

# Seminář Java

Proudy, Zpracování XML, Sokety

Radek Kočí

Fakulta informačních technologií VUT

Duben 2010

- Vstup/Výstup
  - koncepce proudů
  - soubory, adresáře
  - binární proudy
  - znakové proudy
  - filtrované proudy
  - komprimace
  - serializace
  
- Zpracování XML dokumentů
  
- Sokety

## Balíčky

- **java.io** – základní knihovna pro práci s proudy
- **java.nio** – alternativní knihovna od verze 1.4+ (new IO)

## I/O operace

- založeny na i/o proudech
- transparentní
- platformově nezávislé

## Datový proud

- libovolný datový zdroj či cíl schopný zpřístupnit/přijmout libovolné datové bloky
- reprezentován objektem – ukrytí detailů (skutečné akce se skutečným i/o zařízením)

## Proudy jako stavebnice

- každá třída reprezentuje jeden typ proudu (definuje jeho vlastnosti – operace)
- nové vlastnosti se dají získat vkládáním objektů (proudů) do sebe

...

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

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

...

## Třída `java.io.File`

- "obaluje" fyzické soubory (adresáře, linky) na disku
- speciality systémů souborů (transparentní odlišení)
  - `char File.separatorChar` – ('/' nebo '\\')
  - `String File.separator`
  - `char File.pathSeparatorChar` – (' :' nebo ';' )
  - `String File.pathSeparator`
  - `System.getProperty("user.dir")` – adresář uživatele s jehož UID byl spuštěn proces JVM
- soubor (adresář) je reprezentován v objektu abstraktním `pathname`
- při přístupu k zařízení se konvertuje na systémově závislé jméno
- vytvořením instance třídy `File` se k zařízení nepřístupuje!

## Konstruktory

- `File(String pathname)` – vytvoří instanci `File` konvertováním řetězce `pathname` na abstraktní pathname
- `File(String parent, String child)` – vytvoří instanci `File` z řetězce `parent` a řetězce `child`
- `File(File parent, String child)` – vytvoří instanci `File` z abstraktního pathname objektu `parent` pathname a řetězce `child`
- `File(URI uri)` – vytvoří instanci `File` konvertováním daného `file:URI` na abstraktní pathname

## Testovací operace

- `boolean exists()`
- `boolean isFile()`
- `boolean isDirectory()`
- `boolean canRead()`
- `boolean canWrite()`
- ...

## Vytvoření souboru (adresáře)

- `boolean createNewFile()` – vytvoří nový prázdný soubor podle abstraktního `pathname`; vrací `true`, pokud se operace zdaří
- `boolean mkdir()` – vytvoří adresář ...
- `boolean mkdirs()` – jako `boolean mkdir()`, vytváří i adresářovou strukturu, pokud je potřeba
- ...



Vytvoření dočasného (temporary) souboru

- `static File createTempFile(String prefix, String suffix)` – vytvoří dočasný soubor ve standardním adresáři
- `static File createTempFile(String prefix, String suffix, File directory)` – vytvoří dočasný soubor v uvedeném adresáři

## Další vlastnosti

- `boolean delete()`
- `boolean renameTo(File dest)`
- `long length()`
- `long lastModified()`
- `String getName()`
- `String getPath()`
- `String getAbsolutePath()`
- `String getParent()`
- ...

## Všechny vlastnosti a operace

- viz API documentation

## `String[] list()`

- vrací pole řetězců reprezentujících jména souborů a adresářů v adresáři daném abstraktním `pathname`

## `String[] list(FileNameFilter filter)`

- vrací pole řetězců reprezentujících jména souborů a adresářů v adresáři daném abstraktním `pathname`, které odpovídají filtru

## `File[] listFiles()`

- vrací pole abstraktních `pathname` reprezentujících soubory a adresáře v adresáři daném abstraktním `pathname`

## `File[] listFiles(FileFilter filter)`

- vrací pole abstraktních `pathname` reprezentujících soubory a adresáře v adresáři daném abstraktním `pathname`, které odpovídají filtru

