Java 12 & 13 New Features Overview

Guide Java New Java Features About 2182 words About 7 minutes

Java 12

String Enhancements

Java 12 adds two string processing methods, as shown below.

`indent()` Method can achieve string indentation.

```java
String text = "Java";
// 缩进 4 格
text = text.indent(4);
System.out.println(text);
text = text.indent(-10);
System.out.println(text);
```

Output:

```plain
    Java
Java
```

`transform()` Method can be used to transform the specified string.

```java
String result = "foo".transform(input -> input + " bar");
System.out.println(result); // foo bar
```

Files enhancements (file comparison)

Java 12 adds the following methods to compare two files:

```
public static long mismatch(Path path, Path path2) throws
IOException
```

`mismatch()` The compare method compares two files and returns the position of the first mismatched character, or -1L if the files are identical.

Code example (when the two files have the same content):

```java
Path filePath1 = Files.createTempFile("file1", ".txt");
Path filePath2 = Files.createTempFile("file2", ".txt");
Files.writeString(filePath1, "Java 12 Article");
Files.writeString(filePath2, "Java 12 Article");

long mismatch = Files.mismatch(filePath1, filePath2);
assertEquals(-1, mismatch);
```

Code example (when the contents of the two files are different):

```java
Path filePath3 = Files.createTempFile("file3", ".txt");
Path filePath4 = Files.createTempFile("file4", ".txt");
Files.writeString(filePath3, "Java 12 Article");
Files.writeString(filePath4, "Java 12 Tutorial");

long mismatch = Files.mismatch(filePath3, filePath4);
assertEquals(8, mismatch);
```

Number formatting tools

`NumberFormat` Added support for formatting complex numbers

```java
NumberFormat fmt = NumberFormat.getCompactNumberInstance(Locale.US,
NumberFormat.Style.SHORT);
String result = fmt.format(1000);
System.out.println(result);
```

Output:

```plain
1K
```

Shenandoah GC

Red Hat led the development of the Pauseless GC implementation, with the main goal of 99.9% of the pauses being less than 10ms, and the pauses being independent of the heap size.

Compared with the open source ZGC of Java 11 (which requires upgrading to JDK 11 to use), Shenandoah GC has a stable JDK 8u version, which has greater feasibility today when Java 8 occupies a major market share.

G1 Collector Optimization

Java 12 brings two updates to the default garbage collector G1:

- **Abortable mixed collection sets** : Implementation of JEP 344. In order to achieve the pause time target provided by the user, JEP 344 enables the G1 garbage collector to abort the garbage collection process by splitting the set of regions to be collected (mixed collection sets) into mandatory and optional parts. G1 can abort the collection of the optional part to achieve the pause time target.
- **Timely return of unused allocated memory** : Implementation of JEP346, enhancing the G1 GC to automatically return Java heap memory to the operating system when idle

Preview new features

Added as a preview feature, requires adding parameters during `javac` compilation and `java` runtime `--enable-preview` .

Enhanced Switch

The traditional `switch` syntax has the problem of being easily missed `break` , and from the perspective of code neatness, multiple breaks are essentially a form of repetition.

Java 12 enhances `switch` expressions, using lambda-like syntax to execute blocks after successful condition matching, eliminating the need to write break.

```java
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 pattern matching

`instanceof` Mainly detect the specific type of the object before type coercion.

In previous versions, we needed to explicitly convert objects.

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

The new version `instanceof` can complete the conversion while determining whether it belongs to a specific type.

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

Java 13

Enhance ZGC (release unused memory)

The ZGC experimentally introduced in Java 11 has the problem of failing to actively release unused memory to the operating system in actual use.

The ZGC heap consists of a set of heap regions called ZPages. When ZPages are cleared during a GC cycle, they are released and returned to the page cache **ZPageCache** , wh ZPages are sorted in least recently used (LRU) order and organized by size.

In Java 13, ZGC will return pages that have been marked as unused for a long time to the operating system so that they can be reused by other processes.

SocketAPI refactoring

The Java Socket API finally received a major update!

Java 13 rewrites the underlying Socket API `NioSocketImpl` to be `PlainSocketImpl` a direct replacement for Socket. It uses `java.util.concurrent` locks from the Socket package instead of synchronization methods. If you want to use the old implementation, please use Socket `-Djdk.net.usePlainSocketImpl=true` .

Also, in Java 13, the new Socket implementation is used by default.

