JAVA - nový objektový jazyk

Petr Šaloun, Roman Szturc

Katedra informatiky, FEI VŠB - TU Ostrava, tř. 17. listopadu, 708 33 Ostrava-Poruba, Česká republika

1. Úvod

Java je jednoduchý, objektově orientovaný, distribuovaný, bezpečný a na architektuře nezávislý jazyk, viz [1]. Tyto superlativy o Javě prohlásili její tvůrci ze známé firmy Sun Microsystems. V příspěvku je nabídnut pohled pozorovatele, který zná ANSI C [3], C++ a nejlépe ještě Smalltalk [2]. Nejnovější informace totiž potvrzují začlenění smalltalkovské architektury MVC (Model-View-Controller) do Javy.

První část článku shrnuje podstatné rysy Javy, část druhá ukazuje a komentuje ukázku kódu reálné aplikace.

2. Přenositelnost

Přenositelnost, tedy nezávislost jazyka na architektuře, je podstatným rysem Javy. Moderní operační systémy řeší požadavek mikrojádrem, které tvoří mezivrstvu mezi hardwarem a přenositelným OS napsaným ve vyšším programovacím jazyce. Java, podobně jako třeba Smalltalk, využívá ze stejných důvodů *virtuální stroj*. Zdrojový text, vytvořený v jazyce Java, je přeložen do posloupnosti bajtů, která představuje instrukce virtuálního stroje a přirozeně obsahuje rovněž data. Tento kód je potom opravdu nezávislý na platformě, neboť je vykonáván (interpretován) virtuálním strojem. Přenositelnost Javy jde za zdrojový text. Java je přenositelná i na úrovni vygenerovaného proudu bajtů.

Interpretace snižuje rychlost chodu aplikace, proto jsou navrženy první mikroprocesory optimalizované pro Javu — picoJAVA, microJAVA a UltraJAVA, viz [5]. Virtuální stroj poskytuje programu v Javě možnost paralelního běhu a provádí odstranění objektů bez odkazů (garbage-collection).

3. Specifikace jazyka

Jazyk Java je navržen jako strojově a platformově nezávislý. Jazyk je zevrubně popsán. Žádná část nebo vlastnost jazyka není popsána jako implementačně závislá nebo nedefinovaná.

Snahou autorů je, aby vlastnosti každé konstrukce byly určeny specifikací jazyka, a tedy aby všechny překladače přijímaly stejné programy s výjimkou programů časově závislých či

jinak nedeterministických. Je-li k dispozici dostatek času a paměti, má Java vypočíst stejné výsledky na všech strojích.

API Javy poskytuje takové základní vlastnosti, jaké mohou být poskytovány implementacemi Javy na všech cílových platformách a prostředích.
Java je plně specifikovaný objektově orientovaný programovací jazyk založený na třídách podporující jednoduchou dědičnost (single inheritance) a pozdní vazbu (late binding). Součástí definice Javy jsou balíky (packages) specifikovaných tříd. Java vychází z C, čímž je velmi přitažlivá pro "céčkovské" programátory. Jazyk Java je přehledný a obsahuje dostatečné výrazové prostředky.

Specifikace Javy [4] je volně dostupná na Internetu.

3.1 Lexikální struktury

Zdrojový text Javy využívá kódování znaků Unicode. Skládá se z posloupnosti řádků textu ukončených přechodem na nový řádek. Tím je posloupnost znaků CR LF, nebo jeden z nich. Bílé místo je mezera, tabelátor, nová stránka a nový řádek. Komentáře jsou oproti komentářům C++ rozšířeny o takzvané dokumentační komentáře, vymezené párem /** a */. Komentáře nemohou být vnořovány a nesmí být součástí řetězců a znakových literálů.

Některé číselné literály ukazuje tabulka 1.Přehled klíčových slov jazyka je obsažen v tabulce 2.

