

JAVA

Vstup a výstup

Přehled

- balík java.io
 - základní vstup a výstup
 - "streams"
 - bytes
 - od JDK1.1 Reader a Writer
 - chars (Unicode)
- balík java.nio – od JDK1.4
 - channels, buffers
 - zvýšení výkonu
 - třídy z java.io uvnitř přeimplementovány pomocí java.nio
- java.io.Console
 - od JDK 6
 - přístup k textové konzoli (pokud existuje)
- NIO.2 – od JDK 7
 - zejména balík java.nio.file
 - operace se soubory, procházení stromů,...

Vstup a výstup

Path

Path

- `java.nio.file.Path`
 - interface
 - reprezentuje cestu na filesystému
 - získání cesty
 - `Paths.get(String first, String... more)`
 - statická metoda
 - př.
`Path p = Paths.get("home", "petr", "text.txt");`
 - `Path.of(String first, String... more)`
 - od Java 11
 - to stejné jako `Paths.get()`
 - doporučeno používat
 - `Paths` bude možná deprecated
 - `FileSystems.getDefault().getPath(String first, String... more)`
 - `Path.of()` pracuje s defaultním filesystémem

Path – metody

- porovnávání cest
 - equals(..), startsWith(..), endsWith(..)
- relativizování cesty
 - Path p1 = Paths.get("joe");
 - Path p2 = Paths.get("sally");
 - Path p1_to_p2 = p1.relativize(p2); // -> ../sally
- získávání skutečné cesty ze symlinku
 - toRealPath()
- Path implementuje Iterable<Path>
 - iterování přes komponenty
- normalize()
 - odstranění zbytečných částí cesty
 - d1/../d2/ => d1/d2
- ...

Path – sledování změn

- WatchKey register(WatchService watcher, WatchEvent.Kind<?>... events)

```
WatchService watchService =
FileSystems.getDefault().newWatchService();
WatchKey key = this.path.register(watchService,
ENTRY_CREATE, ENTRY_DELETE);
while (true) {
    for (WatchEvent<?> l : key.pollEvents()) {
        ...
    }
    boolean valid = key.reset();
    if (!valid) {
        ...
    }
}
```

java.nio.file.Files

- pouze statické metody
 - copy(.. src, .. target, CopyOptions... options)
 - CopyOptions
 - REPLACE_EXISTING
 - COPY_ATTRIBUTES
 - NOFOLLOW_LINKS
 - move(.. src, .. target, CopyOptions... options)
 - CopyOptions
 - ATOMIC_MOVE
 - REPLACE_EXISTING
 - delete(), deleteIfExists()
 - byte[] readAllBytes(Path p)
 - List<String> readAllLines(Path path)
 - Path write(Path path, byte[] bytes, OpenOption... options)
 - Path write(Path path, Iterable<? extends CharSequence> lines, Charset cs, OpenOption... options)

CopyOptions, OpenOptions,...

- interface
- používáno v metodách na Files
- implementace
 - StandardCopyOptions
 - enum (ATOMIC_MOVE, COPY_ATTRIBUTES,...)
 - StandardOpenOptions
 - enum (APPEND, READ, WRITE,...)
 - LinkOptions
 - enum (NOFOLLOW_LINKS)

java.nio.file.Files

- (pokračování)
 - Path createLink(Path link, Path existing)
 - Path createSymbolicLink(Path link, Path target, FileAttribute<?>... attrs)
 - createDirectory(Path dir, FileAttribute<?>... attrs)
 - createDirectories(Path dir, FileAttribute<?>... attrs)
 - createFile(Path path, FileAttribute<?>... attrs)
 - createTempFile(String prefix, String suffix, FileAttribute<?>... attrs)
 - createTempFile(Path dir, String prefix, String suffix, FileAttribute<?>... attrs)
 - „testovací“ metody
 - isDirectory()
 - isRegularFile()
 - is....()

java.nio.file.Files

- procházení stromu
 - Path walkFileTree(Path start, FileVisitor<? super Path> visitor)
 - metoda na Files
 - interface FileVisitor<T>
 - FileVisitResult preVisitDirectory(T dir, BasicFileAttributes attrs)
 - FileVisitResult postVisitDirectory(T dir, IOException exc)
 - FileVisitResult visitFile(T file, BasicFileAttributes attrs)
 - FileVisitResult visitFileFailed(T file, IOException exc)

