

Core Java №хз уже какой

Spliterator<T>

Подобно Iterator и ListIterator, Spliterator — это итератор Java, который используется для последовательного перебора элементов. Некоторые важные моменты о Java Spliterator:

- Java Spliterator — это интерфейс API Java Collection.
- Spliterator представлен в выпуске Java 8 в пакете java.util.
- Он поддерживает функцию параллельного программирования.
- Мы можем использовать его как для классов Collection API, так и для классов Stream API.
- Он предоставляет характеристики объектов коллекции или API.
- Мы НЕ можем использовать этот итератор для классов, реализующих Map.
- Он использует метод tryAdvance() для индивидуальной итерации элементов в нескольких потоках для поддержки параллельной обработки.
- Он использует метод forEachRemaining() для последовательного перебора элементов в одном потоке.
- Он использует метод trySplit() для разделения себя на суб-разделители для поддержки параллельной обработки.
- Spliterator поддерживает как последовательную, так и параллельную обработку данных.

Spliterator<T>

Iterator vs Spliterator

Iterator

Introduced in Java 1.2.

It is an Iterator for whole Collection API.

It is an Universal Iterator.

It does NOT support Parallel Programming.

Spliterator

Introduced in Java 1.8.

It is an Iterator for both Collection and Stream API, except Map implemented classes.

It is NOT an Universal Iterator.

It supports Parallel Programming.

`Spliterator<T>`

Spliterator = Splitting + Iterator

Spliterator<T> - методы

- int characteristics(): Возвращает набор характеристик этого Spliterator и его элементов.
- long estimateSize(): возвращает оценку количества элементов, которые могут встретиться при обходе forEachRemaining(), или возвращает Long.MAX_VALUE, если оно бесконечно, неизвестно или слишком дорого для вычисления.
- default void forEachRemaining(Consumer action): выполняет заданное действие для каждого оставшегося элемента последовательно в текущем потоке до тех пор, пока все элементы не будут обработаны или действие не вызовет исключение.
- default Comparator getComparator(): если источник этого Spliterator сортируется с помощью Comparator, возвращает этот Comparator.
- default long getExactSizeIfKnown(): удобный метод, который возвращает AssessmentSize(), если этот разделитель имеет РАЗМЕР, иначе -1.
- default boolean hasCharacteristics(int characteristics): возвращает true, если характеристики этого Spliterator() содержат все заданные характеристики. Могут быть ORDERED, DISTINCT, SORTED, SIZED, NONNULL, IMMUTABLE, CONCURRENT, and SUBSIZED.
- boolean tryAdvance(Consumer action): если оставшийся элемент существует, выполняет над ним заданное действие, возвращая true; иначе возвращает ложь.
- Spliterator trySplit(): если этот Spliterator можно разделить, возвращает элементы, покрывающие Spliterator, которые при возврате из этого метода не будут покрыты этим Spliterator.

Ввод - вывод

Потоковый ввод-вывод (java.io)

- ✓ `InputStream` – читать байты
- ✓ `OutputStream` – писать байты
- ✓ `Reader` – читать символы
- ✓ `Writer` – писать символы

InputStream

- ✓ `int read() → -1 или 0..255`
- ✓ `int read(byte[] b) → количество прочитанных байт`
- ✓ `int read(byte[] b, int offset, int length) → количество прочитанных байт`

InputStream

- ✓ `int read() → -1 или 0..255`
- ✓ `int read(byte[] b) → количество прочитанных байт`
- ✓ `int read(byte[] b, int offset, int length) → количество прочитанных байт`
- ✓ `long skip(long n) → количество пропущенных байт`
- ✓ `int available() → количество байт, доступных без блокировки`
- ✓ `void mark(int readlimit)`
- ✓ `void reset()`
- ✓ `boolean markSupported()`

InputStream сегодня

- ✓ `byte[] readAllBytes()` → всё содержимое (Java 9+)
- ✓ `byte[] readNBytes(int len)` → не больше len байт (Java 11+)
- ✓ `int readNBytes(byte[] b, int offset, int length)` → количество прочитанных байт (до ошибки или конца потока, Java 11+)
- ✓ `void skipNBytes(long n)` → пропускает n байт (EOFException, если меньше)
- ✓ `long transferTo(OutputStream out)` → перекачивает в выходной поток (Java 9+)
- ✓ `static InputStream nullInputStream()` → пустой поток (Java 11+)

InputStream реализации

- ✓ FileInputStream – файл
- ✓ ByteArrayInputStream – массив байт
- ✓ BufferedInputStream – буферизированная обёртка
- ✓ DataInputStream – структурированные двоичные данные
- ✓ PipedInputStream – плохой, негодный
- ✓ ZipFile.getInputStream(ZipEntry) – файл из зипа (без распаковки!)
- ✓ Process.getInputStream() – стандартный вывод процесса
- ✓ URL.openStream() – содержимое по URL (http/https/file/ftp/etc.)
- ✓ ...

