Makroprocesor

Obsah:

Makroprocesor..... 1

Kapitola 1. Úvod..... 3
1.1. O aplikaci..... 3

Kapitola 2. Instalace a spuštění..... 4
2.1. Instalace..... 4
2.2. Parametry příkazové řádky..... 4
2.3. Ukázkové příklady..... 5

Kapitola 3. Popis práce makroprocesoru..... 6
3.1. Zpracování vstupního textu..... 6
3.2. Řetězce..... 6
3.3. Makra..... 6
3.4. Volání maker..... 6
3.5. Parametry maker..... 6
3.6. Kódování výstupu volaných maker..... 7
3.7. Nastavení makroprocesoru..... 7
3.8. Regulární výrazy..... 7
3.9. Seznam direktiv..... 7
3.10. Gramatika..... 10

Kapitola 4. Vestavěná makra..... 14
4.1. Základní vestavěná makra..... 14
4.2. Nástroje pro lexikální analýzu..... 15
4.3. Makra pro práci s řetězci..... 16
4.4. Další pomocná makra..... 17
4.5. Makra z balíku pmp.macro.java..... 18

Kapitola 5. Definovaná makra..... 20
5.1. Přehled speciálních symbolů:..... 20

Kapitola 6. GUI..... 21

Kapitola 7. Implementace..... 22
7.1. Přehled hlavních tříd..... 22
7.2. Vestavěná makra..... 23
7.3. Manipulace s objekty maker..... 24

Kapitola 8. Konfigurační soubory..... 25
8.1. Struktura konfiguračního souboru..... 25

Kapitola 9. Ant Task..... 26
9.1. Definice úlohy..... 26
9.2. Parametry..... 26
9.3. Podelementy..... 26

Kapitola 1. Úvod

1.1. O aplikaci

PMP je univerzální makroprocesor s plně konfigurovatelným lexikálním analyzátorem, díky kterému tento makroprocesor neklade prakticky žádné nároky na formát vstupních dat.

Makroprocesor je obecně program, který kopíruje vstup na výstup a přitom zpracovává makra nalezená v textu. Makra jsou identifikována speciálními symboly ve vstupním textu. Makra jsou dvou typů – vestavěná (*builtin*) a definovaná. Oběma typům může být předán libovolný počet parametrů.

Celý makroprocesor je napsán v jazyce Java. Proto i vestavěná makra jsou implementována pomocí tříd (viz Vestavěná makra).

Kapitola 2. Instalace a spuštění

2.1. Instalace

Ke spuštění je potřeba mít nainstalováno prostředí Java 1.5[1] nebo novější.

Celý makroprocesor se nachází ve spustitelném archivu *pmp.jar* a není tedy potřeba ho instalovat.

Pro případný překlad ze zdrojových kódů je vhodné použít Ant[2]. Vytvoření archivu *ant.jar* se provede pomocí příkazu `ant jar`.

Součásti makroprocesoru:

pmp.jar
: Vlastní makroprocesor. Žádné další soubory k spuštění nejsou potřeba.

gui.bat
: Dávkový soubor pro spuštění grafického rozhraní makroprocesoru.

applet.html
: Stránka demonstrující použití makroprocesoru jako applet.

build.xml
: Konfigurační soubor pro Ant.

packages
: Seznam balíků pro Javadoc.

meta-inf/
: Meta informace přidávané do pmp.jar.

dokumentace/
: Dokumentace k makroprocesoru (tento dokument) ve formátech OpenDocument a PDF.

docs/
: Javadoc dokumentace vygenerovaná ze zdrojových kódů.

samples/
: Ukázkové programy. Spouští se pomocí dávky run.bat s názvem programu jako parametr, např:
`run.bat loops.pmp`
Složitější příklady jsou v podadresářích.

src/
: Zdrojové kódy.

classes/
: Zkompilovaný makroprocesor.

2.2. Parametry příkazové řádky

Makroprocesor se dá pustit z příkazové řádky pomocí parametru *-jar* interpreteru Java.

syntaxe volání:

```
java –jar pmp.jar [přepínače] [--] [soubory]
```

-o {soubor}
: Specifikuje výstupní soubor.

[1] http://java.sun.com/javase/downloads/index.html
[2] http://ant.apache.org/

-d {úroveň}

Nastaví úroveň výpisu ladících informací (možno zadat číselně nebo předdefinovanou konstantu):
NODEBUG (0) – nic se nevypisuje (výchozí nastavení)
MACRO_CALLS (10) – výpis volaných maker
MACRO_RESULTS(15) – výpis výsledků volání maker
EVERY_TOKEN (20) – výpis jednotlivých lexikálních elementů

-D {makro=definice}

Definice makra. Do seznamu maker se vloží makro se zadaným jménem a obsahem.

-m {makro=třída}

Načtení třídy s makrem. Do seznamu maker se k danému jménu přiřadí instance zadané třídy.

-c {soubor}

Načte konfiguraci ze zadaného souboru (viz Konfigurační soubory).

-v

Vypíše verzi makroprocesoru a skončí.

-ie {kódování}

Definice kódování vstupních souborů.

-oe {kódování}

Definice výstupního kódování

-gui

Spustí grafické rozhraní makroprocesoru.

-version

Zobrazí číslo verze a skončí.

-help

Vypíše popis syntaxe volání a seznam přepínačů.

--

Značí konec přepínačů. Následující parametry jsou považovány za jména souborů (i pokud začínají pomlčkou).

2.3. Ukázkové příklady

V adresáři *samples* je několik souborů, na kterých je možné demonstrovat práci makroprocesoru. V souboru *README.txt* je stručný návod k použití.

Kapitola 3. Popis práce makroprocesoru

3.1. Zpracování vstupního textu

Makroprocesor rozkládá vstupní text na posloupnost lexikálních elementů (tokenů). Jejich formát je určen regulárními výrazy v direktivách makroprocesoru (viz Nastavení makroprocesoru).

V počátečním stavu makroprocesor kopíruje vstupní text na výstup, dokud nenarazí na symbol pro začátek bloku s makry (element typu code.start). Pak se v textu začnou vyhledávat makra a řetězce, které mají nastaven příznak string.*.standalone. Pokud makroprocesor narazí na makro, pokusí se před jeho vyvoláním přečíst případné parametry následující identifikátor makra.