java.nio.file

- příklad – smazání celého stromu

```
Path start = ...
Files.walkFileTree(start, new SimpleFileVisitor<Path>() {
    public FileVisitResult visitFile(Path f,
        BasicFileAttributes attrs) throws IOException {
        Files.delete(file);
        return FileVisitResult.CONTINUE;
    }
    public FileVisitResult postVisitDirectory(Path dir,
        IOException e) throws IOException {
        if (e == null) {
            Files.delete(dir);
            return FileVisitResult.CONTINUE;
        } else {
            throw e;
        }
    }
});
```

java.io.File

- od Java 1.0
 - java.nio.files.Path – od Java 7
 - java.io.File není deprecated
 - používá se na mnoha místech v std knihovně
- také reprezentuje cestu
 - obdobné použití jako Path
 - Path má více možností
- lze mezi sebou převádět
 - File.toPath()
 - Path.toFile()

vždy preferujte Path
(pokud je to možné)

Oddělovače cest / souborů

- atributy na `java.io.File`
 - `static String pathSeparator`
 - `static char pathSeparatorChar`
 - oddělovač cest
 - `static String separator`
 - `static char separatorChar`
 - oddělovač souborů v cestě
- metoda na `java.nio.file.FileSystem`
 - `String getSeparator()`

Streams

Přehled

- od Java 1.0
- **InputStream**
 - `int read()`
 - čte jeden byte ze vstupu (na konci vstupu vrací -1)
 - `int read(byte[] b)`
 - čte několik bytů do pole (vrací počet načtených bytů nebo -1)
- **OutputStream**
 - `void write(int b)`
 - `void write(byte[] a)`
- všechny další třídy pro čtení/zápis jsou odvozeny od **InputStream/OutputStream**
 - používají se potomci
 - **InputStream** a **OutputStream** jsou abstract

Vstupní streamy

- `ByteArrayInputStream`
 - čte z buferu v paměti
- `StringArrayInputStream`
 - konvertuje řetězce na vstupní stream
- `FileInputStream`
 - čte ze souboru
- `PipedInputStream`
 - "čtecí" konec roury
 - předávání dat mezi vlákny
- `SequenceInputStream`
 - spojení více vstupních streamů do jednoho
- mají jen základní metody `read()`
 - čtení po bytech

Výstupní streamy

- `ByteArrayOutputStream`
 - zapisuje do buferu v paměti
- `FileOutputStream`
 - zapisuje do souboru
- `PipedOutputStream`
 - "zapisovací" konec roury
 - předávání dat mezi vlákny
- **není** `StringArrayOutputStream`
 - použije se `ByteArrayOutputStream`
- mají jen základní metody `write()`
 - zápis po bytech

Filtry

- `FilterInputStream`
- `FilterOutputStream`
- **abstraktní třídy**
 - mnoho potomků
- pomocí filtrů se přidává další funkcionality k základním streamům
 - filtr dostane při vytvoření stream jako parametr
 - data se čtou/zapisují přes filtr
- základní streamy se používají téměř vždy přes nějaký filtr
- filtry lze aplikovat přes sebe
 - více filtru nad jedním streamem

Druhy filtrů

- `DataOutputStream`
 - definuje metodu `write` pro všechny primitivní typy
- `DataInputStream`
 - definuje metodu `read` pro všechny primitivní typy
 - čte data ve formátu, v jakém je zapsal
- `DataOutputStream`
 - formát dat nezávislý na platformě
- `BufferedInputStream`
- `BufferedOutputStream`
 - nepřidávají nové čtecí/zapisovací metody
 - vstup/výstup bude bufrovaný
 - normálně není
 - lze zadat kapacitu buferu

Druhy filtrů

- `LineNumberInputStream`
 - informace, za kterého řádku se čte
- `PushbackInputStream`
 - umožňuje "vrátit" data na vstup

Druhy filtrů

- `PrintStream`
 - zapisuje data zobrazitelným způsobem
 - `DataOutputStream` zapisuje data tak, aby šla přečíst pomocí `DataInputStream`
 - definuje metody `print()` a `println()` pro "všechny" typy
 - metoda `printf()`
 - jako `printf` v C
 - metoda `flush()`
 - zapíše buffer do streamu „pod“ `PrintStreamem`
 - `PrintStream` je automaticky bufferován
 - `flush()` se volá automaticky při zápisu konce řádku
 - lze v konstruktoru nastavit `autoflush` po každém zápisu
 - metody nevyhazují `IOException`
 - metoda `checkError()`