## `File[] listFiles(FilenameFilter filter)`

- vrací pole abstraktních `pathname` reprezentujících soubory a adresáře v adresáři daném abstraktním `pathname`, které odpovídají filtru

## Rozhraní `FilenameFilter`

- definuje jednu metodu  
`boolean accept(File dir, String name)`
  - test, zda uvedený soubor by měl být obsažen v seznamu

## Rozhraní `FileFilter`

- definuje jednu metodu  
`boolean accept(File pathname)`
  - test, zda uvedený abstraktní `pathname` by měl být obsažen v seznamu

# Ukázka práce s adresáři

```
File cesta = new File(".");
String[] seznam;
seznam = cesta.list(new FilenameFilter() {
    public boolean accept(File adr, String cesta)
    {
        String soubor = new File(cesta).getName();
        return soubor.indexOf("x") != -1;
    }
});

for(String name : seznam) {
    System.out.println(name);
}
```

Vstupní (odvozeny od abstraktní třídy **InputStream**)

- pole bytů
- soubor
- **String**
- roura
- ...

Výstupní (odvozeny od abstraktní třídy **OutputStream**)

- pole bytů
- soubor
- ...

## Operace

- `void close()` – uzavře proud, uvolní zdroje
- `abstract int read()` – čte byte, pokud nelze, vrátí -1
- `int read(byte[] b)` – čte `b.length` bytů
- `int read(byte[] b, int off, int len)` – čte `len` bytů z proudu od pozice `off`
- `long skip(long n)` – přeskočí `n` bytů
- `void mark(int readlimit)` – poznačí aktuální pozici (`readlimit` je počet bytů, které lze přečíst, než se značka stane nevalidní)
- `void reset()` – obnovení pozice uchované při volání `mark()`
- `boolean markSupported()`



Odvozené od **FilterInputStream** → **InputStream**

- **java.io.BufferedInputStream** – proud s vyrovnávací pamětí
- **java.util.zip.CheckedInputStream** – proud s kontrolním součtem
- **javax.crypto.CipherInputStream** – proud dešifrující data
- **java.io.DataInputStream** – proud s metodami pro čtení hodnot primitivních datových typů
- ...

- analogicky k vstupním
- `InputStream`  $\Rightarrow$  `OutputStream`
- ...

```
InputStream fis = new FileInputStream("t.tmp");
InputStream bis = new BufferedInputStream(fis);
byte[] buffer = new byte[10];

int res = bis.read(buffer,0,10);
System.out.println("Pocet prect. znaku:" + res);
for(byte b : buffer)
    System.out.println(b);

bis.mark(20);

res = bis.read(buffer,0,10);
System.out.println("Pocet prect. znaku:" + res);
for(byte b : buffer)
    System.out.println(b);
```

```
bis.reset();

int b = bis.read();
System.out.println("Prect. znak po resetu:" + b);
System.out.println("Prect. znak po resetu:" +
                    (byte)b);

bis.close();
```

Vstupní (odvozeny od abstraktní třídy **Reader**)

- **BufferedReader**
- **PipedReader**
- **InputStreamReader**
- **FileReader** → **InputStreamReader**
- ...

Výstupní (odvozeny od abstraktní třídy **Writer**)

- **BufferedWriter**
- **PipedWriter**
- **OutputStreamWriter**
- **FileWriter** → **OutputStreamWriter**
- ...