3.2. Řetězce

Řetězce jsou posloupnosti znaků, ohraničené nadefinovanými symboly (viz Nastavení makroprocesoru). Jejich hlavním účelem je zabránit zpracování textu, který obsahují, případně ho zpracovat jiným způsobem.

Jsou tři režimy zpracování řetězců: prosté zkopírování obsahu, kopírování se zpracováním speciálních symbolů a zahození celého řetězce (viz string.*.mode).

Makroprocesor umožňuje současně používat libovolné množství různých typů řetězců.

3.3. Makra

Makra jsou dvou druhů – definovaná a vestavěná. Definovaná makra nejsou nic jiného než vlastní substituční řetězec, který má přiřazené jméno. Vytvářejí se elementem `define` v konfiguračním souboru, parametrem příkazové řádky `-D`, nebo voláním makra *`pmp.macro.Define`*.

Vestavěná makra jsou třídy v jazyce Java, které jsou podtřídou *`pmp.AbstractMacro`* a kterým je přiřazeno jméno pomocí elementu `class` v konfiguračním souboru, parametrem `-m`, nebo voláním *`pmp.macro.load`*.

3.4. Volání maker

Lexikální analyzátor makroprocesoru prochází vstupní text a vyhledává identifikátory maker. Formát identifikátoru je dán nastavením direktivy *macro.pattern*.

Pokud je nalezenému identifikátoru přiřazeno makro, pak dojde k jeho provedení. Volání vestavěných a definovaných maker se v programu nijak neliší. Výsledkem volání makra je řetězec, který je vrácen znovu na vstup, a na jeho začátku pokračuje lexikální analýza.

U definovaných maker je výsledkem řetězec získaný ze substitučního řetězce dotyčného makra dosazením parametrů. U vestavěných maker je to návratová hodnota metody `run()`.

Pokud volané makro neexistuje (ani není definována direktiva *macro.fallback*), vypíše se varování a volání se vydá na výstup. Parametry makra se zpracovávají jen, pokud makro existuje.

Definice makra se zjišťuje hned při testu existence, takže následné změny, například pomocí makra *Define* volaného z parametrů, volání nijak neovlivní.

Pokud je potřeba, aby nedocházelo ke zpracování výstupu makra makroprocesorem, lze použít funkci kódovač výstupu. Ta se zadá jako třetí parametr makra *builtin* pro vestavěná makra a pomocí direktivy *defined.encoder* pro definovaná makra. Pro výchozí nastavení se hodí použít makro *pmp.macro.lex.CEncoder*, které daný výstup převede na textový řetězec.

3.5. Parametry maker

Ve vstupním textu může za identifikátorem makra následovat libovolný počet parametrů, které jsou předány volanému makru.

Parametry jsou ohraničeny uvozovacím symbolem (direktiva *macro.opening.bracket*) a ukončovacím symbolem (direktiva *macro.closing.bracket*). Od sebe jsou jednotlivé parametry odděleny oddělovačem (*macro.paramseparator*).

Parametry makra ve vstupním textu jsou zpracovávány stejně jako ostatní text, a to včetně volání maker. K tomuto zpracování dojde před vyvoláním makra, k němuž tyto parametry patří.

Příklad:

```
#def(a, xyz)
#def(b, #defn(a))
#b == #a
```

výstup:

```
xyz == xyz
```

3.6. Kódování výstupu volaných maker

S voláním maker úzce souvisí jeden problém. Výsledkem expanze makra může být posloupnost znaků, kterou makroprocesor nějak interpretuje. Například jako další volání makra. Obecně nelze zajistit, aby výstup makra neobsahoval speciální symboly (příkladem může být volání makra Exec). Proto je nutné poskytnout mechanismus, který dalšímu zpracování výstupu maker zabrání.

Makroprocesor PMP pro tento účel poskytuje funkci kódování výstupu, která umožňuje zpracovávat výstup daného makra jiným makrem. Typickým příkladem, který dobře funguje ve výchozím nastavení, je makro *pmp.macro.lex.CEncoder*. To z textu na svém vstupu udělá řetězec znaků podle pravidel jazyka C.

Tuto funkci je možné používat jak u vestavěných maker (jako parametr makra *builtin*), tak pro uvození parametrů v uživatelsky definovaných makrech.

3.7. Nastavení makroprocesoru

Nastavení makroprocesoru se provádí pomocí direktiv. Direktivy nejsou nic jiného, než definovaná makra s daným jménem. Aby nedocházelo míchání s uživatelskými makry, všechny direktivy začínají prefixem, který má výchozí hodnotu `"pmp:"`[3].

Direktivy obsahují různé typy dat. Nejčastěji je to regulární výraz, nebo logická hodnota `"true"` nebo `"false"` (jiné hodnoty se berou jako `"false"`). Ostatní direktivy mají pevně danou množinu možných hodnot a pokud je direktiva nastavena na jinou hodnotu, makroprocesor si dosadí hodnotu, která je označena jako výchozí.

3.8. Regulární výrazy

Regulární výrazy jsou zpracovávány pomocí standardních tříd v balíku *java.regex.Pattern*. Popis syntaxe lze najít v *Javadoc* dokumentaci k třídě *Patern*.

Navíc je definován výraz `"[]"`, který představuje výraz, kterému neodpovídá žádný řetězec znaků. Použije se tam, kde je potřeba zabránit vrácení nějakého typu lexikálního elementu (např *macro.opening.bracket*, to zabrání zpracování parametrů maker).

3.9. Seznam direktiv

source.append

Text přidaný na začátek vstupního textu. Připojení se provádí před začátkem zpracování programu, takže je nutné tuto direktivu nastavit z vnějšku. Nicméně je možné sem vložit volání makra, které bude definováno až v programu.

[3] lze změnit předefinováním direktivy *prefix*

source.prepend

Text připojený na konec programu. Připojení se samozřejmě provede před začátkem zpracování programu, takže následné změny této direktivy se nijak neprojeví.

code.start

Značka pro uvození kódu (bloku s makry). Pokud je nastaveno na prázdný řetězec, přejde se do kódu okamžitě.