Používání

- vrstvení filtrů nad základní vstupem/výstupem

```
DataInputStream di = new DataInputStream(  
    new BufferedInputStream (  
        new FileInputStream("file.txt")));  
int a = di.readInt();  
long b = di.readLong();
```

```
DataOutputStream ds = new DataOutputStream(  
    new BufferedOutputStream (  
        new FileOutputStream("file.txt")));  
ds.writeInt(100);  
ds.writeLong(1234L);
```

na `java.nio.file.Files`
jsou metody pro
„rychlé“ otevření

Reader & Writer

Přehled

- od Java 1.1
- znakově-orientovaný vstup a výstup
 - znak = 2 byty
- streamy stále zůstávají
 - nejsou deprecated
 - některé ano
- `Reader`
 - definuje metodu `read` pro čtení znaků a pole znaků
- `Writer`
 - definuje metodu `write` pro zápis znaků, pole znaků a řetězců
- `Reader` i `Writer` – abstraktní třídy
- `InputStreamReader`, `OutputStreamWriter`
 - vytvoření `Readeru/Writeru` ze streamu

Druhy vstupu/výstupu

- obdobné jako u streamů

InputStream	Reader InputStreamReader
OutputStream	Writer OutputStreamWriter
FileInputStream	FileReader
FileOutputStream	FileWriter
StringBufferInputStream	StringReader
-	StringWriter
ByteArrayInputStream	CharArrayReader
ByteArrayOutputStream	CharArrayWriter
PipedInputStream	PipedReader
PipedOutputStream	PipedWriter

Filtry

- opět obdobné jako u streamů

FilterInputStream	FilterReader
FilterOutputStream	FilterWriter
BufferedInputStream	BufferedReader
BufferedOutputStream	BufferedWriter
PrintStream	PrintWriter
LineNumberInputStream	LineNumberReader
PushbackInputStream	PushbackReader

Zpracování výjimek

Výjimky

- skoro „vše“ v java.io vyhazuje IOException
 - potomek od Exception
 - nutno odchytit/deklarovat

Kopírování souborů

```
InputStream is;  
OutputStream os;  
try {  
    is = new FileInputStream(finNm);  
    os = new FileOutputStream(foutNm);  
    int c;  
    while ((c = is.read()) != -1) {  
        os.write(c);  
    }  
    os.close();  
    is.close();  
} catch (IOException ex) {  
    System.out.println("Exception occurred");  
}
```

- Je tento kód OK?

Kopírování souborů

```
InputStream is;
OutputStream os;
try {
    is = new FileInputStream(finNm);
    os = new FileOutputStream(foutNm);
    int c;
    while ((c = is.read()) != -1) {
        os.write(c);
    }
    os.close();
    is.close();
} catch (IOException ex) {
    System.out.println("Exception occurred");
}
```

- Je tento kód OK?

NENÍ

Výjimky

- streamy a readery/writery implementují `AutoCloseable`
- vždy používat *try with resources*

```
try (InputStream is = new FileInputStream(finNm);
     OutputStream os = new FileOutputStream(foutNm)) {
    int c;
    while ((c = is.read()) != -1) {
        os.write(c);
    }
} catch (IOException ex) {
    System.out.println("Exception occurred");
}
```

RandomAccessFile

Přehled

- čtení a zápis záznamů ze souboru
- pohyb po souboru
- mimo hierarchii tříd streamů
- implementuje interface `DataInput` a `DataOutput`
 - tyto interface implementují `DataInputStream` resp. `DataOutputStream`
 - metody `read` a `write` pro primitivní typy
- otevírá soubor buď na čtení nebo na čtení i zápis
 - parametr konstruktoru
 - "r" nebo "rw"

NIO

Přehled

- "new I/O"
- od JDK1.4
- zvýšení výkonu
 - přiblížení strukturám vstupu/výstupu v OS
- třídy z `java.io` (stream a reader/writer)
přeimplementovány pomocí tříd z `java.nio`
- definuje *channels* a *buffers*
 - s kanálem se komunikuje pouze přes bufer
- `FileInputStream`, `FileOutputStream` a `RandomAccessFile`
 - metoda `FileChannel` `getChannel()`
 - od Java 7 také `FileChannel.open(Path...)`
- `java.nio.channels.Channels`
 - metody pro vytváření `Readerů` a `Writerů` z kanálů