1234	0x4d2	02322	int
98765L	0x181CDL	0300715L	long
3.14f	3.14e+0f	.14f	float
3.14	3.14e+0d	.14d	double

Tabulka 1. Literály aritmetických primitivních typů.

abstract	do	implements	package	throw
boolean	double	import	private	throws
break	else	inner	protected	transient
byte	extends	instanceof	public	try
case	final	int	rest	var
cast	finally	interface	return	void
catch	float	long	short	volatile
char	for	native	static	while
class	future	new	super	
const	generic	null	switch	
continue	goto	operator	synchronized	
default	if	outer	this	

Tabulka 2. klíčová slova Javy.

Řetězce a znaky se zapisují stejně jako v C, tedy mezi uvozovky a apostrofy. Oddělovače a operátory jsou rovněž stejné. Java rozlišuje aritmetický a bitový posun vpravo. Pro bitový (neznaménkový) posun vpravo zavádí operátor >>>, který může být kombinován s přiřazením >>>=.

3.2 Typy a hodnoty

Java má čtyři skupiny datových typů: třídy, rozhraní, pole a primitivní typy. Každá proměnná musí mít určen datový typ již při překladu. U datových hodnot jsou rozlišovány odkazy a primitivní hodnoty. Dynamicky alokované objekty mohou být instancemi tříd nebo polí. Významnou skutečností je, že primitivní typy nejsou objekty. Například, jestliže se dvě proměnné odkazují na týž objekt, je jedno, prostřednictvím které z nich je měněn stav tohoto objektu. Oproti tomu proměnná primitivního typu nemůže sdílet svou hodnotu s žádnou jinou proměnnou. Hodnota takové proměnné může být změněna pouze operacemi nad touto proměnnou.

Java definuje následující *primitivní typy*: char, byte, short, int, long, float, double a boolean. Aritmetické primitivní typy mohou být vzájemně kombinovány. Konverze je provedena automaticky, ale v některých případech může způsobit ztrátu přesnosti. Celočíselné typy jsou se znaménkem a po řadě zaujímají 8, 16, 32 a 64 bitů. Racionální typy jsou definovány ve shodě s normou IEEE 754. Logické hodnoty nemohou být zaměňovány s ostatními primitivními typy, jak umožňuje C, ale nedoporučuje jeho ANSI norma.

Celočíselné dělení nulou vyvolává výjimku (exception) ArithmeticException, což je jediný typ výjimky související s aritmetickými výpočty.

Každá proměnná Javy má definovanou implicitní (default) hodnotu nula (null), pokud ji program neinicializuje jinak.

Zavedení typové kontroly spolu s implicitní definicí počáteční hodnoty každé proměnné velmi zvyšuje bezpečnost jazyka.

3.3 Programové struktury

Zdrojový kód Javy je uspořádán do balíků, majících hierarchické pojmenování sestavené z jednotlivých identifikátorů. Globální jednoznačná jména určují internetovskou doménu. Začínají identifikátorem psaným velkými písmeny. Ostatní jména jsou vyhrazena pro lokální použití, s výjimkou předdefinovaných části jazyka a systému používajících jméno java. Například:

COM.Sun.sunsoft.DOE
CZ.vsb.FAKULTY.FEI.KINFO.kat_info

Každý balík je složen z kompilačních jednotek. Implicitní umístění nepojmenovaného balíku je v aktuálním adresáři. Kompilační jednotka deklaruje nejvýše jeden typ jako public. Toto omezení usnadňuje překladači i virtuálnímu stroji její nalezení. Příkazem import je možno určit balík nebo typ deklarovaný kdekoliv. Pokus o deklaraci importem již jednou deklarovaného jména způsobí chybu. Chybu vyvolá rovněž použití nedefinovaného jména, které je při importu nalezeno v nejméně dvou explicitně importovaných balících. Java udržuje informaci o skutečných jménech typů a balíků tak, aby při importech nemohlo existovat více cest v jejich pojmenování.

Spojování balíků probíhá dynamicky při spuštění aplikace. Dojde-li ke změně (upgrade) některého používaného balíku, umístěného kdekoliv v síti, není nutné aplikaci znovu překládat.

3.4 Deklarace tříd a typů rozhraní

Java je objektový jazyk založený na třídách. *Třída* je odvozena z jiné třídy, jíž se říká supertřída. Toto odvození je možné z jediného předka. Objekt je instancí některé z tříd.

