure the same logging implementation for both the JDK and your applications. A new `System.LoggerFinder` logger implementation has been added to manage the logger implementations used by the JDK. The JVM has only one system-wide instance of logger at runtime `LoggerFinder` . You can add your own `System.LoggerFinder` implementation to allow the JDK and applications to use other logging frameworks, such as SLF4J.
- **`CompletableFuture` Class enhancement** : Several new methods (`completeAsync` , `orTimeout` etc.) were added.
- **Enhancements to the Nashorn engine** : Nashorn is a JavaScript engine introduced in Java 8. Java 9 has made some enhancements to Nashorn and implemented some new features of ES6 (which have been deprecated in Java 11).
- **New features for I/O streams** : New methods have been added to read and copy `InputStream` the data contained in .
- **Improve application security** : Java 9 adds four new SHA-3 hash algorithms: SHA3-224, SHA3-256, SHA3-384, and SHA3-512.
- **Improved method handle** : Method handle was introduced in Java 7. Java 9 `java.lang.invoke.MethodHandles` added more static methods to the class to create different types of method handles.

- ...

refer to

- Java version history: https://en.wikipedia.org/wiki/Java_version_history
- Release Notes for JDK 9 and JDK 9 Update Releases: https://www.oracle.com/java/technologies/javase/9-all-relnotes.html
- "In-depth Analysis of Java New Features" - Geek Time - JShell: How to quickly verify simple problems?
- New Features in Java 9: https://www.baeldung.com/new-java-9
- Java – Try with Resources: https://www.baeldung.com/java-try-with-resources

Recently Updated2023/12/30 16:14
Contributors: guide , Guide , Mr.Hope , paigeman , zouzanyan