Používání

- `java.nio.ByteBuffer`
 - jediná možnost jak komunikovat s kanálem

```
FileChannel fc =
    new FileOutputStream("data.txt").getChannel();
fc.write(ByteBuffer.wrap("Some text ".getBytes()));
fc.close();
```

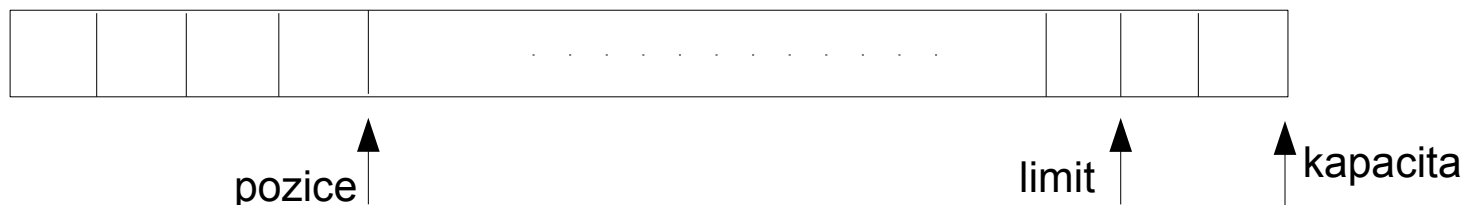
```
fc = new FileInputStream("data.txt").getChannel();
ByteBuffer buff = ByteBuffer.allocate(1024);
fc.read(buff);
buff.flip();
while (buff.hasRemaining())
    System.out.print((char)buff.get());
```

Vytváření buferu

- `ByteBuffer.wrap(byte[] b)`
 - statická metoda
 - z pole bytů udělá bufer
 - bufer je s polem stále svázaný
 - kapacita buferu = `b.length`
- `ByteBuffer.allocate(int capacity)`
 - statická metoda
 - alokuje prázdný bufer s danou kapacitou
- `ByteBuffer.allocateDirect(int capacity)`
 - statická metoda
 - alokovaný bufer je "více" svázan s OS
 - používání buferu by mělo být rychlejší
 - záleží na OS

Buffer

- kapacita
 - kolik prvků bufer obsahuje
 - nelze ji zvětšovat
- limit
 - index prvního prvku, který nebude zapisován nebo čten
 - nikdy není větší než kapacita
- pozice
 - index prvního prvku, který bude zapisován nebo čten při další operaci
 - nikdy není větší než limit



Buffer: metody

- flip()
 - nastaví limit na současnou pozici a
 - pozici nastaví na 0
- clear()
 - nastaví limit na kapacitu a
 - pozici nastaví na 0
- mark()
 - nastaví značku na aktuální pozici
- reset()
 - nastaví pozici na značku
 - neruší značku
- rewind()
 - nastaví pozici na 0 a zruší značku

Kopírování mezi kanály

- metody `transferTo()` a `transferFrom()`

```
public static void main(String[] args) throws
    Exception {

    FileChannel
        in = new FileInputStream(args[0]).getChannel(),
        out = new
    FileOutputStream(args[1]).getChannel();

    in.transferTo(0, in.size(), out);

    // nebo:
    // out.transferFrom(in, 0, in.size());
}
```


Používání buferu

- "pohledy" na bufer (views)
- čtení a zapisování primitivních typů
- metody na ByteBuffer
 - asCharBuffer()
 - asDoubleBuffer()
 - asFloatBuffer()
 - asIntBuffer()
 - asLongBuffer()

```
ByteBuffer bb = ByteBuffer.allocate(1024);  
bb.asIntBuffer().put(1234);  
System.out.println(bb.getInt());
```

Endian

- implicitně `ByteBuffer` používá pro data *velký endian*
- lze změnit na *malý endian*
 - metoda `order(ByteOrder b)`
 - třída `ByteOrder` má dva statické atributy typu `ByteOrder`
 - `BIG_ENDIAN`
 - `LITTLE_ENDIAN`

Soubory mapované do paměti

- přístup k souboru jako k poli v paměti
- metoda na kanálu
 - MappedByteBuffer map()

```
public class LargeMappedFiles {
    static int length = 0x8FFFFFFF; // 128 Mb
    public static void main(String[] args) throws Exception {
        MappedByteBuffer out =
            new RandomAccessFile("test.dat", "rw").getChannel()
                .map(FileChannel.MapMode.READ_WRITE, 0, length);
        for(int i = 0; i < length; i++)
            out.put((byte) 'x');

        for(int i = length/2; i < length/2 + 6; i++)
            System.out.print((char) out.get(i));
    }
}
```