```java
public final class NioSocketImpl extends SocketImpl implements PlatformSocketImpl {
}
```

FileSystems

`FileSystems` The following three new methods have been added to the class to make it easier to use file system providers that treat file contents as a file system:

- `newFileSystem(Path)`
- `newFileSystem(Path, Map<String, ?>)`
- `newFileSystem(Path, Map<String, ?>, ClassLoader)`

Dynamic CDS Archive

Java 13 further simplifies, improves, and expands the Application Class Data Sharing (AppCDS) introduced in Java 10. This **allows dynamic class archiving at the end of a Java application's execution** . Specifically, the classes that can be archived include all loaded application classes and classes in referenced class libraries that do not belong to the default base CDS.

This improves the usability of Application Class Data Sharing (AppCDS) by eliminating the need for users to perform a trial run to create a class list for each application.

```bash
java -XX:ArchiveClassesAtExit=my_app_cds.jsa -cp my_app.jar
java -XX:SharedArchiveFile=my_app_cds.jsa -cp my_app.jar
```

Preview new features

Text Block

To solve the problem that Java can only support multi-line strings through line break escape or line break concatenation, **triple double quotes** are introduced to define multi-line text.

Java 13 supports that `"""` anything between two </marks> will be interpreted as part of the string, including newline characters.

HTML writing method before text blocks are supported:

```java
String json ="{\n" +
              "   \"name\":\"mkyong\",\n" +
              "   \"age\":38\n" +
              "}\n";
```

Support HTML writing after text blocks:

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

SQL writing method before text blocks are supported:

```sql
String query = "SELECT `EMP_ID`, `LAST_NAME` FROM `EMPLOYEE_TB`\n" +
               "WHERE `CITY` = 'INDIANAPOLIS'\n" +
               "ORDER BY `EMP_ID`, `LAST_NAME`;\n";
```

Supports SQL writing after text blocks:

```sql
String query = """
               SELECT `EMP_ID`, `LAST_NAME` FROM `EMPLOYEE_TB`
               WHERE `CITY` = 'INDIANAPOLIS'
               ORDER BY `EMP_ID`, `LAST_NAME`;
               """;
```

In addition, `String` the class adds 3 new methods to operate text blocks:

- `formatted(Object... args)` : It is similar `String` to `format()` the method of . It was added to support formatting of text blocks.
- `stripIndent()` : Used to remove spaces at the beginning and end of each line in a text block.
- `translateEscapes()` : Escape sequences such as "\\t" are converted to "*t*"

Because text blocks are a preview feature and may be removed in a future release, these new methods are marked as deprecated.

```java
@Deprecated(forRemoval=true, since="13")
public String stripIndent() {
}
@Deprecated(forRemoval=true, since="13")
public String formatted(Object... args) {

}
@Deprecated(forRemoval=true, since="13")
public String translateEscapes() {
}
```

Enhance Switch (introduce the yield keyword into Switch)

`Switch` There is an additional keyword in the expression for jumping out of `Switch` the block `yield` , which is mainly used to return a value.

`yield` The difference between and `return` is that: `return` will directly jump out of the current loop or method, while `yield` will only jump out of the current `Switch` block. At the same time, when using , the condition `yield` is required. `default`

```java
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";
        };
}
```

Replenish

About Preview Features

First, let me post a paragraph from the original text of Oracle's official website: `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.`

This is a preview feature whose design, specification, and implementation are complete but not permanent, which means that the feature might 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.

Take `switch` the enhancement as an example. It was launched in Java 12 and will continue to be enhanced in Java 13. It will not be officially incorporated into JDK until Java 14. You can use it with confidence without worrying about changes or modifications to it in subsequent JDK versions.

On the one hand, it can be seen that JDK, as a standard platform, is rigorous in adding new features. On the other hand, I personally think that preview features should be used with caution. The design and implementation of features are easy, but their actual value still needs to be verified in use.

JVM virtual machine optimization

Each release of Java is accompanied by optimizations to the JVM virtual machine, including improvements to existing garbage collection algorithms, introduction of new garbage collection algorithms, and removal of old garbage collection algorithms that are no longer applicable today.

The overall optimization direction is **efficient and low-latency garbage collection performance**

For daily application developers, they may pay more attention to new syntax features, but from a company's perspective, when considering whether to upgrade the Java platform, they are more concerned about **the improvement of the JVM runtime.**

refer to

- JDK Project Overview: https://openjdk.java.net/projects/jdk/
- Oracle Java12 ReleaseNote: https://www.oracle.com/java/technologies/javase/12all-relnotes.htm
- What is new in Java 12: https://mkyong.com/java/what-is-new-in-java-12/
- 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
- Overview of the new features of Java 13 https://www.ibm.com/developerworks/cn/java/the-new-features-of-Java-13/index.html

JavaGuide官方公众号
（微信搜索JavaGuide）

1、公众号后台回复“PDF”获取原创PDF面试手册

2、公众号后台回复“学习路线”获取Java学习路线最新版

3、公众号后台回复“开源”获取优质Java开源项目合集

4、公众号后台回复“八股文”获取Java面试真题+面经

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