

Организация ввода-вывода

Пакеты

- java.io
- java.nio
- java.util.jar
- java.util.zip
- отдельные классы пакета java.net

Работа с файлами

- в пакете `java.io` класс `File` - абстрактное представление имени файла или пути каталогу
- в пакете `java.nio` классы `Files` и `Paths` – представляют статические методы для работы с файлами и каталогами

Список файлов в каталоге

```
import java.io.*;
import java.util.*;

public class DirList {
    public static void main(String[ ] args) {
        File path = new File(".");
        String[ ] list;
        if(args.length == 0) list = path.list( );
        else
            list = path.list(new DirFilter(args[0]));
        for(int i = 0; i < list.length; i++)
            System.out.println(list[i]);
    }
}
```

- list() – базовая функциональность
- интерфейс FilenameFilter – предоставляет алгоритм, необходимый для работы list()
FilenameFilter.accept (File, String)

```
class DirFilter implements FilenameFilter {  
    private Pattern ptr;  
    DirFilter(String afn) {  
        ptr = Pattern.compile( afn);  
    }  
    public boolean accept(File dir, String name) {  
        return ptr.matcher(name).matches();  
    }  
}
```

java.nio

Статические методы в классе Files

`createDirectory(path)`

`createDirectories(path)`

`createFile(path)`

`createTempFile(path,prefix,suffix)`

`exists(path)`

`isDirectory(path)`

`isRegularFile(path)`

`copy(fromPath, toPath)`

`move(fromPath, toPath)`

`delete(path)`

`deleteIfExists(path)`

Копирование файлов

java.nio

```
try {  
    Path source = Paths.get(. . .); //путь к файлу - источнику  
    Path target = Paths.get(. . .); // путь к файлу копии  
    Files.copy(source, target,  
                StandardCopyOption.REPLACE_EXISTING);  
} catch(InvalidPathException e) {  
    System.out.println("Ошибка указания пути " + e);  
} catch (IOException e) {  
    System.out.println("Ошибка ввода-вывода " + e);  
}
```

Элементы каталога

Список элементов каталога

```
try (Stream<Path> entries = Files.list(path)) {  
    entries.forEach(System.out::println);  
}
```

Обход дерева каталогов с подсчетом количества элементов

```
try (Stream<Path> entries = Files.walk(path)) {  
    System.out.printf("%d files\n", entries.count());  
}
```


Копирование каталога

```
Path source = path;
Path target = Files.createTempDirectory("./copy");
Files.walk(source).forEach(p -> {
    try {
        Path q = target.resolve(source.relativize(p));
        if (Files.isDirectory(p)) {
            Files.createDirectory(q);
        }
        else {
            Files.copy(p, q);
        }
    } catch (IOException ex) {
        throw new UncheckedIOException(ex);
    }
});
```

Обход дерева каталогов

```
try {  
    Files.walkFileTree(Paths.get(dirname),  
                      new MyFileVisitor());  
} catch (IOException exc) {  
    System.out.println("Ошибка ввода-вывода");  
}
```

- класс `MyFileVisitor` расширяет класс `SimpleFileVisitor`
- должен определять в методе `visitFile()`, что делать для каждого файла
- Может определять действия перед обработкой каталога и после обработки `preVisitDirectory()`, `postVisitDirectory()`

Пример «посетителя»

```
class MyFileVisitor extends SimpleFileVisitor<Path> {  
    public FileVisitResult visitFile( Path path,  
                                     BasicFileAttributes attribs)  
        throws IOException {  
        System.out.println(path);  
        return FileVisitResult.CONTINUE;  
    }  
}
```

Потоки ввода-вывода java.io

- Поток ввода – источник, из которого вводятся байты. Источником может быть файл, сетевое соединение, массив, строка или другой поток
- Поток вывода представляет адресат выводимых байтов
- Базовые абстрактные файлы
 - InputStream – базовый метод read()
 - OutputStream – базовый метод write()

