Vstupy a výstupy v Javě

Tomáš Pitner, upravil Marek Šabo

Koncepce vstupně/výstupních operací v Javě

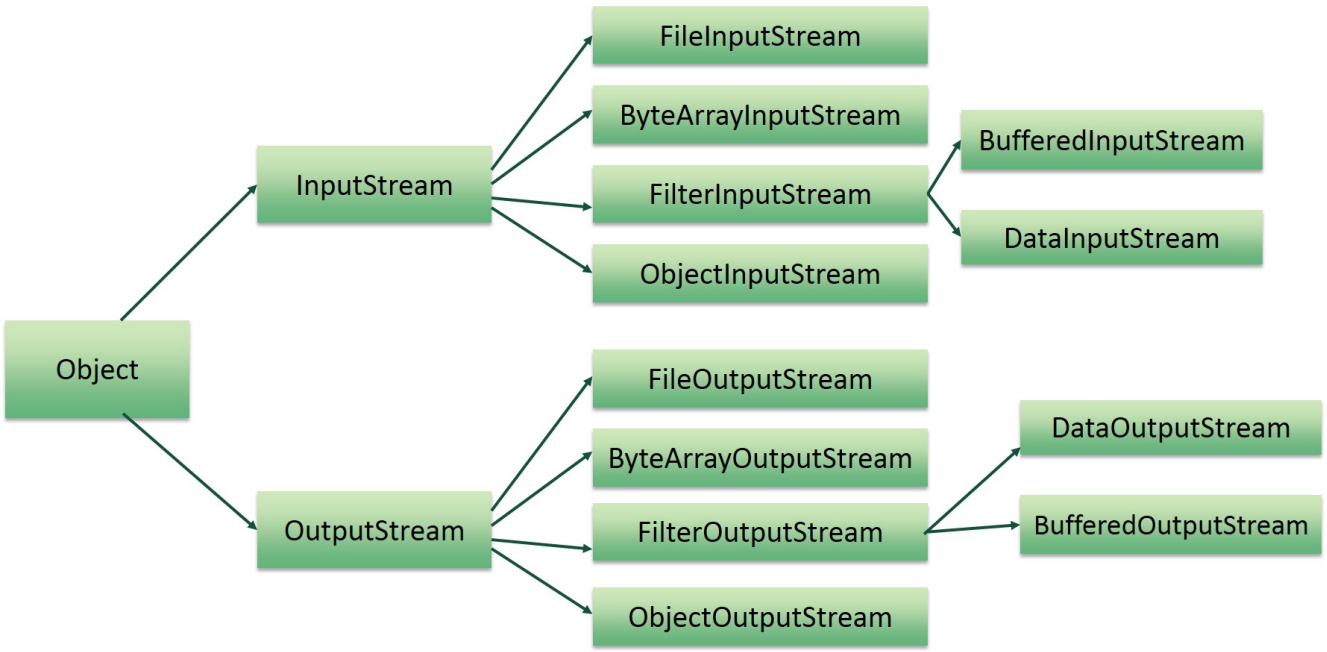
- V/V operace jsou založeny na *vstupně/výstupních proudech* (streams).
- Tím pádem je možno značnou část logiky programu psát nezávisle na tom, o který *konkrétní typ* V/V zařízení jde.
- Současně s tím jsou díky tomu V/V operace plně *platformově nezávislé*.

Table 1. Vstupně/výstupní proudy

typ dat	vstupní	výstupní
znakové	Reader	Writer
binární	InputStream	OutputStream

Vstupy a výstupy v Javě

Zdroj: http://www.tutorialspoint.com/java/



Skládání vstupně/výstupních proudů

- Proudy jsou koncipovány jako "stavebnice": lze je spojovat za sebe a tím přidávat vlastnosti, např.

```
// máme libovolný InputStream is
is = new InputStream(...);
// bis přidá k proudu is vlastnost vyrovnávací paměti
bis = new BufferedInputStream(is);
```

API proudů

- Téměř vše ze vstupních/výstupních tříd a rozhraní je v balíku java.io.
- Počínaje Java 1.4 se rozvíjí alternativní balík java.nio *(New I/O)*, zde se ale budeme věnovat klasickým I/O z balíku java.io.
- Blíže viz dokumentace API balíků java.io, java.nio.

Práce s binárními proudy

Vstupní

odvozeny od abstraktní třídy InputStream, např. FileInputStream.

Výstupní

odvozeny od abstraktní třídy OutputStream, např. FileOutputStream.

Příklad

- Uvedený příklad ilustruje proudy na kopírování binárního soubor do jiného.
- V reálu to takto neděláme, protože
  1. už existuje hotové řešení a
  2. toto je zoufale neefektivní, neboť čte i píše opravdu po bajtech a ne po větších blocích.

```
public class CopyFile {
   public static void main(String args[]) throws IOException {
      FileReader in = null;
      FileWriter out = null;
      try {
         in = new FileReader("input.txt");
         out = new FileWriter("output.txt");
         int c;
         // ve skutečnosti je v c opravdu bajt (nebo -1)
         while ((c = in.read()) != -1) {
            out.write(c);
         }
      } finally { // nutné pro korektní zavření za všech okolností
            if (in != null) {
               in.close();
            }
            if (out != null) {
               out.close();
            }
      }
   }
}
```

Hlavní metody vstupních binárních proudů

void close()
: uzavře proud a uvolní příslušné zdroje (systémové "file handles" apod.)

void mark(int readlimit)
: poznačí si aktuální pozici (později se lze vrátit zpět pomocí reset())

boolean markSupported()
: ale jen když platí tohle

abstract int read()
: přečte bajt (0-255 pokud OK; jinak -1, když už není možné přečíst)

