Seminář Java

VI

Rekapitulace

- Úvod do kontejnerů - kategorie
- Iterátory
- Rozhraní List
- Rozhraní Set
- Rozhraní Map
- Volba implementace
- Nástroje
- Souběžný přístup

Obsah

- Vstup a výstup
  - Koncepce proudů
  - Práce se soubory a adresáři
  - Binární proudy, třídy `InputStream`, `OutputStream`
  - Znakové proudy, třídy `Reader`, `Writer`
  - Filtrované proudy
  - Standardní vstup a výstup
- Komprimace
- Serializace

Koncepce v/v operací (1)

- Založeny na v/v proudech
- Plně platformově nezávislé
- Transparentní přístup

Rozdělení

- znakové (`Reader/Writer`)
- binární `Stream`

Datový proud - jakýkoliv datový zdroj nebo cíl jako objekt, schopný zpřístupnit přijmout libovolné datové bloky. Ukrývá detaily toho, co se děje s daty uvnitř skutečného vstupního nebo výstupního zařízení.

Koncepce v/v operací (2)

Koncipovány jako "stavebnice"

- lze vkládat do sebe a přidávat tak vlastnosti

```
is = new InputStream(...);
bis = new BufferedInputStream(is);
```

Téměř vše je obsazaženo v balíku **java.io**

- od verze 1.4+ alternativní balík **java.nio** (new I/O)
- blíže viz. dokumentace API java.io a java.nio

Práce se soubory

- Základem je třída **java.io.File**
  - "brána" k fyzickým souborům/adresářům na disku
  - používá se jak pro soubory, tak adresáře, linky i soubory identifikované UNC jmény
  - opět platformově nezávislé
  - transparetní odlišení systémů souborů
    - `char File.separatorChar`
    - `String File.separator`
    - `char File.pathSeparatorChar`
    - `String File.pathSeparator`
    - `System.getProperty("user.dir")` - adresář uživatele pod jehož UID je proces JVM spuštěn

Třída `File` *(1)*

Vytvoření konstruktorem

- `File(String` **filename**`)` - vytvoří v aktuálním adresáři soubor s názvem **filename**
- `File(File` **baseDir**`, String` **filename**`)` - vytvoří v adresáři **baseDir** soubor s názvem **filename**
- `File(String` **baseDirName**`, String` **filename**`)` - vytvoří v adresáři **baseDirName** soubor s názvem **filename**
- `File(URL` **url**`, String` **filename**`)` - vytvoří soubor (soborovým file:) URL **url**

Třída `File` *(2)*

Testy na existence a povahy souboru

- `boolean exist()` - test na existenci souboru (nebo adresáře)
- `boolean isFile()` - test zda jde o soubor (tj. ne adresář)
- `boolean isDirectory()` - test zda jde o adresář

Test práv - čtení/zápisu

- `boolean canRead()` - test, zda lze soubor číst
- `boolean canWrite()` - test, zda lze do souboru zapisovat

Třída `File` *(3)*

Vytvoření souboru nebo adresáře

- `boolean createNewFile()` - (pro soubor) vrací true podaří-li se soubor vytvořit
- `boolean mkdir()` - (pro adresář) vrací true podaří-li adresář vytvořit
- `boolean mkdirs()` - vytvoří případně celou adresářovou cestu

Vytvoření dočasného (temporary) souboru

- `static File createTempFile(String` **prefix, String suffix**`)` - vytvoří dočasný soubor ve standardním pro to určeném adresáři s uvedeným prefixem a suffixem v názvu
- `static File createTempFile(String` **prefix, String suffix,** `File` **directory**`)` - dtto, ale vytvoří dočasný soubor v adr. **directory**

Třída `File` *(4)*

Zrušení

- `boolean delete()` - zrušení souboru nebo adresáře
- `boolean renameTo(File` **`dest`**`)` - přejmenuje soubor nebo adresář

Další vlastnosti

- `long length()` - délka souboru v bajtech
- `long lastModified()` - čas poslední modifikace souboru v ms
- `String getName()` - jen jméno souboru (tj. poslední část cesty)
- `String getPath()` - absolutní cesta k souboru i se jménem
- `String getAbsolutePath()` - celá cesta k souboru i se jménem
- `String getParent()` - adresář v němž je soubor nebo adresář obsažen
- další vlastnostni viz dokumentace API třídy File

Práce s adresáři

Základem je opět třída `File`, použitelná i pro adresáře.
Jak např. získat (filtrovaný) seznam souborů v adresáři?