Zamykání souborů

```
FileOutputStream fos = new
FileOutputStream("file.txt");
FileLock fl = fos.getChannel().tryLock();
if (fl != null) {
    System.out.println("Soubor zamknut");
    Thread.sleep(100);
    fl.release();
    System.out.println("Soubor odemknut");
}
fos.close();
```

- přesné chování zámků závisí na OS
- lze zamknout jen část souboru
- metoda lock() – čeká dokud zámeček nedostane
- metoda tryLock() – nečeká

Vstup a výstup

...ještě zpět k Path/Files

Otevírání souborů/adresářů

- metody na Files
 - `BufferedReader newBufferedReader(Path p, Charset cs)`
 - `BufferedWriter newBufferedWriter(Path p, Charset cs, OpenOption... opts)`
 - `InputStream newInputStream(Path p, OpenOption... opts)`
 - `OutputStream newOutputStream(Path p, OpenOption... opts)`
 - `SeekableByteChannel newByteChannel(Path p, OpenOption... opts)`
 - `DirectoryStream<Path> newDirectoryStream(Path dir)`
 - ...

Console

Console

- přístup ke (znakové) konzoli
 - ne vždy může fungovat
- `System.console()`
 - získání konzole
- `Console printf(String format, Object... args)`
 - obdoba `printf()` v Ccku
- `String readLine()`
 - vrací jeden načtený řádek (bez znaku konce řádku)
- `char[] readPassword()`
 - vrací jeden načtený řádek (bez znaku konce řádku)
 - zadávané znaky nejsou zobrazovány
- `Reader reader()`
- `PrintWriter writer()`
 - vrací reader/writer asociovaný s konzolí

Vstup a výstup

Komprese

Přehled

- balík `java.util.zip`
- komprese pomocí filtrů
 - `FilterInputStream` a `FilterOutputStream`
- `CheckedInputStream`, `CheckedOutputStream`
 - poskytují kontrolní součet čtených/zapisovaných dat
- `InflaterInputStream`, `DeflaterOutputStream`
 - základní třídy pro kompresi a dekompresi
- `GZIPInputStream`, `GZIPOutputStream`
 - komprese ve formátu GZIP
- `ZipInputStream`, `ZipOutputStream`
 - komprese ve formátu ZIP

GZIP

- komprimace jednoho souboru
- kompatibilní s UNIXovými programy gzip a gunzip

```
BufferedInputStream in = new BufferedInputStream(  
    new FileInputStream(args[0]));  
BufferedOutputStream out = new  
    BufferedOutputStream(  
        new GZIPOutputStream(  
            new FileOutputStream("test.gz")));  
int c;  
while((c = in.read()) != -1)  
    out.write(c);  
in.close();  
out.close();
```

ZIP

- komprimace více souboru do jednoho archivu
- kompatibilní se ZIP programy
- vytváření archivu
 - `ZipOutputStream`
 - **metoda** `putZipEntry (ZipEntry ze)`
 - další soubor do archivu
 - **třída** `ZipEntry`
 - jméno souboru
 - informace o souboru (velikost před kompresí a po, komentář, kontrolní součet,...)
- čtení z archivu
 - `ZipInputStream`
 - **metoda** `getNextEntry ()`
 - `ZipFile`
 - **metoda** `entries ()` - **vrací** `Enumeration`

ZIP

```
ZipOutputStream zos = new ZipOutputStream(  
    new BufferedOutputStream(  
        new FileOutputStream("test.zip")));  
zos.setComment("Testovací ZIP");  
for(int i = 0; i < args.length; i++) {  
    System.out.println("Ukladam soubor: " + args[i]);  
    BufferedInputStream in = new BufferedInputStream(  
        new FileInputStream(args[i]));  
    zos.putNextEntry(new ZipEntry(args[i]));  
    int c;  
    while((c = in.read()) != -1)  
        zos.write(c);  
    in.close();  
}  
zos.close();
```



Verze prezentace J07.cz.2020.01

Tato prezentace podléhá licenci [Creative Commons Uved'te autora-Neužívejte komerčně 4.0 Mezinárodní License](https://creativecommons.org/licenses/by-nc/4.0/).