Výchozí nastavení: prázdné[4]

code.end

Značka pro ukončení kódu.

Výchozí nastavení: prázdné

code.marks.output

Určuje zda se mají vypisovat na výstup značky pro uvození a ukončení kódu.

Výchozí nastavení: makro není definováno

code.token

Formát lexikálních elementů v kódu.

Výchozí nastavení: `[A-Za-z0-9_]+`

macro.pattern

Regulární výraz definující možná jména makra. Je možné používat *capturing groups*[5], jméno makra je pak získáno z jejich spojení v daném pořadí.

Výchozí nastavení: `#([a-z]+)`

macro.opening.bracket

Definuje lexikální symbol uvozující parametry makra (<*opening-bracket*>).

Výchozí nastavení: `[ \t]*\([ \t]*`

macro.closing.bracket

Definuje lexikální symbol ukončující parametry makra (<*closing-bracket*>).

Výchozí nastavení: `[ \t]*\)`

macro.paramseparator

Oddělovač parametrů. Slouží k oddělení parametrů volaných maker.

Výchozí nastavení: `[ \t]*\,[ \t]*`

macro.casesensitive

V názvech maker se nerozlišuje velikost písmen. Pokud je zapnuto, pak se při volání nerozlišuje velikost písmen. Protože lze vždy vytvářet makra, jejichž název se liší pouze velikostí písmen, může danému volání odpovídat více maker. V tomto případě je použito makro, které přesně odpovídá volání a pokud není definováno, pak se náhodně použije některé jiné (není dáno které ani není zaručeno, že to bude vždy to samé).

Výchozí nastavení: `true`

macro.args.required

Určuje zda za názvem makra musí následovat seznam parametrů. Pokud je nastaveno na `true`, pak je makro vyvoláno jen pokud za ním následují parametry (tj: další token odpovídá výrazu v *macro.opening.bracket*.

Výchozí nastavení: `false`

macro.fallback

Makro, které vyvoláno, pokud se ve zdrojovém textu najde token, který je identifikován jako volání makra, ale příslušné makro není definováno. Pokud toto makro není definováno, pak se volání makra neprovede a přečtený identifikátor je pouze vydán na výstup.

Výchozí nastavení: makro není definováno

[4] tzn: makra jsou zpracovávána okamžitě od začátku vstupu.

[5] viz nápověda k *java.util.regex.Pattern*

prefix

Určuje prefix konfiguračních direktiv. Změna této direktivy se okamžitě promítne do všech části makroprocesoru, které s její hodnotou pracují.

Výchozí nastavení: `pmp:`

string.*.start

Značka pro uvození řetězce.

string.*.end

Značka pro ukončení řetězce. Pokud není nastaveno, použije se stejný symbol jako pro uvození řetězce.

string.*.marks-mode

Určuje co se má udělat se značkami pro uvození a ukončení řetězce.

copy – kopírují se na výstup
copy-start – uvozující značka se kopíruje na výstup
copy-end – ukončovací značka se kopíruje na výstup
discard – zahodí se

string.*.tokens

Definuje formát lexikálních elementů v řetězci.

string.*.symbols

Formát speciálních symbolů zpracovávaných parserem.

string.*.parser

Parser speciálních symbolů pro tento typ řetězce. Tomuto makru se jako jediný parametr předá nalezený symbol.

string.*.mode

Určuje způsob zpracování obsahu řetězce.

copy – obsah řetězce se kopíruje na výstup
discard – obsah řetězce se zahodí
parse – speciální symboly se se před vydáním zpracují přiřazeným parserem
nested – řetězec s vnořenými elementy

string.*.value

Určuje text, kterým se nahradí obsah řetězce, pokud je nastaven režim *discard*.

string.*.standalone

Určuje zda tento typ řetězce má být rozpoznáván i mimo parametry makra. Pokud zde není nastaveno `true`, pak tento řetězec bude rozpoznáván kdekoliv v kódu.

defined.encoder

Makro, které slouží z zakódování parametrů v definovaných makrech. Používá se k transformaci řetězců, tak aby v nich bylo možné používat znaky, které by se jinak přeložily jako speciální symboly.

defined.separator

Oddělovač parametrů v definovaných makrech. Viz kapitola Definovaná makra.

Výchozí nastavení řetězců

Ve výchozím nastavení jsou definovány řetězce v uvozovkách a apostrofech a komentáře ve stylu jazyka C.

Řetězec v uvozovkách

Uvnitř je možné používat escape-sekvence jazyka C. Na výstup se kopíruje obsah řetězce po zpracování escape-sekvencí.

string.quot.mode: `parse`

string.quot.marks-mode: `remove`
string.apos.standalone: `true`
string.quot.tokens: `[_0-9a-zA-Z]+`
string.quot.symbols: `\\.[0-9a-fA-F]{0,4}`
string.quot.parser: `pmp.macro.lex.CEscDecoder`
string.quot.start: `"`
string.quot.end: `"`

Řetězec v apostrofech

Na výstup se kopíruje přímo obsah řetězce.

string.apos.mode: `copy`
string.apos.marks-mode: `remove`
string.apos.standalone: `true`
string.apos.start: `'`
string.apos.end: `'`

Jednořádkový komentář uvozený dvěma lomítky

Celý komentář se při zpracování zahodí a to včetně ukončovacího znaku `'\n'`.

string.comm.mode: `discard`
string.comm.marks-mode: `remove`
string.apos.standalone: `true`
string.comm.start: `//`
string.comm.end: `\n`

Komentář /* … */

string.comm2.mode: `discard`
string.comm2.marks-mode: `remove`
string.apos.standalone: `true`
string.comm2.start: `/\*`
string.comm2.end: `\*/`

Příklad programu:

Krátký příklad, který ukazuje volání maker a použití řetězců k zabránění nechtěnému vyvolání makra.

```
#def("pmp:macro.pattern","[a-z]+")
builtin('substr','pmp.macro.str.SubStr')
def('x', "substr(ab")
def('y', 'cde, 3)')
x''y
```

výstup:

```
de
```

3.10. Gramatika

Gramatika, kterou se řídí syntaktický analyzátor při zpracovávání programu:

