Java 10

Java 10 Features

Java 10 introduced several significant features and enhancements aimed at improving developer productivity and the performance of Java applications

1. Local-Variable Type Inference ("var")

What is Local-Variable Type Inference?

Java 10 introduced the var keyword to allow local variable type inference. This means that the Java compiler can infer the type of a local variable based on the initializer, making the code more concise and readable.

Explanation

- var list = new ArrayList<String>(); uses var to infer the type ArrayList<String>.
- var map = new HashMap<String, Integer>(); uses var to infer the type HashMap<String, Integer>.
- var is only used for local variables with initializers, not for method parameters or class fields.

2. Immutable Collections

Java 10 enhances the creation of immutable collections with new methods.

List.copyOf(), Set.copyOf(), and Map.copyOf()

These methods create unmodifiable copies of existing collections.

Explanation

- List.copyOf(), Set.copyOf(), and Map.copyOf() create unmodifiable views of the original collections, ensuring they cannot be modified.

 # Collectors.toUnmodifiableList(), toUnmodifiableSet(), and toUnmodifiableMap()

These collectors are used in conjunction with streams to collect elements into unmodifiable collections.

Explanation

- Collectors.toUnmodifiableList(), Collectors.toUnmodifiableSet(), and Collectors.toUnmodifiableMap() are used to collect stream elements into unmodifiable collections.

3. Optional.orElseThrow()

What is Optional.orElseThrow()?

Java 10 adds a new method orElseThrow() to the Optional class. This method returns the contained value if present or throws a NoSuchElementException if not.

Explanation

- orElseThrow() is a more concise way to throw an exception if no value is present, compared to the previous method get() which throws an exception without context.

4. Time-Based Release Versioning

What is Time-Based Release Versioning?

Java 10 introduces a new time-based versioning scheme. Java releases now follow a pattern of feature releases every six months and long-term support (LTS) versions every three years.

Example

- Java 10 is referred to as 10.0.1, where 10 is the feature release number, 0 is the update release number, and 1 is the patch release number.
- Time-based release versioning ensures predictable release cycles, allowing developers to plan and adapt to new features more efficiently.

5. Parallel Full GC for G1

What is Parallel Full GC for G1?

Java 10 improves the G1 garbage collector by enabling parallel processing for full garbage collections, reducing pause times and improving performance for applications with large heaps.

Example

To enable parallel full GC for G1, use the JVM option:

```
java -XX:+UseG1GC -XX:+UnlockExperimentalVMOptions -XX:+UseParallelGCForG1
```

The parallel full GC for G1 enhances the performance of applications with large heaps by distributing the GC workload across multiple threads. Use any one of the gc or it will create a conflict of multiple gcs are selected, (-XX:+UnlockExperimentalVMOptions -XX:+UseParallelGCForG1) use this line mostly.

6. Application Class-Data Sharing

Application Class-Data Sharing (AppCDS) extends the existing Class-Data Sharing (CDS) feature to allow application classes to be stored in a shared archive, reducing startup time and memory usage.

Application Class-Data Sharing Example

Step 1: Create a Shared Archive

```
java -Xshare:dump -XX:SharedArchiveFile=appcds.jsa -XX:ArchiveClassesAtExit=appcds.jsa -cp MyApp.jar
```

Step 2: Use the Shared Archive

```
java -Xshare:on -XX:SharedArchiveFile=appcds.jsa -cp MyApp.jar MyApp
```

Explanation

- AppCDS allows developers to create a shared archive of application classes, reducing startup time and memory footprint by reusing shared data across JVM instances.

7. Experimental Java-Based JIT Compiler

What is the Experimental Java-Based JIT Compiler?

Java 10 introduces Graal, an experimental Java-based Just-In-Time (JIT) compiler, which aims to improve the performance and scalability of Java applications.

Example: Enable Graal JIT Compiler

To enable Graal as the JIT compiler, use the JVM option:

```
java -XX:+UnlockExperimentalVMOptions -XX:+UseJVMCICompiler -jar MyApp.jar
```

The Graal JIT compiler provides a more flexible and maintainable approach to JIT compilation, with the potential for better optimization and performance.

8. Other Changes in Java 10

Java 10 includes several other changes that improve various aspects of the Java platform.

Heap Allocation on Alternative Memory Devices

Java 10 supports heap allocation on alternative memory devices, allowing applications to leverage different memory types for improved performance and resource utilization.

Additional Unicode Language-Tag Extensions

Java 10 adds support for additional Unicode language-tag extensions, enhancing internationalization capabilities.

Garbage Collector Interface

Java 10 introduces a new garbage collector interface, making it easier to implement and integrate custom garbage collectors.

Root Certificates

Java 10 includes a default set of root certificates in the JDK, simplifying the configuration of secure applications.

Thread-Local Handshakes

Java 10 introduces thread-local handshakes, allowing individual threads to execute a callback without performing a global VM safepoint.

Remove the Native-Header Generation Tool

Java 10 removes the native-header generation tool (javah), consolidating header generation into the javac compiler.

Consolidate the JDK Forest into a Single Repository

Java 10 consolidates the JDK source code into a single repository, simplifying the build process and improving maintainability.

Complete List of All Changes in Java 10

For a complete list of all changes in Java 10, refer to the official **Java 10 release notes**.