Разновидности входных потоков

InputStream

- `ByteArrayInputStream` поток - массив байтов
- `StringBufferInputStream` поток - строка
- `FileInputStream` поток - файл
- `PipedInputStream` поток - `PipedOutputStream`
- `SequenceInputStream` объединение потоков
- `FilterInputStream` абстрактный класс для реализации надстроек
 - `BufferedInputStream`
 - `DataInputStream`
 - `LineNumberInputStream`
 - `PushBackInputStream`

Типизированные данные

```
InputStream in = Files.newInputStream(path);
```

```
DataInputStream in = new DataInputStream (in);
```

```
double d= in.readDouble();
```

```
String s = in.readUTF();
```

```
int n= in.readInt();
```

Разновидности выходных потоков

OutputStream

- ByteArrayOutputStream
- FileOutputStream
- PipedOutputStream поток - PipedInputStream
- FilterOutputStream
 - DataOutputStream
 - BufferedOutputStream
 - PrintStream

Типизированные данные

```
DataOutputStream out =  
    new DataOutputStream ("data.inf");  
out.writeDouble(3.14159);  
out.writeInt(123);  
out.writeUTF("text string");
```


Потоки чтения-записи

java.io

- Предназначены для посимвольного ввода-вывода
- Базовые абстрактные классы

Reader

Writer

Классы – наследники

Reader	Writer
InputStreamReader	OutputStreamWriter
FileReader	FileWriter
StringReader	StringWriter
CharArrayReader	CharArrayWriter
PipedReader	PipedWriter
FilterReader	FilterWriter
BufferedReader	BufferedWriter
LineNumberReader	PrintWriter
PushBackReader	

Построчный ввод текста

```
BufferedReader in =  
    new BufferedReader (new FileReader (filename));  
// допустимо начиная с JDK5  
//BufferedReader in = new BufferedReader (filename);  
  
String s;  
while ( (s = in.readLine()) != null)  
    System.out.println(s);
```

ТЕКСТОВЫЙ ВЫВОД

```
PrintWriter out = new PrintWriter (  
    new BufferedWriter (  
        new FileWriter( filename)));
```

//сокращенная форма (JDK 5)

```
PrintWriter out = PrintWriter(filename);
```

Методы :

print()

println()

printf()

Файлы с произвольным доступом

- RandomAccessFile (прямой наследник Object)
- конструктор требует указания режима
 - “r”
 - “rw”
- методы read() и write() и для разных типов
- дополнительно
 - seek()
 - skipBytes()
 - getFilePointer()

Стандартный ввод-вывод

- System.in (InputStream)
- System.out (PrintStream)
- System.err (PrintStream)

Текстовый ввод из System.in

```
BufferedReader stdin= new BufferedReader(  
    new InputStreamReader(System.in));
```

```
String s=stdin.readLine();
```

Автоматическое закрытие потока

- Все классы потоков ввода-вывода реализуют интерфейс `Closeable`
- Для освобождение ресурсов используется метод
`void close() throws IOException`
- Начиная с Java 7 введен интерфейс `AutoCloseable`
- `void close() throws Exception`


```
try{
OutputStream stream = openOutputStream();
// что-то делаем со stream ***
    stream.close();
} catch(IOException e) {
// сообщение об ошибке
}
// flush() для других целей!!!
```

```
FileInputStream fin=null;
try {
    fin = new FileInputStream("notes.txt"); // что-то делаем
} catch(IOException ex){
    System.out.println(ex.getMessage());
} finally{
    try{
        if(fin!=null) fin.close();
    }
    catch(IOException ex){
        System.out.println(ex.getMessage());
    }
}
```

try with resource

```
try(FileInputStream fin=new FileInputStream("notes.txt"))
{
    // что-то делаем с потоком
}
catch(IOException ex){
    System.out.println(ex.getMessage());
}
```

try with resource

```
try( FileInputStream fin=new FileInputStream("Hello.txt");  
    FileOutputStream fos = new FileOutputStream ("Hello2.txt" )  
{  
    //.....  
}
```