# Konverze mezi proudy

Z binárního proudu (vstupní/výstupní) lze vytvořit proud znakový (vstupní/výstupní)

```
// vytvoříme binární vstupní proud
```

```
InputStream is = ...
```

```
// z is vytvoříme znakový proud (pro  
// dekódování se použije std. znaková sada)
```

```
Reader isr = new InputStreamReader(is);
```

```
// znakové sady jsou v balíku java.nio
```

```
Charset chrs =
```

```
    java.nio.charset.Charset.forName("ISO-8859-2");
```

```
// z is vytvoříme znakový proud (pro
```

```
// dekódování se použije jiná znaková sada)
```

```
Reader isr2 = new InputStreamReader(is, chrs);
```

```
InputStream fis = new FileInputStream("t.tmp");
/*1*/ Reader isr = new InputStreamReader(fis);
/*2*/ Reader isr = new InputStreamReader(fis,
    Charset.forName("ISO-8859-2"));
Reader br = new BufferedReader(isr);

char[] buffer = new char[10];

res = br.read(buffer,0,10);
System.out.println("Pocet prect. znaku:" + res);
for(char b : buffer)
    System.out.println(b);

br.mark(20);
...
```

Ve třídě `java.lang.System`

- `static InputStream in`
- `static PrintStream out`
- `static PrintStream err`

Změna standardních proudů

- `setIn(InputStream in)`
- `setOut(PrintStream out)`
- `setErr(PrintStream err)`



```
Reader r = new InputStreamReader(System.in);
BufferedReader in = new BufferedReader(r);

OutputStream fos = new FileOutputStream("o.tmp");
Writer w = new OutputStreamWriter(fos,
                                   Charset.forName("ISO-8859-2"));
PrintWriter out = new PrintWriter(w, true);

String line;
while((line = in.readLine()).length() != 0) {
    out.println(line);
}

out.close();
```

Package `java.util.zip`

---

```
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

- perzistence objektů
- získání reprezentace objektu jako sekvence bytů
- uložení reprezentace do souboru

## Deserializace

- rekonstrukce serializovaného objektu

## Serializovaný objekt

- musí implementovat rozhraní **java.io.Serializable**
  - nemá žádné metody, slouží pouze pro identifikaci serializovatelného objektu
  - všechny podřídny třídy, která implementuje toto rozhraní, jsou také serializovatelné
- proměnné objektu, které nemají být serializovány, musí být označeny modifikátorem **transient**
- pokud je při (de)serializace nutné speciální chování, musí objekt definovat tyto metody

```
private void readObject(ObjectInputStream is)
private void writeObject(ObjectOutputStream os)
```
- viz API pro **java.io.Serializable**

# Serializace objektů – příklad

```
class Clovek implements Serializable {
    String jmeno;
    protected Clovek() { }
    public Clovek(String jmeno) {
        this.jmeno = jmeno;
    }
    public String info() {
        return jmeno;
    }
}
```

# Serializace objektů – příklad

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

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

# Serializace objektů – příklad

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

## XML

- je standard konsorcia W3C jak vytvářet značkovací jazyky
- ideově vychází ze zhruba o deset let mladšího standardu SGML (Structure Generalized Markup Language).
- XML není jeden konkrétní značkovací jazyk
  - je to specifikace určující, jak mají vypadat značkovací jazyky
  - je to metajazyk
  - konceptuálně jde o zjednodušení SGML standardu, aby se usnadnila práce tvůrcům parserů (analyzátorů) a aplikací
- se základním standardem úzce souvisejí další, např. XML Namespaces, XInclude, XML Base.
- společně s dalšími standardy (XSLT, XSL-FO, XHTML, CSS...) tvoří "rodinu" standardů XML.



## Logická

- *dokument* se člení na elementy (jedna z nich je kořenová), atributy elementů, textové uzly v elementech, instrukce pro zpracování, notace, komentáře
- *dokument* je reprezentován stromem elementů

## Fyzická

- jeden logický dokument může být uložen ve více fyzických jednotkách (entitách); vždy alespoň v jedné – document entity.
- `<!ENTITY kap1 SYSTEM "kapitola1.xml">`
- `<!ENTITY swn "Softwarové noviny">`

Hlavní typy rozhraní pro zpracování XML dat:

- Stromově orientovaná rozhraní (Tree-based API)
- Rozhraní založená na událostech (Event-based API)
- Rozhraní založená na "vytahování" událostí/prvků z dokumentu (Pull API)

Implementace pro Javu

- parsery
- součást JAXP (Java API for XML Processing)  
`http://java.sun.com/xml/jaxp/index.html`
- nezávislé – např. `http://dom4j.org`

Mapují XML dokument na stromovou strukturu v paměti

- dovolují libovolně procházet vzniklý strom;
- nejznámější je Document Object Model (DOM) konsorcia W3C, viz <http://www.w3.org/DOM>

Modely specifické pro konkrétní prostředí

- pro Javu: JDOM – <http://jdom.org>
- pro Javu: dom4j – <http://dom4j.org>
- pro Python: 4Suite – <http://4suite.org>

Při analýze dokumentu generují události (podle toho, kde se nacházejí)

- zpracovávající aplikace implementuje obsluhu těchto událostí (pomocí metod – callback)
- událostmi řízená rozhraní jsou "nižší úrovně" než stromová (pro aplikaci zůstává "více práce")
- jsou úspornější na paměť (i čas), samotná analýza nevytváří trvalé objekty

Událost je ...

- začátek a konec dokumentu (start document, end document)
- komentář (comment)
- odkaz na entitu (entity reference)
- ...

Nejznámějším rozhraním je SAX  
(<http://www.saxproject.org>)

## Ukazka