<input> ::= (<code-block> | <plaintext>)*

- vstupní neterminál

<plaintext> ::= <character>*

- text mimo kód je chápán pouze jako posloupnost znaků

<code-block> ::= <code-start> <code>* <code-end>

- kód (část vstupního programu, kde jsou identifikována volání maker

<code> ::= <string>

- řetězec znaků

<code> ::= <macro>

- volání makra

<code> ::= <code-token>

- posloupnost znaků, které nemají pro makroprocesor žádný speciální význam

<string> ::= <string-start> <string-element>* <string-end>

- řetězec znaků zpracovávaný makroprocesorem

<string-element> ::= <parser-token>

<string-element> ::= <string-token>

<string-element> ::= <character>

- elementy řetězce
- parser-token je dále zpracováván přiřazeným makrem

<macro> ::= <macro-name> <parameters>

- volání makra s parametry

<macro> ::= <macro-name>

- volání makra bez parametrů
- toto pravidlo platí jen pokud není nastavena direktiva *macro.args.required*

<parameters> ::= <opening-bracket> <param-list> <closing-bracket>

<param-list> ::=<param> <param-separator> <param-list>

<param-list> ::= <param>

<param> ::= <code>*

- parametry volaného makra

Význam neterminálních symbolů

<input>
: vstup

<plaintext>
: text, který se nezpracovává

<character>
: znak vstupního souboru

<code-block>
: text ve kterém se budou zpracovávat makra

<code-start>
: značka označující začátek bloku kde se budou zpracovávat makra

<code-end>
: konec bloku s makry

<code-token>
: element v kódu, který není dále zpracováván

<string>
řetězec

<string-start>
začátek řetězce

<string-end>
konec řetězce

<string-element>
prvek řetězce

<string-token>
speciální symbol řetězce

<macro>
volání makra

<macroname>
jméno makra

<parameters>
parametry makra

<opening-bracket>
symbol uvozující parametry makra

<param-list>
vlastní parametry

<closing-bracket>
symbol uzavírající parametry makra

<param-separator>
oddělovač parametrů

<param>
vlastní hodnota parametru

<>
prázdné slovo

Lexikální analyzátor

Rozděluje vstupní text na tokeny (posloupnosti znaků, které tvoří terminální symboly). Jeho práce řízena stavem syntaktického analyzátoru, který určuje jakým způsobem se má zpracovávat vstup. Stav syntaktického analyzátoru závisí na posledním tokenu zpracovaném syntaktickým analyzátorem. Spolu s preferencí pravidel to umožňuje lexikálnímu analyzátoru jednoznačně identifikovat neterminály na vstupu. V případě nejednoznočnosti je vždy vrácen první token z posloupnosti:

```
   <code-start>, <string-start>, <opening-bracket>, <param-separator>,
<macroname>, <character>, <closing-bracket>, <code-end>
```

Syntaktický analyzátor

Přijímá tokeny z lexikálního analyzátoru. Interpretuje je podle typu aktualizuje svůj stav[6]. K práci s makry mu slouží zásobník, kam ukládá načtené parametry pro makra, která ještě nemají načtené všechny parametry.

[6] stavy jsou označeny slovem v hranatých závorkách

Stavy automatu a možné výstupy lexikálního analyzátoru

[plaintext]

Výchozí stav. Lexikální analyzátor hledá symbol <*code-start*>, jinak vydává přečtené znaky.

[code-block]

Lexikální analyzátor vydává symboly pro identifikátory maker, komentáře, řetězce a pro konec bloku a tokeny podle nastavení direktivy `token`.

[string]

Vydává znaky, speciální symboly a symbol <*string-end*>. Speciální symboly jsou posloupnosti znaků definované v direktivách "string.{*název*}.symbols" a zpracované makrem (obvykle třídou) určeným dirktivou "string.{*název*}.parser". Tomuto makru je předán speciální symbol jako první parametr. Pokud je možné interpretovat vstup více způsoby, je nejdříve vydán <*string-end*>, pak speciální symbol parseru a nakonec řetězcový token.

[macro]

Vydá token nebo <*opening-bracket*>. Syntaktický analyzátor podle dalšího tokenu rozhodne o dalším stavu a akci – buď zavolat přečtené makro, nebo začít ukládat parametry na zásobník.

[params]

Vydá stejné tokeny jako [code-block] a navíc <*param-separator*>, <*closing-bracket*>.

Možné přechody mezi stavy:

stav	**přijatý token**	**nový stav**
[plaintext]	<*code-start*>	[code-block]
[code-block]	<*macro*>	[macro]
[code-block]	<*comment-start*>	[comment]
[code-block]	<*string-start*>	[string]
[macro]	<*opening-bracket*>	[params]
[params]	<*closing-bracket*>	[code-block] / [params]
[code-block]	<*code-end*>	[plaintext]
[string]	<*string-end*>	[code-block] / [params]

Kapitola 4. Vestavěná makra

Vestavěná makra jsou definována jako třídy v balících (*packages*) odvozených od *pmp.macro* (např. *pmp.macro.Include*) a výchozí stav u většiny z nich je, že jsou definována v tabulce maker pod svým jménem bez názvu balíku.

V následujícím seznamu je u maker, která jsou použita ve výchozím nastavení, uvedeno v závorce přiřazené jméno.