URL (HTTP клиент для бедных)

```
InputStream is = new URL("http://www.google.com").openStream();
System.out.println(new String(is.readAllBytes(), StandardCharsets.UTF_8));
```

N.B.: складывать URL в HashSet может быть очень плохой идеей (лучше URI)

HttpClient (Java 11) - гибко

```
HttpClient client = HttpClient.newHttpClient();
HttpRequest request = HttpRequest.newBuilder()
    .uri(URI.create("http://www.google.com/"))
    .build();
client.sendAsync(request, HttpResponse.BodyHandlers.ofString())
    .thenApply(HttpResponse::body)
    .thenAccept(System.out::println)
    .join();
```

Управление внешними ресурсами

- ✓ Закрыть явно (close())
- ✓ Положиться на Garbage collector (finalize(), PhantomReference, Cleaner)

Закрываем явно

```
InputStream f = new FileInputStream("/etc/passwd");
int b = f.read();
f.close();
```

```
InputStream f = new FileInputStream("/etc/passwd");
try {
    int b = f.read();
}
finally {
    f.close();
}
```

```
InputStream in = new FileInputStream("/etc/passwd");
OutputStream out = new FileOutputStream("/etc/passwd.bak");
try {
    in.transferTo(out);
}
finally {
    out.close();
    in.close();
}
```

```
InputStream in = new FileInputStream("/etc/passwd");
OutputStream out = new FileOutputStream("/etc/passwd.bak");
try {
    in.transferTo(out);
}
finally {
    out.close();
    in.close();
}
```

```
InputStream in = new FileInputStream("/etc/passwd");
try {
    OutputStream out = new FileOutputStream("/etc/passwd.bak");
    try {
        in.transferTo(out);
    } finally {
        out.close();
    }
}
finally {
    in.close();
}
```

finalize() (deprecated for removal)

```
/*
 * Ensures that the <code>close</code> method of this file input stream is
 * called when there are no more references to it.
 *
 * @exception IOException if an I/O error occurs.
 * @see java.io.FileInputStream#close()
 */
protected void finalize() throws IOException {
    if ((fd != null) && (fd != FileDescriptor.in)) {
        /* if fd is shared, the references in FileDescriptor
         * will ensure that finalizer is only called when
         * safe to do so. All references using the fd have
         * become unreachable. We can call close()
        */
        close();
    }
}
```

Почему finalize() - это плохо

- ✓ Создание объекта дороже, надо зарегистрировать его в очереди финализации
- ✓ Сборка объекта выполняется в два прохода GC
- ✓ finalize() выполняются в отдельном треде линейно
- ✓ Если объект уже закрыт, finalize() выполняется всё равно
- ✓ Объект можно оживить в finalize()
- ✓ Головная боль для авторов GC

```
try (InputStream in = new FileInputStream("/etc/passwd");
     OutputStream out = new FileOutputStream("/etc/passwd.bak")) {
    in.transferTo(out);
}
```

OutputStream

- ✓ `void write(int b)` → пишет 8 младших бит
- ✓ `void write(byte[] b)` → пишет весь массив (если не упал, то всё записал)
- ✓ `void write(byte[] b, int offset, int length)` → часть массива
- ✓ `void flush()`
- ✓ `void close()`
- ✓ `static OutputStream nullOuputStream()` → пиши сколько влезет (Java 11)

OutputStream реализации

- ✓ FileOutputStream – файл
- ✓ ByteArrayOutputStream – массив байт
- ✓ BufferedOutputStream – буферизированная обёртка
- ✓ PrintStream – для удобства печати (System.out)
- ✓ DataOutputStream – структурированные двоичные данные
- ✓ PipedOutputStream (тоже плохой)
- ✓ Process.getOutputStream() – стандартный ввод процесса
- ✓ URL.openConnection().getOutputStream() – писать в URL (HTTP POST)
- ✓ ...

ByteArrayOutputStream

- write(int b)
- write(byte[] b, int off, int len)
- write(byte[] b) **throws IOException** ☹
- writeBytes(byte[] b) – Java 11
- toByteArray()
- size()
- toString(Charset)

Reader

- ✓ `int read() → -1 или 0..0xFFFF`
- ✓ `int read(char[] cb) → количество прочитанных символов`
- ✓ `int read(char[] cb, int offset, int length) → количество прочитанных символов`
- ✓ `long skip(long n) → количество пропущенных символов`
- ✓ `boolean ready() → можно ли прочитать хоть что-то без блокировки`
- ✓ `void mark(int readlimit)`
- ✓ `void reset()`
- ✓ `boolean markSupported()`
- ✓ `long transferTo(Writer out) – Java 10`
- ✓ `static Reader nullReader() – Java 11`

Reader – реализации

- ✓ InputStreamReader (InputStream + charset)
- ✓ FileReader (осторожно)
 - ✓ Java 11: FileReader(file, charset)
 - ✓ Java 18: UTF-8 by default
- ✓ StringReader
- ✓ BufferedReader
- ✓ ...