```
<?xml version="1.0"?>
<doc>
  <para>Hello, world!</para>
</doc>
```

---

## Sled udalosti:

```
start document
start element: doc {seznam atributu: prazdny}
start element: para {seznam atributu: prazdny}
characters: Hello, world!
end element: para
end element: doc
end document
```

- World Wide Web Consortium (W3C)  
<http://www.w3.org/>
- XML Startkabel (EN/NL): aktuality, odkazy  
<http://xml.startkabel.nl>
- Soubor tutoriálů a on-line referencí v mnoha jazycích  
<http://zvon.org>
- O'Reilly XML.COM: články, tutoriály atd. na vysoké technické úrovni  
<http://www.xml.com>
- Free XML Software (L. M. Garshol): kolekce odkazů na nekomerční XML software  
<http://www.garshol.priv.no/download/xmltools/>
- J. Kosek: <http://kosek.cz/xml/>





## Připojení k síti

- obvykle jedno fyzické připojení
- všechna data procházejí tímto připojením
- problém přiřazení konkrétních dat konkrétní aplikaci

## Porty

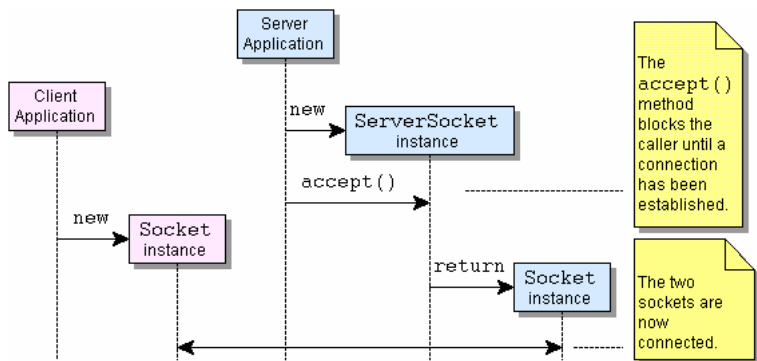
- protokoly TCP a UDP mapují data na jednotlivé procesy podle portů
- port je 16bitové číslo
- 0 – 1023 vyhraženo pro standardní služby (http, ssh, ...)
- adresace dat = adresa stroje (IP) + port

## Balíček `java.net`

- Podpora komunikace s TCP
  - `URL`
  - `URLConnection`
  - `Socket`
  - `ServerSocket`
  
- Podpora komunikace s UDP
  - `DatagramPacket`
  - `DatagramSocket`
  - `MulticastSocket`

## Sokety

- komunikace typu klient – server
  - server čeká na žádost (připojení) klienta
  - klienti se připojují na server
  - server vytváří spojení
  - klient i server zasílají/čtou data
  
- vytvořené spojení
  - soket na straně klienta
  - soket na straně serveru
  - komunikace prostřednictvím proudů



## `java.net.Socket`

- reprezentuje jednu stranu komunikace
- soket na straně klienta
  - žádost o vytvoření spojení se serverem
  - `new Socket(String host, int port)`
  - `new Socket(InetAddress address, int port)`
- čtení ze soketu
  - získání vstupního proudu
  - `InputStream getInputStream()`
- zápis do soketu
  - získání výstupního proudu
  - `OutputStream getOutputStream()`

## `java.net.ServerSocket`

- reprezentuje server
- není potomek třídy `Socket`!
- po připojení klienta vytváří objekt třídy `Socket`
- vytvoření serveru
  - `new ServerSocket(int port)`
  - `port` na kterém server naslouchá
- naslouchání spojení
  - metoda `accept()`
  - blokující operace
  - vytváří objekt třídy `Socket`

## Metody

- `getPort()`
- `getLocalPort()`
- `getInetAddress()`
- `getLocalAddress()`
- ...

# Sokety: ukázka aplikace (Server)