4.1. Základní vestavěná makra

pmp.macro.Include (include)
Provede vložení zadaného souboru na vstup makroprocesoru.
Parametry: jméno souboru (URI)

pmp.macro.Output (output)
Provede přesměrování výstupu do jiného souboru. Je možné zadat speciální identifikátory *pmp://stdout*, *pmp://stderr* a *pmp://null*.
Parametry: jméno výstupního souboru (URI)

pmp.macro.Error (error)
Ukončí zpracování a vypíše chybu.
Parametry: text chybové zprávy

pmp.macro.Warning (warning)
Vypíše varování.
Parametry: text varovné zprávy

pmp.macro.Echo (echo)
Vypíše daný text přímo na výstup.
Parametry: text

pmp.macro.Define (def)
Definuje nové makro.
Parametry: jméno makra
obsah makra

pmp.macro.Builtin (builtin)
Načte třídu s makrem. Třída musí být potomkem *pmp.AbstractMacro*.
Parametry: jméno makra
jméno třídy
[kódovač výstupu]

pmp.macro.IsDefined (ifdef)
Testuje, zda se jedná o uživatelsky definované makro. Vrací 1 pokud je makro definováno, jinak prázdný řetězec.
Parametry: jméno makra

pmp.macro.IsSet (ifdef)
Testuje, zda je makro definováno. Vrací `"1"` pokud je makro definováno, jinak prázdný řetězec.
Parametry: jméno makra

pmp.macro.IsBuiltin
Vrací `"1"`, pokud je zadané makro vestavěné. Pokud je definované, pak vrací `""`.
Parametry: jméno makra

pmp.macro.Undefine (undef)
Zruší definici makra. Nefunguje pokud bylo makro vytvořeno s příznakem *read-only*.

Parametry: jméno makra

pmp.macro.Defn (defn)

Výpis definice makra. Vrací obsah makra. Pro vestavěná makra vrací prázdný řetězec.

Parametry: název makra

pmp.macro.Copydef (copydef)

Zkopíruje existující makro. Umožňuje definovat pro jedno makro více jmen. Tato dvě přiřazení jsou ale na sobě nezávislá; pozdější změna přiřazení makra k jednomu jménu nemá vliv na druhé jméno. Makro *copydef* umožňuje přejmenovat všechna vestavěná makra.

Parametry: nové jméno
původní jméno

pmp.macroCall (call)

Nepřímé volání makra. Umožňuje volat makra jejichž název neodpovídá regulárnímu výrazu pro identifikaci jmen maker a nelze je tedy volat přímo.

Parametry: jméno volaného makra
parametry předané volanému makru

pmp.macro.Pushdef (pushdef)

Provede předefinování makra. Obnovit původní definici je možné pomocí *popdef*.

Parametry: jméno makra
obsah makra

pmp.macro.Popdef (popdef)

Obnovení předchozí definice makra. Pokud makro již nemá žádnou další definici, pak je oddefinováno.

Parametr: jméno makra

pmp.macro.MacroTable (macros)

Makro pro manipulaci s tabulkou maker. Umožňuje provést import konfiguračního souboru a export současného stavu. Nemění makra s příznakem *read-only*.

Možné akce jsou: `export`, `import` a `replace` (nahrazení současného seznamu obsahem konfiguračního souboru).

Parametry: požadovaná akce
jméno souboru

pmp.macro.Halt (halt)

Ukončí práci makroprocesoru.

Parametry: typ zprávy zprávy (chyba, varování,…)
text vygenerované zprávy

pmp.Version

Vypíše informace o verzi makroprocesoru.

4.2. Nástroje pro lexikální analýzu

pmp.macro.lex.CEncoder

Zakóduje daný vstup do řetězce dle syntaxe jazyka C.

Parametry: vstupní text

pmp.macro.lex.CEscDecoder

Analyzátor řetězcových symbolů podle syntaxe jazyka C. Ve výchozím nastavení se používá pro zpracování speciálních symbolů v řetězcích.

Parametry: speciální symbol

pmp.macro.lex.JavaEscDecoder

Analyzátor řetězcových symbolů podle jazyka Java.

pmp.macro.lex.HTMLEntityDecoder
Analyzátor znakových entit z HTML/XML.
Parametry: znaková entita

4.3. Makra pro práci s řetězci

pmp.macro.str.SubStr
Vrací podřetězec daného řetězce.
Parametry: řetězec
počáteční index vráceného podřetězce (počítáno od 0)
[maximální délka podřetězce]

pmp.macro.str.Length
Vrací délku zadaného řetězce.
Parametry: řetězec znaků

pmp.macro.str.Locate
Vrací pozici podřetězce v daném řetězci. Vrací prázdný řetězec, pokud neuspěje.
Parametry: hledaný podřetězec
prohledávaný text

pmp.macro.str.Compare
Porovnává dva řetězce. Pokud jsou stejné vrací `"1"`.
Parametry: první řetězec
druhý řetězec
[příznak `"ignore-case"`, jinak se porovnává s rozlišením velikosti písmen]

pmp.macro.str.LowerCase
Převede velká písmena v textu na malá.
Parametry: vstupní text

pmp.macro.str.UpperCase
Převede malá písmena v textu na velká.
Parametry: vstupní text

pmp.macro.str.Trim
Odstraní mezery z konce a/nebo začátku řetězce.
Parametry: text
režim (*LEADING*, *TRAILING*, *BOTH*)

pmp.macro.str.Match
Porovnává daný řetězec s regulárním výrazem. Vrací `"1"`, pokud řetězec odpovídá výrazu.
Parametry: řetězec
regulární výraz

pmp.macro.str.Replace
Nahradí všechny výskyty daného vzoru v textu zadanou náhradou
Parametry: text
hledaný vzor (regulární výraz)
náhrada

pmp.macro.str.Rot13
Provede „šifru“ Rot13 na zadaný text.
Parametry: vstupní text

pmp.macro.str.MD5
spočítá MD5 kontrolní součet ze zadaného řetězce
Parametry: vstupní text

pmp.macro.str.Adler32
Spočítá kontrolní součet Adler32 pro zadaný text.
Parametry: vstupní text

pmp.macro.str.CRC32
Spočítá kontrolní součet CRC32 pro zadaný text.
Parametry: vstupní text

pmp.macro.str.Base64
Kóduje a dekóduje Base64.
Parametry: směr (*ENCODE* / *DECODE*)
text

pmp.macro.str.UUID
Vygeneruje náhodný UUID/GUID identifikátor.