Сериализация

- Сериализация объекта – это сохранение объекта в поток в виде набора байтов
- Восстановление объекта из набора байтов – это десериализация
- Сериализация и десериализация должны идти по одним правилам
- Сериализация нарушает безопасность объекта
- Механизм легковесного долговременного хранения

Сериализация

- Библиотеки для сериализации (Java JDO)
- Фреймворк Hibernate
- Сериализация необходима для RMI и JavaBean

Интерфейс

- `Serializable`
- не содержит ни одного метода и служит флагом, помечающим класс, для которого возможна сериализация/десериализация
- когда сериализуемый объект содержит ссылки на другие объекты, будет предпринята попытка сериализовать также и их (глубокое копирование – создание дубликата графа объектов)
- Если сериализация невозможна *`NotSerializableException`*

Методы

- Ввод/вывод - потоки `FileInputStream/FileOutputStream`
- поток-обертка `ObjectOutputStream`
 - метод `writeObject()`
- поток-обертка `ObjectInputStream`
 - метод `readObject()`
- В выходной поток выводятся все нестатические поля объекта, независимо от прав доступа к ним
- для восстановления объекта необходимо иметь доступ к реализации класса этого объекта

Долговременное хранение

- Сохранить состояние программы для последующего восстановления
- Как решать проблему множественных ссылок на один объект?
- Когда сохранять?
- Если проводить сериализацию в единый выходной поток, сохранится вся сеть объектов и она будет гарантированно восстановлена без излишних повторений
- Безопасное сохранение – атомарной операцией. Поместить все объекты в контейнер и сохранить одной операцией.

Сжатие данных

- Пакет `java.util.zip.*`
- Форматы GZIP и ZIP для отдельных файлов
- Фильтры
 - `GZIPOutputStream/GZIPInputStream`
 - `ZipOutputStream/ZipInputStream`
- Многофайловые архивы (классы)
 - `ZipEntry`
 - `ZipFile`
 - `CRC32`
 - `Adler32`

Java ARchives (JAR)

- Формат ZIP
- Файл, содержащий набор сжатых файлов и их описание (манифест)
- Пакет классов для работы `java.util.jar.*`
- Инструмент – архиватор `jar`
- Может содержать набор классов с классом, запускающим работу приложения

```
java -jar nameArh.jar
```

Файлы в сети

- Классы URI и URL пакет java.net

```
URL url = new URL (строка_URL);
```

```
//URLConnection hpCon = url.openConnection();
```

```
InputStream in = url.openStream();
```

```
// InputStream in = hpCon.getInputStream();
```

- далее поток in можно обернуть фильтром, например

```
Scanner sin = new Scanner(in);
```

НОВЫЙ ВВОД/ВЫВОД

- Пакет `java.nio.*`

начиная с версии JDK 1.4

- Основные структуры

- каналы (`channels`) и

- буферы (`buffers`)

близкие к средствам операционной системы

Основные положения

- Реализуют концепцию файлов, отображаемых в память
- Канал представляет открытое соединение с устройством ввода-вывода
- Часть файла (возможно, весь файл) отображается в память – буфер, где хранятся данные
- Для применения системы ввода-вывода NIO нужно получить канал для устройства ввода –вывода и буфер для хранения данных
- Все операции реализуются через ввод-вывод данных в буфер

Buffer

- Базовый абстрактный класс
- Набор операций, позволяющий определять и изменять организацию буфера

Наследники

ByteBuffer

IntBuffer

CharBuffer

LongByffer

DoubleBuffer

FloatBuffer

ShortBuffer

MappedByteBuffer

- Методы
 - get()
 - put()
 - allocate()
 - wrap()
 - slice()

Каналы

- Интерфейс Channel – наследник Closeable
- с версии JDK 7 – наследник AutoCloseable (для try with resource)
- канал может быть получен от потокового класса java.io методом getChannel()
- в JDK7 через статический метод класса Files
- Каналы позволяют управлять блокировкой файла через методы lock() и tryLock()