```
ServerSocket ss = new ServerSocket(9000);  
Socket s = ss.accept();  
  
BufferedReader in = new BufferedReader(  
    new InputStreamReader(s.getInputStream()));  
PrintStream out = new PrintStream(  
    s.getOutputStream());  
  
System.out.println(in.readLine());  
out.print("server");  
out.flush();  
  
out.close(); in.close();  
s.close(); ss.close();
```



# Sokety: ukázka aplikace (Klient)

```
try {
    s = new Socket("localhost", 9000);
    System.out.println("Client: new socket port " +
        s.getLocalPort());

    out = new PrintStream(s.getOutputStream());
    out.print("klient");
    out.flush();

    out.close();
    s.close();
}
catch (UnknownHostException ex) { ... }
catch (java.io.IOException ex) { ... }
```

## Problém uvedené ukázky

- obslouží pouze jednoho klienta a pak skončí
- ⇒ nekonečná (podmíněná) smyčka

```
ServerSocket ss;  
try {  
    ss = new ServerSocket(9000);  
    while(true) {  
        Socket s = ss.accept();  
        // obsluha klienta s  
    }  
    ss.close();  
}  
catch (java.io.IOException ex) { ... }
```

## Problém uvedené ukázky

- při obsluze klienta nemůže obsloužit další
- souběžné obslužení klientů
- ⇒ vlákna

```
ServerSocket ss;  
try {  
    ss = new ServerSocket(9000);  
    while(true) {  
        Socket s = ss.accept();  
        (new ClientThread(s)).start();  
    }  
    ss.close();  
}  
catch (java.io.IOException ex) { ... }
```

```
class ClientThread extends Thread {
    Socket s;

    public ClientThread(Socket s) {
        super();
        this.s = s;
    }

    public void run() {
        try {
            // obsluha klienta s
            s.close();
        } catch (java.io.IOException ex) { ... }
    }
}
```

# Implementace jednoduchého WWW serveru

```
public class WebServer {
    public static void main(String[] argv) {
        new WebServer().start();
    }
    public void start() {
        try {
            ServerSocket ss = new ServerSocket(9000);
            for (int conns = 0; conns < 1; conns++) {
                Socket s = ss.accept();
                (new WebClientThread(s)).start();
            }
            ss.close();
        }
        catch (java.io.IOException ex) { ... }
    }
}
```

# Implementace jednoduchého WWW serveru

```
class WebClientThread extends Thread {
    Socket s;
    public WebClientThread(Socket s) {
        super();
        this.s = s;
    }
    public void run() {
        try {
            PrintStream out = new PrintStream(
                s.getOutputStream());
            out.print("<html>"); ...
            out.print("<p>Time: " +
                (new java.util.Date()).toString());
            ...
        } catch (java.io.IOException ex) { ... }
    }
}
```

`http://java.sun.com/j2se/1.5.0/docs/guide/net/`

`http://www.webopedia.com/quick_ref/OSILayers.asp`

`http://www.javaworld.com`