Rozhraní (interface) určuje nový typ odkazu na množinu metod a označených konstant, ale nespecifikuje implementaci. Rozhraní může rozšiřovat rozhraní stávající. Třída může při deklaraci jejího rozhraní obsahovat odkazy na více rozhraní. Každá instance takové třídy musí definovat všechny metody určeného rozhraní. Tato násobná dědičnost rozhraní umožňuje objektům mít společné vlastnosti bez sdílení jakékoliv implementace. Deklarace rozhraní určuje rozhraní třídě samotné i jejím podtřídám. Syntaxe tvorby tříd a typů rozhraní nutí programátora jasně specifikovat vlastnosti tvořeného objektu.

Deklarace metody musí obsahovat nejen modifikátor metody, typ návratové hodnoty a argumenty metody, ale musí deklarovat i všechny normální výjimky, které během jejího provádění mohou nastat. V metodách je možno používat odkazy na this i super a přirozeně i metody přetěžovat (overloading). Speciální metodou, volanou při tvorbě objektů, je konstruktor.

Jak bylo již dříve řečeno, Java provádí automatické uvolnění paměti obsazené objekty bez odkazu.

Komentované příklady tvorby objektů, jejich rozhraní i vyvolání a zpracování výjimek jsou obsaženy v textu následujícím za specifikací jazyka.

3.5 Pole

Pole jsou v Javě dynamicky alokované objekty, které mohou být přiřazeny proměnným typu Object. Pole může volat všechny metody třídy Object. Pole jsou stejně jako v C homogenní, jednorozměrná a začínají indexem 0. Velikost pole není součástí jeho typu. Proměnná se tedy postupně může odkazovat na pole stejného typu různé velikosti. Příklady deklarace polí a jejich případné alokace:

```
int []   ai;
short          as[],
               aas[][];
Object aao[][] = new Exception[2][3];
```

4. Příklady použití

Pro lepší ilustraci jsou některé ze základních vlastností jazyka Java uvedeny na příkladu implementace jednoduchého lexikálního analyzátoru a parseru.[1] Vzhledem k tomu, že tento

[1] Jedná se o částečnou implementaci lexikálního analyzátoru a překladače uvedeného v [6].

dokument má za cíl přiblížení jazyka, není zde uveden popis jednotlivých tříd a metod, ale jsou zde uvedeny pouze části kódu charakteristické pro tento jazyk.[2]

4.1 Třída (class)

Třída LexBase tvoří základ lexikálního analyzátoru.

```
public abstract class LexBase implements LexImpl {
  public static final char char_eof = 0;

  protected char push_back_char      = char_eof;
  protected char current_char        = char_eof;
  protected int  current_state       = state_start;

  protected int        token_type;
  protected StringBuffer token_value;
  protected InputStream  input_stream;
```

Protože v definici třídy není uveden předchůdce, implicitně se předpokládá, že třída LexBase dědí z třídy Object. Navíc jde o třídu abstraktní, protože neobsahuje definice některých metod (obsahuje pouze jejich deklarace). Klíčové slovo public umožňuje přístup k této třídě i z balíků (packages) jiných, než ve kterém je tato třída definována. Konečně, klíčové slovo implements s parametrem LexImpl znamená, že třída LexBase implementuje rozhraní LexImpl.

4.2 Rozhraní (interface)

Rozhraní obsahuje deklaraci metod nutných k základní komunikaci s instancí třídy implementující toto rozhraní. Rozhraní LexImpl navíc obsahuje deklaraci několika užitečných konstant.

```
public interface LexImpl {
  public static final int state_error = 0;
  public static final int state_start = 1;
  public static final int state_end   = Integer.MIN_VALUE;
  public static final int state_eof   = state_end | 2;

  public static final int symbol_empty = 0;

  public void setInputStream(InputStream input);
  public LexToken next() throws IOException, LexException;
  public int getState();
}
```

Toto rozhraní je s výhodou použito v definici třídy ParserBase, kde proměnná lex je typu LexImpl, což není třída, ale právě rozhraní. Tím lze dosáhnout toho, že objekt v proměnné

---

[2]Kompletní kód lze získat u autorů.