- pomocí metody `File[] listFiles(FileFilter ff)`
- nebo podobné `File[] listFiles(FilenameFilter fnf)`

Třída `FilenameFilter`

- `FileFilter` je rozhraní s jedinou metodou `boolean accept(File pathname)`
- obdobně `FilenameFilter` viz dokumentace API java.io.FilenameFilter

Příklad - FilenameFilter

```
public class VypisAdresare {

  public static void main(final String[] args) {
    File cesta = new File(".");
    String[] seznam;
    if (args.length == 0)
      seznam = cesta.list();
    else
      seznam = cesta.list(new FilenameFilter() {
        public boolean accept(File adresar, String fn) {
          String soubor = new File(fn).getName();
          return soubor.indexOf(args[0]) != -1;
        }
      });
    for(int i = 0; i < seznam.length; i++)
      System.out.println(seznam[i]);
  }
}
```

Příklad - vytvoření adresáře

```
public class MakeDir {

  public static void main(String[] args) {
    String dirName = "adresar\\vnorenyadresar";
      if (args.length != 0)
        dirName = args[0];

      File adresar = new File(dirName);
      adresar.mkdirs();
  }
}
```

Práce s binárními proudy

- Vstupní jsou odvozeny od abstraktní třídy `InputStream`
  - pole bajtů
  - soubor
  - objekt typu `String`
  - roura
  - sekvence proudů
  - ...
- Výstupní jsou odvozeny od abstraktní třídy `OutputStream`
  - pole bajtů
  - soubor
  - roura
  - ...

Vstupní binární proudy

- `void close()` - uzavře proud a uvolní příslušné systémové zdroje
- `void mark(int readlimit)` - poznačí si aktuální pozici (později se lze vrátit zpět pomocí reset())...
- `boolean markSupported()` - pokud je podpodporováno
- `abstract int read()` - přečte bajt. vrací -1, když už není možné číst
- `int read(byte[] b)` - přečte pole bajtů
- `int read(byte[] b, int off, int len)` - přečte pole bajtů se specifikací délky a pozice plnění pole b
- `void reset()` - vrátí se ke značce nastavené metodou `mark(int)`
- `long skip(long n)` - přeskočí zadaný počte bajtů

Uvedené metody, kromě `abstract byte read()`, nemusejí být nutně v neabstraktní podtřídě překryty.

Některé třídy odvozené od `InputStream`

`java.io.FilterInputStream` je bázová třída k odvozování všech vstupních proudů přidávajících vlastnost/schopnost filtrovat poskytnutý vstupní proud.

- `BufferedInputStream` - proud s vyrovnávací pamětí (je možno specifikovat její optimální velikost)
- `java.util.zip.CheckedInputStream` - proud s kontrolním součtem (např. CRC32)
- `javax.crypto.CipherInputStream` - proud dešifrující data ze vstupu
- `DataInputStream` - má metody pro čtení hodnot primitivních typů
- `java.security.DigestInputStream` - počítá současně i haš (digest) čtených dat, použitý algoritmus lze nastavit
- `ProgressMonitorInputStream` - přidává schopnost informovat o průběhu čtení z proudu

Další vstupní proudy

Příklad rekonstrukce objektů ze souboru

```
FileInputStream istream = new FileInputStream("t.tmp");
ObjectInputStream p = new ObjectInputStream(istream);
int i = p.readInt();
String today = (String)p.readObject();
Date date = (Date)p.readObject();
istream.close();
```

- `ObjectInputStream` - proud na čtení serializovaných objektů
- `ByteArrayInputStream` - proud dat čtených z pole bajtů
- `PipedInputStream` - roura napojená na "protilehlý" `PipedOutputStream`
- `SequenceInputStream` - proud vzniklý spojením více podřízených proudů do jednoho virtuálního

Práce se znakovými proudy

Základem je abstraktní třída `Reader`

- `BufferedReader, CharArrayReader, InputStreamReader, PipedReader, StringReader`
- `LineNumberReader, FileReader, PushbackReader`

Výstupní proudy

- jedná se o protějšky k vstupním proudům, názvy jsou konstruovány analogicky (např. `FileReader` -> `FileWriter`)

Příklady

- `PrintStream`
- `PrintWriter`

Příklad - výpis souboru - Reader

```
public class VypisSouboru {

  public static void main(String[] args) {
    try {
      String filename = "VypisSouboru.java";
      if (args.length != 0)
        filename = args[0];

      BufferedReader in = new BufferedReader(
        new FileReader(filename));

      String line;
        while ((line = in.readLine()) != null)
          System.out.println(line);
      in.close();
    }catch(Exception ex) {
      ex.printStackTrace(System.err);
    }
  }
}
```

Příklad - kopírování souborů - Stream

```
public class CopyFile {

  public static void main(String[] args) {
  try {
    FileInputStream in = new FileInputStream(args[0]);
    FileOutputStream out = new FileOutputStream(args[1]);
    byte[] byteArray = new byte[in.available()];
    in.read(byteArray);
    out.write(byteArray);
    in.close();
    out.close();
  } catch(Exception ex) {
    ex.printStackTrace(System.err);
    }
  }

}
```

Konverze mezi proudy

Ze vstupního binárního proudu `InputStream` (čili každého) je možné vytvořit znakový `Reader`

```
//nejprve binární vstupní proud
//toho kódování znaků nezajímá
InputStream is = ...

//znakový proud použije pro dekódování
//standardní znakovou sadu
Reader isr = new InputStreamReader(is);

//sady jsou definovány v balíku java.nio
Charset chrs = java.nio.Charset.forName("ISO-8859-2");

//použije pro dekódování jinou znakovou sadu
Reader isr2 = new InputStreamReader(is, chrs);
```

Obdobně pro výstupní proudy - lze vytvořit Writer z OutputStream.