BufferedReader extends Reader

- ✓ `String readLine()`
- ✓ `Stream<String> lines()`

Writer

- ✓ `void write(int b)` → пишет 16 младших бит
- ✓ `void write(char[] cb)` → пишет весь массив (если не упал, то всё записал)
- ✓ `void write(char[] cb, int offset, int length)` → часть массива
- ✓ `void write(String str)` → пишет всю строку (если не упал, то всё записал)
- ✓ `void write(String str, int offset, int length)` → часть строки
- ✓ `void flush()`
- ✓ `void close()`
- ✓ `static Writer nullWriter()` – Java 11

Writer – реализации

- ✓ OutputStreamWriter (OutputStream + charset)
- ✓ FileWriter (осторожно)
 - ✓ Java 11: FileWriter(file, charset)
 - ✓ Java 18: UTF-8 by default
- ✓ StringWriter
- ✓ BufferedWriter
- ✓ ...

java.io.File

- ✓ Представляет и путь к файлу, и сам файл (каталог и т. д.)
- ✓ Может быть абсолютным или относительным
- ✓ Относительный считается относительно текущего каталога Java-процесса (и его крайне трудно поменять)

java.io.File (классика)

-  `new File(parentDirectory + "/" + fileName);`
-  `new File(parentDirectory + File.separator + fileName);`
-  `new File(parentDirectory, fileName);`

java.io.File (классика)

- ✓ `getName()`/`getPath()`/`getParent()`/`getParentFile()`
- ✓ `getAbsoluteFile()`/`getAbsolutePath()`
- ✓ `getCanonicalFile()`/`getCanonicalPath()`
- ✓ `exists()`/`canRead()`/`canWrite()`/`canExecute()`/`length()`
- ✓ `isDirectory()`/`isFile()`/`isHidden()`
- ✓ `createNewFile()`/`mkdir()`/`mkdirs()`
- ✓ `renameTo()`/`delete()`
- ✓ `list([filter])`/`listFiles([filter])`

```
import java.io.File;

public class Listing {
    public static void main(String[] args) {
        var etcDirectory = new File("/etc");
        for (String fileName : etcDirectory.list()) { 
            System.out.println("File: "+fileName);
        }
    }
}
```

Exception in thread "main" java.lang.NullPointerException
at pkg.Listing.main(Listing.java:8)

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;

public class Listing {
    public static void main(String[] args) throws IOException {
        var etcDirectory = Paths.get("/etc");
        try (var stream = Files.List(etcDirectory)) {
            stream.forEach(fileName -> System.out.println("File: "+fileName));
        }
    }
}
```



Exception in thread "main" java.nio.file.NoSuchFileException: \etc

```
public class Listing {  
    public static void main(String[] args) throws IOException {  
        var etcDirectory = Paths.get("/etc");  
        try (var stream = Files.list(etcDirectory)) {  
            stream.forEach(fileName -> System.out.println("File: "+fileName));  
        }  
    }  
}
```

Exception in thread "main" java.nio.file.NoSuchFileException: \etc
Exception in thread "main" java.nio.file.AccessDeniedException: \System Volume Information
Exception in thread "main" java.nio.file.NotDirectoryException: \bootmgr
Exception in thread "main" java.nio.file.FileSystemException: \\server\share\: The network path was not found.

java.nio.file.Path/Paths/Files (модерн)

- ✓ Путь – это только путь + файловая система
- ✓ Возможны виртуальные пути (например, внутри zip-файла)
- ✓ API похоже, но отличается
 - ✓ Было: `new File("hello/world").getAbsoluteFile()`
 - ✓ Стало: `Path.of("hello/world").toAbsolutePath()`

java.nio.file.Path/Paths/Files (модерн)

```
Path p = Paths.get("/etc/passwd");
Path p = Paths.get(parentDirectory, fileName);
Path p = Paths.get("foo", "bar", "baz");
```

Java 11+

```
Path p = Path.of("/etc/passwd");
Path p = Path.of(parentDirectory, fileName);
Path p = Path.of("foo", "bar", "baz");
```

Path (Comparable)

- ✓ `getFileName()`
- ✓ `getParent()`
- ✓ `getRoot()`
- ✓ `resolve(Path/String)`, `resolveSibling(Path/String)`
- ✓ `isAbsolute()/toAbsolutePath()`
- ✓ `startsWith(Path/String)`, `endsWith(Path/String)`
- ✓ `getName(int)`, `getNameCount()`, `iterator()`
- ✓ `toString()/toFile()/toUri()`
- ✓ `register(WatchService, WatchEvent.Kind...)`

```
var path = Paths.get("C:\\Windows\\System32");
for (Path p : path) {
    System.out.println(p);
}
```

Files

- ✓ copy
- ✓ move
- ✓ delete/deleteIfExists
- ✓ createFile/createDirectory/createDirectories
- ✓ createLink/createSymbolicLink
- ✓ createTempFile/createTempDirectory
- ✓ readAllBytes/readAllLines
- ✓ write (byte[], Iterable<String>)
- ✓ lines/list/walk – Stream
- ✓ walkFileTree
- ✓ exists/size/getAttribute/isDirectory/isRegularFile/isSymbolicLink...
- ✓ newBufferedReader/newInputStream/newOutputStream/newByteChannel

Другое I/O

- ✓ ByteBuffer/DirectBuffer/MappedByteBuffer
- ✓ Channel/FileChannel
- ✓ RandomAccessFile
- ✓ Socket/ServerSocket