lex nemusí být instancí konkrétní třídy nebo jejího potomka, ale stačí implementuje-li dané rozhraní.

```
public abstract class ParserBase implements ParserImpl
{
  protected LexImpl lex;
```

4.3 Inicializace instančních proměnných

Obdobně jako konstanty, lze přímo během deklarace inicializovat i instanční proměnné. Z tohoto důvodu je konstruktor třídy LexBase prázdný. Veškeré inicializace jsou buďto uvedeny přímo v deklaraci třídy (viz proměnné push_back_char, current_char a current_state) nebo probíhají automaticky, tj. proměnné reprezentující objekty jsou inicializovány hodnotou null (token_value a input_stream).

4.4 Práce s typem char

Jak již bylo uvedeno, pracuje Java se znakovou sadou Unicode, tzn. že typ char je reprezentován pomocí 16 bitů. Jelikož třída InputStream nedisponuje metodou, která je schopna přečíst právě 16 bitový znak, bylo nutné zvolit následující postup.[3]

```
int ch1 = input_stream.read();
int ch2 = input_stream.read();

if ((ch1 | ch2) < 0)
  return char_eof;

return (char)((ch1 << 8) + ch2);
```

Podobná komplikace nastává i v případě, že je potřeba přečtený znak ze vstupu vrátit zpět.[4]

```
protected void pushBack(char ch) throws IOException {
  if (push_back_char != char_eof)
    throw new IOException("Only one character can be pushed back");

  push_back_char = ch;
}
```

4.5 Zpracovávání výjimek (exception handling)

Významným rysem jazyka Java je zpracování výjimek. Způsob použití lze ukázat na příkladě metody perform.

```
if (new_state == state_error && current_char != char_eof) {
  String symbol;
```

[3] Vhodná metoda je definována např. v třídě `DataInputStream`. Pro větší univerzálnost je však jako vstup očekávána instance třídy `InputStream` nebo jejího potomka.

[4] Obdobná metoda je definována v třídě `PushBackInputStream`, pracuje však pouze s 8-bitovou hodnotou.

```
  if (token_value.length() > 0)
    symbol = new String(token_value);
  else
    symbol = new String(""+current_char);

  throw new LexException("Unknown symbol :", symbol);
}
```

V případě, že se analyzátor dostal do nedefinovaného stavu, tj. došlo k přečtení neznámého symbolu, dojde k vyvolání výjimky LexException. Jako parametry jsou zde použity jednak řetězec popisující tuto výjimku (standartní parametr konstruktoru třídy Exception), jednak samotný neznámý symbol (parametr konstruktoru třídy LexException, potomek třídy Exception).

Poznámka: Na rozdíl od C++, kde je uvedení seznamu generovaných výjimek v deklaraci funkce na dobré vůli programátora, je v Javě tento seznam povinný (a překladač to velmi přísně hlídá).
Způsob zachycování výjimek je použit např. v metodě parse třídy ParserBase.

```
try {
  token = getNextToken();
  stack.push(new Integer(0));
}
catch (LexException e) {
  state = state_lex_error;
  throw e;
}
```

Pokud dojde během vykonávání těla bloku try k vygenerování výjimky LexException (v tomto případě v metodě getNextToken), dojde k její zachycení a provede se tělo bloku catch, v němž se nastaví příslušný stav parseru a opět se vygeneruje původní výjimka (aby ji bylo možné zachytit ve volající metodě). Ostatní výjimky (kromě potomků třídy LexException) zde zachycovány nejsou s tím, že jejich ošetření se předpokládá ve volajících metodách.

4.6 Paralelní zpracování

Dalším významným rysem jazyka Java je možnost použití paralelního zpracování (jak procesů, tak jejich podprocesů (threads)) [7] a [8]. Použití paralelních podprocesů je ilustrováno na třídě SimpleLex, metoda prepareToken na následující straně. Její instance zajišťuje to, aby token nebyl načítán, až když ho parser o to požádá, ale iniciativně načítá souběžně s tím, jak parser zpracovává předchozí token.

Chce-li třída využívat možností paralelního běhu, musí implementovat rozhraní Runnable.[5] Navíc potřebuje několik instančních proměnných souvisejících s nově vytvořeným podprocesem a s případnou synchronizací (proměnné thread a token_ready).

