

Работа с потоками ввода-вывода, NIO и файловыми системами в Java

Роман Гордеев




План занятия

- Введение в работу с потоками ввода-вывода
 - Разница между `InputStream` и `Reader`
 - Интерфейс `AutoCloseable` и конструкция `try-with-resources`
 - Оптимизация работы потоков (буферизация)
 - Работа с файлами: Java NIO (`java.nio.file`)
 - Виртуальные файловые системы (пример: ZIP)
 - Структура проекта (Gradle, JUnit, Log4j 2)
 - Итоги и вопросы
-

Введение в работу с потоками ввода-вывода

- Потоки – основной способ обмена данными с внешним миром в Java.
- Поток (Stream) – это упорядоченная последовательность данных.
- Типы потоков:
 - **Байтовые** (InputStream, OutputStream)
 - **Символьные** (Reader, Writer)



Разница между InputStream и Reader

InputStream

Байтовые данные
(бинарные)

Нет кодировки

Работа с файлами, медиа,
сетью

Reader

Символьные данные
(текстовые)

Использует кодировку

Работа с текстовыми
файлами



Демонстрация примера №1

Исходный код:

StreamExample.java

Юнит-тест:

InputStreamVsReaderTest.java

Диагностические сообщения:

Используем Log4j 2 для логирования операций чтения из потоков.

Интерфейс AutoCloseable и try-with-resources

Проблема: ресурсы (файлы, соединения) требуют закрытия вручную.

Решение:

- Интерфейс AutoCloseable (Java 7+)

```
public interface AutoCloseable { void close() throws Exception; }
```

- Конструкция try-with-resources

```
try (Resource res = new Resource()) { res.doWork(); } // автоматически вызовется res.close()
```

- Подавленные исключения (getSuppressed())



Демонстрация примера №2

Исходный код:

MyResource.java, AutoCloseableExample.java

Юнит-тест:

MyResourceTest.java

Диагностические сообщения:

Заккрытие ресурсов логируется через Log4j 2.

Оптимизация работы потоков (буферизация)

- Частые операции ввода-вывода могут замедлять программу.
- Используем буферизацию для снижения нагрузки:
 - `BufferedInputStream`, `BufferedOutputStream`
 - `BufferedReader`, `BufferedWriter`

Преимущества буферизации:

- Снижение количества обращений к диску
- Чтение/запись данных блоками (эффективность)

Демонстрация примера №3

Исходный код:

IOOptimizationExample.java

Юнит-тест:

BufferingTest.java – измерение времени работы.

Диагностические сообщения:

Замеры производительности записываются в логи (Log4j 2).

Работа с файлами: Java NIO (`java.nio.file`)

Ключевые классы:

- `Path` – удобная замена устаревшему классу `File`
- `Files` – статические методы работы с файлами

Возможности:

- Простое чтение/запись (`readAllLines`, `writeString`)
- Улучшенная обработка ошибок
- Поддержка атрибутов файлов и директорий

Демонстрация примера №4

Исходный код:

NioFileExample.java

Юнит-тест:

NioFilesTest.java

Диагностические сообщения:

Подробное логирование всех операций с файлами (Log4j 2).

Виртуальные файловые системы (пример: ZIP)

- Java поддерживает различные типы файловых систем.
- ZIP-файлы могут быть представлены как виртуальная файловая система.

Работа с ZIP через NIO:

```
try (FileSystem zipFs = FileSystems.newFileSystem(zipUri, props)) {  
    Path fileInZip = zipFs.getPath("/hello.txt");  
    Files.writeString(fileInZip, "Hello, ZIP!");  
}
```

Демонстрация примера №5

Исходный код:

ZipFileSystemExample.java

Юнит-тест:

ZipFileSystemTest.java

Диагностические сообщения:

Отслеживание операций с ZIP-архивом через Log4j 2.

Структура проекта (Gradle, JUnit, Log4j 2)

Структура:

```
project-root/  
├── build.gradle  
├── src/  
│   ├── main/java/...  
│   ├── main/resources/log4j2.xml  
│   └── test/java/...
```

Структура проекта (Gradle, JUnit, Log4j 2)

Файл **build.gradle**:

```
plugins { id 'java' }
repositories { mavenCentral() }
dependencies {
    implementation 'org.apache.logging.log4j:log4j-api:2.17.1'
    implementation 'org.apache.logging.log4j:log4j-core:2.17.1'
    testImplementation 'org.junit.jupiter:junit-jupiter:5.8.2'
}
test { useJUnitPlatform() }
```

Итоги занятия

Вы узнали:

- Разницу между потоками `InputStream` и `Reader`
- Как использовать `AutoCloseable` и `try-with-resources`
- Как оптимизировать потоки с помощью буферизации
- Как использовать современный подход Java NIO для работы с файлами
- Как работать с виртуальными файловыми системами (ZIP)