

```

1 < 2      # true
1 <= 1     # true
1 < 1      # false
2 < 1 + 2  # true (najskôr prebehne sčítanie)
1 < 2 > 3   # chyba syntaxe:
            #   testy sa nedajú kombinovať
'1' < 2     # true (reťazec '1' bude
            #   prevedený na číslo 1)
'add' < 'adder' # pri reťazcoch používajte
            #   lt; s prepínačom -w dôjde k chybe
'add' lt 'adder' # true
'add' lt 'Add'  # false; malé písmená sa líšia,
            #   veľké sú 'menšie'
'add' eq 'add ' # false; líši sa počet medzier

```

V prvom rade si treba všimnúť, že žiadny z týchto operátorov sa nemôže kombinovať s operátorom rovnakého typu. Zatiaľ čo $5 + 1 - 2$ je v poriadku, $5 > 4 < 6$ už vypíše chybu syntaxe.

Dost často sa vyskytuje chyba, že pri numerických operátoroch si začínajúci programátori mylia znak priradenia (=) a znak operátora (==).

Logické operátory

Logické (boolovské) operátory vyhodnocujú svoje operandy podľa logických pravidiel. Pre hodnoty x a y potom platí:

- x and y vracajú hodnotu true len v prípade, pokiaľ x aj y sú samy osebe vyhodnotené ako true
- x or y vracajú hodnotu true len v prípade, pokiaľ je x alebo y true alebo pokiaľ sú obidve premenné true
- not x vracia hodnotu true len v prípade, pokiaľ x je vyhodnotené ako false, a vracia hodnotu false v prípade, pokiaľ je x vyhodnotené ako false

Aj pri logických operátoroch existujú dve súpravy značiek z toho dôvodu, že PERL akceptuje aj logické značky operátorov z jazykov C/C++. Prehľadne je to znázornené v tabuľke:

Štýl jazykov C/C++	Štýl jazyka PERL	Čo daný operátor znamená
&&	and	logické AND
	or	logické OR
!	not	logické NOT

Jediný rozdiel v používaní perlovského štýlu a skráteného štýlu, tak ako ho poznáme z jazykov C/C++, je ten, že skrátený „céčkarský“ štýl sa nachádza v hierarchii operátorov na vyššej (dôležitejšej) pozícii, ináč povedané, má vyššiu prioritu. Obidva štýly logického AND a OR sa vyhodnocujú skrátené, čiže pokiaľ vyhodnotenie na ľavej strane výrazu určuje celkový výsledok, pravá strana je okamžite automaticky ignorovaná. Názorne si to ukážeme na nasledujúcom príklade:

```
($x < $y) && ($y < $z)
```

Pokiaľ je ľavá strana výrazu vyhodnotená ako false, potom má automaticky celý výraz hodnotu false. To isté platí aj pre operátor ||. Pokiaľ je ľavá strana vyhodnotená ako true, celý výraz je vyhodnotený ako true a pravá strana sa automaticky preskočí, čiže už nie je vyhodnocovaná. Ukážeme si to v nasledujúcom príklade:

```
$vysledok = $a || $b || $c || $d;
```

Operátor or alebo || sa najčastejšie používa pri overovaní pripojenia k databáze, či sa nám otvoril súbor (napríklad na čítanie). Malý príklad:

```
open(FILE, 'súbor') || die 'Súbor nemôžem otvoriť.';
```

Na ľavej strane výrazu je použitá funkcia open, ktorá slúži na sprístupnenie súboru, a na pravej strane je funkcia die, ktorá slúži na zastavenie behu programu a na vypísanie chybovej správy. Pretože || patrí medzi operátory so skráteným vyhodnocovaním, bude funkcia die na pravej strane výrazu spustená len v prípade, že sa vami zvolený súbor nepodarilo otvoriť. V budúcnosti budete takúto kombináciu používať často.

Takisto existujú operátory pre logickú operáciu XOR (^ a xor), obyčajne sa však používajú len na operácie s jednotlivými bitmi čísla, ale touto problematikou sa budeme zaoberať neskôr.

Igor Lengvasky

Caché: definícia objektovej triedy

V tomto pokračovaní nášho seriálu o technológiách spoločnosti InterSystems sa pozrieme napríklad na to, akým spôsobom možno vytvoriť pomocou jazyka Class Definition Language definíciu objektovej triedy.

Základom každej databázovej aplikácie je kvalitný dátový model. Ten sa dá v prípade zložitých a rozsiahlych aplikácií vytvoriť v niektorom z vyspelých nástrojov CASE priamo spolupracujúcich s Caché. Medzi podporované nástroje patria napríklad produkty spoločnosti Rational Rose.

V jednoduchších prípadoch možno definovať zodpovedajúce triedy priamo pomocou nástroja Caché Studio. Pri návrhu definície triedy môže vývojár použiť sprievodcu jednotlivými čiastkovými úlohami, prípadne môže definíciu zapísať pomocou jazyka Class Definition Language. Zápis v tomto jazyku pre triedu osoba môže vyzeráť napríklad takto:

```

Class User.Osoba Extends %Persistent [ ClassType
= persistent, ProcedureBlock ]
{
Property DatumNarodenia As %Date;

Property Meno As %String;

```

```

Property Vek As %Integer [ Calculated ];
Property Priatelina As %String [ Collection = array ];
Property Adresa As User.Adresa;

/// vypočíta vek z dátumu narodenia
Method VekGet() As %Integer
{
// približný výpočet ...
set vek = +$h-.DatumNarodenia/365
Quit vek
}

/// Pozdraví
Method Pozdrav()
{
write "Dobrý deň!",!
quit
}
}

```

S každou triedou sú zviazané jednotlivé vlastnosti a metódy. Pod vlastnosťami sa môžu skrývať atribúty konkrétnych dátových typov, referencie a relácie. Vlastnosti teda budú vyjadrovať stav či stavy istej konkrétnej inštancie. Metódy môžu

predstavovať funkčné bloky inštancie alebo triedy (v tomto prípade môže ísť aj o uložené procedúry v jazyku SQL). Všeobecne telá metód môžu byť zapísané pomocou jazykov Caché ObjectScript alebo CachéBasic, čo sú skriptovacie jazyky určené na prácu s dátami a na realizáciu požadovanej funkčnosti.

Objektový model, ktorý používa Caché, je v porovnaní s objektovými modelmi iných programovacích jazykov rozšírený o prvky z relačného databázového prostredia. Možno teda napríklad definovať indexy, obmedzenia (constraints) a štruktúru uloženia dát. Definícia fyzickej štruktúry uloženia dát objektov je nezávislá od opisu triedy. Vývojári majú možnosť vybrať si prednastavenú štruktúru, používanú kompilátorom tried, alebo zvoliť vlastnú štruktúru a ručne ju nadefinovať s ohľadom na maximálny výkon pri dopytoch alebo transakciách a pod.

Plnohodnotné objekty v zmysle objektových technológií sú reprezentované tzv. registrovanými triedami. V databázových aplikáciách sa tieto triedy využívajú predovšetkým na zaistenie aplikačnej logiky. Vďaka objektovému prístupu môžu vývojári využiť všetky výhody podobnosti s reálnym svetom, ktorý vo svojich aplikáciách modelujú.

Nabudúce sa pozrieme na to, ako je v Caché podporovaný jazyk SQL.

Marek Kocan