---

[5]Nebo musí být potomkem třídy `Thread`. To však v tomto případě není možné, jelikož Java nepodporuje vícenásobnou dědičnost.

```
public class SimpleLex extends LexBase implements Runnable {
  protected boolean      token_ready;
  protected LexToken     token;
  protected Thread       thread;
  protected Exception    exception;
```

K vytvoření a spuštění nového podprocesu dochází ve chvíli, kdy je analyzátoru předložen vstup ke zpracování. Tím, že parametrem nově vytvářeného podprocesu je vlastní instance analyzátoru, způsobí vyvolání metody podprocesu start vyvolání metody analyzátoru run.

```
  public void setInputStream(InputStream input) {
    super.setInputStream(input);

    thread = new Thread(this);
    thread.start();
  }
```

Výše uvedená metoda run obsahuje pouze volání metody prepareToken. Klíčové slovo synchronized uvozuje část kódu, v němž několik podprocesů přistupuje ke stejné datové položce (tato část kódu se nazývá kritická sekce). Synchronizace je zajištěna pomocí instanční proměnné token_ready tak, že nabývá-li hodnoty true, je podproces vykonávající tuto metodu pozastavena pomocí metody wait.

```
  protected synchronized void prepareToken() {
    do {
      try {
        while (token_ready == true)
          wait();

        token = super.next();
      }
      catch (IOException e) {
        exception = e;
      }
      catch (LexException e) {
        exception = e;
      }
      catch (InterruptedException e) {
      }

      token_ready = true;

      notify();
    }
    while (token != null && exception == null);
  }
```

Byl-li token převzat parserem (pomocí metody obtainToken) nebo jde-li o první token, je volána rodičovská (super) metoda next, vracející přečtený token. Následně je nastavena

proměnná token_ready na true a je vyvolána metoda notify. Ta způsobí "probuzení" všech podprocesů, které souvisejí s touto instancí, a které byly "uspány" metodou wait. V případě, že dojde k vygenerování výjimky rodičovskou metodou next, je tato zachycena, uložena do instanční proměnné exception a simulována metodou obtainToken. Metody wait a notify jsou obdobně použity v metodě obtainToken vyvolávané z metody next. Za povšimnutí stojí operátor instanceof umožňující testovat, zda je objekt instancí dané třídy.

```
protected synchronized LexToken obtainToken() throws IOException,
LexException {
  try {
    while (token_ready == false)
      wait();
  }
  catch (InterruptedException e) {
  }
  if (exception != null) {
    if (exception instanceof IOException)
      throw (IOException)exception;
    else if (exception instanceof LexException)
      throw (LexException)exception;
  }

  LexToken t = token;
  token_ready = false;

  notify();

  return t;
}

public LexToken next() throws IOException, LexException {
  return obtainToken();
}
```

6. Závěr

V našem příspěvku jsme popsali Javu jako nový, na objektech založený, programovací jazyk. Její využití pro tvorbu distribuovaných aplikací, jako například stránek WWW, je námětem pro článek jiný.

7. Literatura

1. Java - Programovací nástroje. Chip, 4/1996, str. 180-182
2. LaLonde,W.R.-Pugh,J.R.: Inside Smalltalk, volume I, II. Prentice Hall, 1990
3. Plauger,P.J.-Brodie,J.: ANSI and ISO Standard C, Programmer's Reference. Microsoft Press 1992
4. The Java Language Specification, version 1.0 Beta. ftp://ftp.javasoft.com/docs/ October 30, 1995.
5. Sun Microsystems zvyšuje svůj tlak směrem na jazyk Java. ComputerWorks 13/96, str. 30.

6. Šaloun,P.: Implementace a použití generátoru překladačů v prostředí Smalltalku, Tvorba Software '95
7. Donald G. Drake, Introduction to Java threads, JavaWorld, Vol. 1 Issue 2.
8. Chuck McManis, Synchronizing threads in Java, JavaWorld, Vol. 1 Issue 2.
9. Minihofer - J. Kratochvílová, Anglicko-český slovník výpočetní techniky, SNTL, Praha 1987