Pokračování

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 počet bajtů

Důležité vstupní proudy (1)

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

Příklady filtrů (ne všechny jsou v java.io):

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, např. float readFloat()

Důležité vstupní proudy (2)

java.security.DigestInputStream
: počítá současně i haš (digest) čtených dat, použitý algoritmus lze nastavit

java.util.zip.InflaterInputStream
: dekomprimuje (např. GZIPem) zabalený vstupní proud (má ještě specializované podtřídy)

LineNumberInputStream
: doplňuje informaci o tom, ze kterého řádku vstupu čteme (zavrhovaná — *deprecated* — třída)

ProgressMonitorInputStream
: přidává schopnost informovat o průběhu čtení z proudu

PushbackInputStream
: do proudu lze data vracet zpět

Další vstupní proudy

Příklad rekonstrukce objektů ze souborů:

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

Další vstupní proudy (2)

javax.sound.sampled.AudioInputStream
: vstupní proud zvukových dat

ByteArrayInputStream
: proud dat čtených z pole bajtů

PipedInputStream
: roura napojená na "protilehlý" PipedOutputStream

SequenceInputStream
: proud vzniklý spojením více podřízených proudů do jednoho virtuálního

ObjectInputStream

proud na čtení serializovaných objektů

Práce se znakovými proudy

- Základem znakových vstupních proudů je abstraktní třída Reader.
- Konkrétními implementacemi jsou:
  - BufferedReader
  - CharArrayReader
  - InputStreamReader
  - PipedReader
  - StringReader
  - LineNumberReader
  - FileReader
  - PushbackReader

Znakové výstupní proudy

- Nebudeme důkladně probírat všechny typy.
- Jedná se o protějšky k vstupním proudům, názvy jsou konstruovány analogicky (např. FileReader → FileWriter).
- Místo generických metod read mají write(...).
- Za pozornost stojí PrintStream:
  1. Je to typ proudu standardního výstupu System.out (a chybového System.err).
  2. Poskytuje možnost do znakové reprezentace vypsat hodnoty různých typů.
- Obdobně PrintWriter

Konverze znakové → binární proudy

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

```
// nejprve binární vstupní proud, ten kódování znaků nezajímá
InputStream is = ...
// znakový proud isr
// 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");
// znakový proud isr2
// použije pro dekódování jinou znakovou sadu
Reader isr2 = new InputStreamReader(is, chrs);
```

- Podporované názvy znakových sad naleznete na webu IANA Charsets .
- Obdobně pro výstupní proudy - lze vytvořit Writer z OutputStream.

Povinné zavírání proudů

- Po přístupu k proudům, zejména zápisu do nich, je třeba korektně proudy uzavřít, aby se uvolnily systémové zdroje.
- Tradičně (před Java 7) se muselo dělat blokem finally:

```
String readFirstLineFromFileWithFinallyBlock(String path)
throws IOException {
   BufferedReader br = new BufferedReader(new FileReader(path));
   try {
      return br.readLine();
   } finally {
      if (br != null) br.close();
   }
}
```

Zavírání proudů try-with-resources

- Java 7 a vyšší nabízí syntakticky kratší cestu bez nepříjemného opakování kódu s finally, close atd.
- Nabízí nově tzv. try-with-resources:

```
String readFirstLineFromFile(String path) throws IOException {
   // toto je "try with resources", automaticky proud zavře
   try (BufferedReader br = new BufferedReader(new FileReader(path))) {
      return br.readLine();
   }
}
```

Více proudů

- Pomocí try-with-resources lze ošetřit i více proudů současně — zavřou se pak všechny.

```
try (
   java.util.zip.ZipFile zf =
      new java.util.zip.ZipFile(zipFileName);
   java.io.BufferedWriter writer =
      java.nio.file.Files.newBufferedWriter(outputFilePath, charset))
```

- Obecně lze do hlavičky try-with-resources dát nejen proud, ale cokoli, co implementuje java.io.Closeable.

Výpis textu PrintStream a PrintWriter

PrintStream

je typ proudu standardního i chybového výstupu System.out.

- Vytváří se z binárního proudu, lze jím přenášet i binární data.
- Většina operací nevyhazuje výjimky, čímž uspoří neustálé hlídání (try-catch).
- Na chybu se lze zeptat pomocí checkError().

PrintWriter

pro znaková data

- Vytváří se ze znakového proudu, lze specifikovat kódování.

Příklad s nastavením kódování:

```
PrintWriter writer = new PrintWriter(new OutputStreamWriter(output, "UTF-8"));
```

Serializace objektů

- Nebudeme podrobně studovat, zatím stačí vědět, že:

  serializace objektů

  postup, jak z objektu vytvořit sekvenci bajtů perzistentně uložitelnou na paměťové médium (disk) a později restaurovatelnou do podoby výchozího javového objektu.

  deserializace

  je právě zpětná rekonstrukce objektu

- Aby objekt bylo možno serializovat, musí implementovat (prázdné) rozhraní java.io.Serializable.

Serializace objektů (2)

- Proměnné objektu, které nemají být serializovány, musí být označeny modifikátorem, klíčovým slovem, transient.
- Pokud požadujeme "speciální chování" při de/serializaci, musí objekt definovat metody:
- private void readObject(java.io.ObjectInputStream stream) throws IOException, ClassNotFoundException
- private void writeObject(java.io.ObjectOutputStream stream) throws IOException

Odkazy

Tutoriál essential Java I/O: kapitola z Oracle Java Tutorial