pmp.macro.str.IsNumber
Vrací "1", pokud je zadaný text platným celým číslem podle specifikace Javy.
Parametr: text

pmp.macro.str.Chr
Vrací znak s daným kódem.
Parametry: kód znaku

pmp.macro.str.Ord
Vrací kód prvního znaku v daném řetězci.
Parametr: znak

4.4. Další pomocná makra

pmp.macro.math.Eval
Provede vyhodnocení zadaného výrazu. Vrací výslednou hodnotu. Umí vyhodnocovat základní matematické operace, včetně modulo ("%"), a disponuje sadou vestavěných funkcí (abs, signum, sin, cos, log,…). Detailní popis syntaxe je uveden v Javadoc dokumentaci této třídy.
Parametry: aritmetický výraz

pmp.macro.math.IfNotZero
Vyhodnotí zadaný výraz a podle výsledku vrátí hodnotu příslušného parametru.
Parametry: aritmetický výraz
výsledek v případě, že hodnota výrazu není nula
výsledek v případě, že hodnota výrazu je nula

pmp.macro.math.Min
Vrátí nejmenší číselnou hodnotu nalezenou mezi parametry. Pokud žádnou nenalezne, vrátí 0.
Parametry: vstupní hodnoty

pmp.macro.math.Max
Vrátí největší číselnou hodnotu nalezenou mezi parametry. Pokud žádnou nenalezne, vrátí 0.
Parametry: vstupní hodnoty

pmp.macro.math.Dec
Sníží číselnou hodnotu v definici makra.
Parametry: jméno makra (pokud neobsahuje číslo, vezme se 1)
hodnota o kterou se má snižovat (výchozí hodnota 1)

pmp.macro.math.Inc
Zvýší číselnou hodnotu v definici makra.
Parametry: jméno makra (pokud neobsahuje číslo, vezme se 0)
hodnota o kterou se má zvyšovat (výchozí hodnota 1)

pmp.macro.math.Random

Vrací náhodné číslo z daného rozsahu. Pokud je makro zavoláno s jedním parametrem, považuje se ten za horní mez.

Parametry: dolní mez (včetně); výchozí hodnota 0
horní mez (vyjma); výchozí hodnota 2^31

pmp.macro.math.Const

Vrátí hodnotu zadané konstanty (PI, SQRT_2, SQRT_3, apod).

Parametry: název konstanty (kompletní přehled viz Javadoc dokumentace této třídy).

pmp.macro.tools.Exec

Spustí program se zadanými parametry. Volání tohoto makra se nahradí výstupem volaného programu.

Parametry: název programu
parametry pro spouštěný program

pmp.macro.tools.GetProperty

Vrací hodnotu systémové proměnné (*system propery*, viz dokumentace *java.lang.System*).

Parametry: jméno proměnné

pmp.macro.tools.SetProperty

Smaže nebo zruší obsah systémové proměnné.

Parametry: jméno
nová hodnota; pokud chybí, je proměnná smazána

pmp.macro.tools.Pause

Pozastaví běh makroprocesoru po danou dobu. Podporuje několik režimů:

SLEEP – prosté čekání
WAIT – čekání na uplynutí intervalu a pak na stisk klávesy
INTR – čekání na uplynutí intervalu nebo na stisk klávesy

Poznámka: interakce s klávesnicí nefunguje v prostředí Ant z důvodu přesměrování vstupu

Parametry: čas v milisekundách
režim

4.5. Makra z balíku pmp.macro.java

Tato makra umožňují psát jednoduché skripty pro manipulaci s Java objekty. Manipulace s objekty probíhá pomocí identifikátorů ve tvaru *{jméno třídy}.{číslo objektu}*. Tyto identifikátory jsou vraceny makry pro vytváření objektů a makra pro vyvolávání metod je očekávají na místě parametrů.

pmp.macro.java.Construct

Vytvoří novou instanci dané třídy.

Parametry: jméno třídy
parametry konstruktoru

pmp.macro.java.New

Vytvoří instanci wrapper objektu pro zadaný primitivní typ, nebo pole tohoto typů.

Parametry: jméno primitivního typu (v hranatých závorkách může následovat délka pole)
hodnoty...

pmp.macro.java.NewString

Makro pro jednodušší vytváření objektů typu String.

Parametry: textová konstanta

pmp.macro.java.Invoke

Zavolá metodu daného objektu, nebo statickou metodu dané třídy.

Parametry: jméno třídy nebo identifikátor objektu
jméno metody

parametry (identifikátory objektů)...

pmp.macro.java.Field

Vrací hodnotu, nebo nastavuje novou, daného pole specifikovaného objektu

Parametry: jméno třídy nebo identifikátor objektu
jméno pole
[nová hodnota (identifikátor objektu)]

pmp.macro.java.ToString

Zavolá metodu *toString()* daného objektu a vrátí výsledek.

Parametry: identifikátor objektu

pmp.macro.java.ToClassName

Vrátí jméno třídy daného objektu

Parametry: identifikátor objektu

pmp.macro.java.InstanceOf

Vrací "1" pokud je daný objekt instancí dané třídy nebo rozhraní.

Parametry: identifikátor objektu
jméno třídy nebo rozhraní

pmp.macro.java.Cast

Změní typ daného objektu. Cílovým typem musí být předek třídy tohoto objektu, nebo implementované rozhraní.

Parametry: identifikátor objektu
jméno třídy nebo rozhraní

Kapitola 5. Definovaná makra

Definovaná makra se vytvářejí pomocí vestavěných maker, jako je například *pmp.macro.Define*. Tomuto makru se zadává jako druhý parametr substituční řetězec, kterým je ve zdrojovém textu nahrazeno volání makra a případné parametry.

Aby bylo možné používat v definovaných makrech parametry, jsou definovány speciální symboly, které jsou při expanzi makra nahrazeny parametry volaného makra.

5.1. Přehled speciálních symbolů:

$$

Nahradí se znakem `$`.

$#

Vypíše počet parametrů jako dekadické číslo. Do výsledku se započítává i nultý parametr (jméno makra).

$*n*

Pokud je *n* číslice k se vypíše příslušný parametr 0 do 9 vypíše příslušný parametr. Numerická hodnota znaku se zjišťuje z Unicode tabulky[7], takže je možné zadat i jiné číslice než 0-9.

${*nnn*}

Nahradí se parametrem s číslem *nnn.*

${*nnn*-*mmm*}

Nahradí se parametry v rozsahu *nnn* až *mmm.*

$*

${*}

Nahradí se parametry makra (bez parametru nula) oddělenými znakem čárka.