Standardní vstup a výstup

- Vstup - vrací i přijímá `InputStream`
  - `System.in`
- Výstup - vrací i přijímá `PrintStream`
  - `System.out`
  - `System.err`

Nastavení proudu provedeme metodou `setXXX` (`Out`,`Err`,`In`).

Příklad převodu `System.out` na `PrintWriter`

```
PrintWriter newOut = new PrintWriter(System.out, true);
```

Příklad - čtení ze standardního vstupu

```
public class ReadIn {

  public static void main(String[] args) {
  try {
    BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
    String line;
    while((line = in.readLine()).length() != 0)
      System.out.println(line);
    } catch(IOException ioe) {
      ioe.printStackTrace(System.err);
    }
  }

}
```

Komprimace

Nástorje pro komprimaci dat jsou umístěny v balíku `java.util.zip` API.

- Čtení a zápis proudů v komprimovaném formátu GZIP a ZIP
  - Nádstavba existujících tříd
- Funkce pro komprimaci a kontrolu dat
- Bajtově orientované

Komprimace - přehled tříd

- `CheckedInputStream` - metoda `getCheckSum()` vrací kontrolní součet vstupního proudu typu `InputStream`
- `CheckedOutStream` - metoda `getCheckSum()` vrací kontrolní součet výstupního proudu typu `OutputStream`
- `ZipInputStream` - třída typu `InflaterInputStream`, obnovuje data dříve uložená ve formátu **ZIP**.
- `GZIPInputStream` - třída typu `InflaterInputStream`, obnovuje data dříve uložená ve formátu **GZIP**.
- `ZipOutputStream` - třída typu `DeflaterOutputStream`, komprimuje data do formátu **ZIP**.
- `GZIPOutputStream` - třída typu `DeflaterOutputStream`, komprimuje data do formátu **GZIP**.

Příklad - komprimace (GZIP)

```
public class GZIPCompress {

  public static void main(String[] args) throws IOException {
    BufferedReader in = new BufferedReader(
      new FileReader(args[0]));

    BufferedOutputStream out = new BufferedOutputStream(
      new GZIPOutputStream(new FileOutputStream(args[1])));

    int b;
    while((b = in.read()) != -1)
      out.write(b);

    in.close();
    out.close();
  }

}
```

Serializace objektů (1)

- **serializace objektu** - postup, jak z objektu vytvořit sekvenci bajtů perzistentně uložitelnou na paměťové médium (disk), a později restaurovatelnou do podoby výchozího javového objektu
- **deserializace** - zpětná rekonstrukce objektu

Serializace objektů (2)

Serializovatelný objekt musí splňovat následující podmínky

- musí implementovat rozhraní `java.io.Serializable`
- proměnné objektu, které nemají být serializovány, musí být označeny modifikátorem `transient`
- pokud požaduje "speciální chování" při de/serializaci, musí objekt definovat metody
  - `private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException`
  - `private void writeObject(ObjectOutputStream stream) throws IOException`

Serializace objektů - příklad (1)

```
public class Clovek implements java.io.Serializable {

  String jmeno;

  protected Clovek() {}

  public Clovek(String jmeno) {
    this.jmeno = jmeno;
  }

  public String info() {
    return jmeno;
  }

}
```

Serializace objektů - příklad (2)

```
public class Serializace {

  private static final String soubor = "clovek.dat";

  public static void serializuj(Clovek p) throws Exception {
    FileOutputStream fos = new FileOutputStream(soubor);
    ObjectOutputStream oos = new ObjectOutputStream(fos);
    oos.writeObject(p);
    fos.close();
  }

  public static Clovek deserializuj() throws Exception {
    FileInputStream fis = new FileInputStream(soubor);
    ObjectInputStream ois = new ObjectInputStream(fis);
    Clovek p = (Clovek)ois.readObject();
    fis.close();
    return p;
  }
```

Serializace objektů - příklad (3)

```
    .
    .
    .
  public static void main(String args[]) throws Exception {
    if (args.length == 1) {
      Clovek p = new Clovek(args[0]);
      serializuj(p);
      System.out.println("Clovek (" + p.info()
        + ") serializovan.");
    } else {
        Clovek p = deserializuj();
        System.out.println("Deserializovany clovek: "
          + p.info());
    }
  }
}
```