${*makro}*

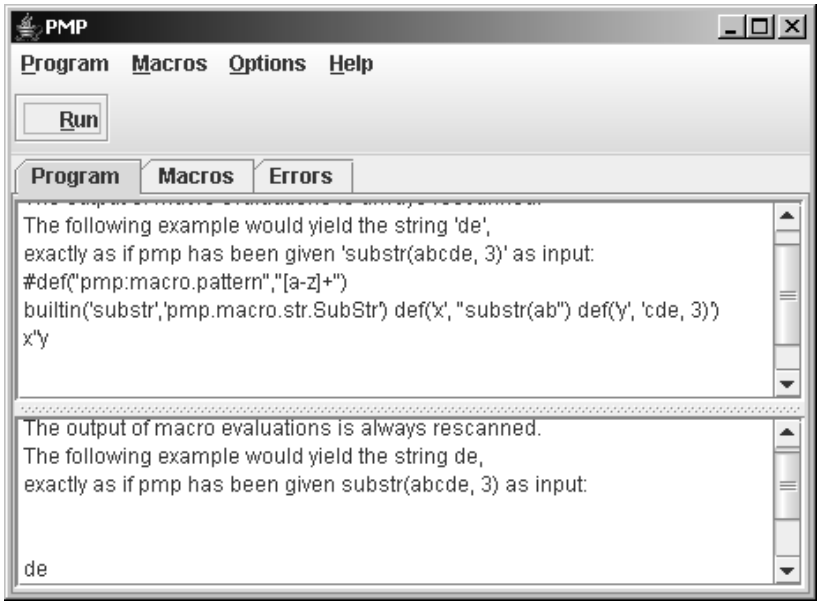
Nahradí se definicí daného makra.

Složené závorky ve speciálních symbolech je možné nahradit kulatými (např: `$(4-6)` místo `${4-6}`). V tomto případě dojde ke zpracování obsahu makrem (enkodérem) definovaným v direktivě *defined.encoder* a jednotlivé parametry budou odděleny řetězcem získaným z direktivy *defined.separator*.

[7] pomocí metody *java.lang.Character.getNumericValue(char)*

Kapitola 6. GUI

Grafické rozhraní slouží hlavně k otestování funkčnosti složitějších maker.



Obrázek 6.1 Hlavní okno grafického rozhraní

Nabídka program obsahuje příkaz *Run* pro spuštění zadaného programu a příkaz *Quit*, který ukončí grafické rozhraní.

Nabídka **Macros** obsahuje příkazu pro manipulaci s makry. Položka *Load default configuration* provede znovunačtení výchozího nastavení makroprocesoru. K načtení konfigurace z externího souboru slouží příkaz *Load config file*. Přepínač *Always reload macros* zajistí obnovení výchozího stavu před dalším spuštěním programu.

Nabídka **Options** obsahuje pouze volbu *Show error dialog*, která umožňuje zakázat zobrazování chybového dialogu při výskytu chyby. Bez ohledu na stav této volby, jsou všechny chyby zaznamenávány do seznamu v záložce *Errors*.

Panel nástrojů obsahuje zatím pouze tlačítko pro spuštění programu (*Run*).

Hlavní částí okna jsou záložky *Program*, *Macros*, *Errors*. Záložka **Program** je rozdělena na dvě části. Editační box v horní polovině slouží k zadání vstupního programu. V dolní části se zobrazí výstup makroprocesoru. Záložka **Macros** obsahuje přehled maker. Tlačítko *Change* slouží ke změně definovaného makra. Pokud není zaškrtnuta volba *Always reload macros*, bude při příštím spuštění makroprocesoru použita tato změněná hodnota. Pod záložkou **Errors** je umístěn seznam chybových a varovných hlášení.

Kapitola 7. Implementace

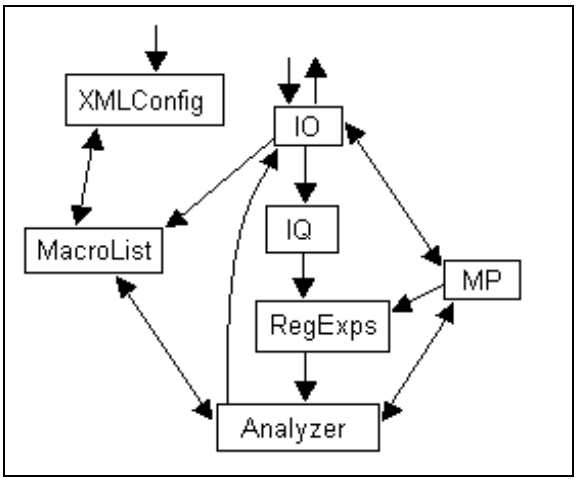
Vlastní makroprocesor je implementován jako třída, která má přiřazeno několik komponent. Těmi hlavními jsou InputOutput, InputQueue, MacroList a Analyzer. Vzájemné vazby mezi těmito komponentami zobrazuje obrázek 5.1.

InputOutput (IO) představuje vstupně-výstupní část makroprocesoru. Tento objekt zajišťuje načítání programu a předávání zpráv a chybových hlášení makroprocesoru.

InputQueue je interním vyrovnávací paměť mezi objekty InputOutput a Analyzer.

Analyzer se skládá z kolekce objektů RegExps (lexikální analyzátor) a objektu Analyzer (syntaktický analyzátor).

MacroList je rozhraní k tabulce maker.



Obrázek 7.1: Schematický přehled vzájemné komunikace jednotlivých komponent.

7.1. Přehled hlavních tříd

pmp.AbstractMacro

Abstraktní předek všech maker.

pmp.DefinedMacro *extends pmp.AbstractMacro*

Instance této třídy reprezentují definovaná makra. Metoda run zajišťuje expanzi makra podle pravidel popsaných v kapitole Definovaná makra.

pmp.Version *extends pmp.AbstractMacro*

Aktuální číslo verze makroprocesoru.

pmp.cli.Launcher

Hlavní třída makroprocesoru, která podle zadaných parametru provede příslušnou akci (spustí požadované rozhraní makroprocesoru, zobrazí nápovědu, apod...)

pmp.applet.PMPApplet

Applet, který umožňuje spustit makroprocesor v rámci WWW prohlížeče.

pmp.cli.Main

Hlavní třída pro spuštění makroprocesoru z příkazové řádky

pmp.gui.MainWindow

Hlavní třída pro spuštění grafického rozhraní (obsahuje pouze vlastí okno, nikoliv obsah).

pmp.gui.MainPanel

Třída zajišťující grafické rozhraní

pmp.Analyzer

Obsahuje syntaktický analyzátor makroprocesoru.

pmp.RegExps

Obsahuje lexikální analyzátor a metody pro manipulaci s regulárními výrazy.

pmp.InputQueue

Třída zajišťující čtení vstupních dat. Pracuje s objekty *pmp.InputSource*.

pmp.InputOutput

Třída zajišťující čtení vstupních dat. Pracuje s objekty *pmp.InputSource*.

pmp.InputSource

Třída zastřešující všechny vstupně/výstupní operace.

pmp.MacroList

Tabulka maker. Implementuje datovou strukturu pro uložení maker a metody pro manipulaci s nimi.

pmp.ConfigDirectives

Tato třída obsahuje názvy direktiv makroprocesoru. Přehled položek i s popisem najdete v Javadoc dokumentaci.

pmp.Macroprocessor

Hlavní třída makroprocesoru zajišťující vzájemnou spolupráci jednotlivých komponent (které jsou potomky vnitřní třídy *Macroprocessor.Component*).

pmp.xml.XMLConfig

Třída zajišťující načítání konfiguračních souborů.

7.2. Vestavěná makra

Vestavěná makra jsou potomci třídy *pmp.AbstractMacro*.

public AbstractMacro()

Nové instance maker makroprocesor vytváří pomocí konstruktoru bez parametrů.

public static int PARAMS_REQUIRED

Minimální počet parametrů předaných makru při vyvolání. Pole parametrů je vždy doplněno na požadovanou délku prázdnými řetězci.

String run(Macroprocessor mp, String[] params)

Metoda, kterou je voláno makro. Parametr `mp` obsahuje odkaz na instanci objektu *pmp.Macroprocessor*, která makro volá. Pole `params` obsahuje parametry makra. Prvek s indexem 0 vždy obsahuje jméno, pod kterým bylo toto makro voláno.

7.3. Manipulace s objekty maker

Makroprocesor může v průběhu zpracování programu instanciovat více objektů téže třídy, například může nahradit existující instanci nově vytvořeným objektem. Proto by objekty maker neměly obsahovat žádná data, která by nějak ovlivňovala práci makra. Pokud makro potřebuje ukládat nějaká data, mělo by ukládat svoje data do tabulky maker pod zvoleným identifikátorem. Například pomocí:

```
public static final String uuid =
      java.util.UUID.randomUUID().toString();

mp.getMPMacros().
      setMacro(mp.getPrefix() + uuid, "hodnota");
```

Makra nemusí používat synchronizaci, pracovat s makrem bude vždy pouze jedno vlákno a všechny metody jednotlivých komponent makroprocesoru budou vždy případné synchronizace provádět interně.

Kapitola 8. Konfigurační soubory

Konfigurační soubory umožňují provést snadnou konfiguraci makroprocesoru. Mají formát XML[8] a jejich struktura je definována pomocí pomocí DTD[9]. Výchozí nastavení je umístěno v jar archivu v souboru *pmp/config/config.dtd.*

8.1. Struktura konfiguračního souboru

Kořenový element má jméno `config`. Jeho podelementy jsou:

define
: Definice makra. Jméno je v atributu `name`. Atribut `readOnly` určuje zda bude možné definici makra změnit.

class
: Načtení makra z *.class* souboru. Jméno je v atributu `name`.

import
: Přidá konfigurační data ze souboru zadaného v atributu `url`.

input
: Určí vstupní program makroprocesoru.

library
: Umístění knihovny (jar archiv) s vlastními makry v class souborech.

[8] Extensible Markup Language
[9] Document Type Definition

Kapitola 9. Ant Task

9.1. Definice úlohy

Import do Ant se provede pomocí příkazu *taskdef, např:*

```
<taskdef resource="pmp/ant/pmp.properties"
   classpath="pmp.jar" />
```

9.2. Parametry

Atribut	*Popis*
debug	určuje množství vypisovaných ladících informací
src	zdrojový adresář
dest	cílový adresář
srcEncoding	kódování zdrojových souborů
destEncoding	výstupní kódování
defaultConfig	určuje, zda má se použít výchozí konfigurace
failonerror	určuje, zda se má ukončit zpracovávání souborů, pokud se narazí na chybu
importPrefix	prefix pro import proměnných (properties) z Antu

9.3. Podelementy

Tato úloha je založená na třídě FileSet a je možné používat podelementy této úlohy (*include*, *exclude*, apod...).

Pro transformaci jmen souboru je možné vložit *Mapper*.

Element config

Tento element odpovídá kořeni konfiguračního souboru, tak jak je popsán výše.

Element export

Tento element určuje, která makra se mají exportovat jako Ant property po skončení práce makroprocesoru.

Podelement *macro* exportuje dané makro a podelement *classname* exportuje název třídy, která odpovídá danému makru.

Jméno makra určeno textovým obsahem elementu.

Atribut	*Popis*
name	název, pod jakým se má makro exportovat
mode	určuje, jak se má export provést: first – exportuje se první definovaná hodnota last – exportuje se poslední definovaná hodnota concat – exportují se všechny hodnoty spojené do jednoho řetězce sum – exportuje se součet číselných hodnot

Podelement *all* zajistí export všech maker.

Příklad použití:

```
<pmp src="src"
        dest="${build}"
        srcEncoding="utf-8" >
      <include name="*.pmp" />
      <mapper type="glob" from="*.pmp" to="*.txt" />
</pmp>
```

Zavolá makroprocesor na všechny soubory s příponou *pmp* v adresáři *src* a uloží výstup do adresáře definovaného proměnnou *build*. Zdrojové soubory by měli mít kódování *utf-8*. Zapisované soubory budou mít příponu *txt* a budou ve výchozím kódování dané platformy. Struktura podadresářů zůstane zachována, ale ve výstupu se vytvoří pouze adresáře, které obsahovali nějaký soubor **.pmp*.