Pružné zprávy

High Order Messaging je velmi zajímavá objektová technologie, která umožní podstatným způsobem zjednodušit kód a přitom zvýšit jeho flexibilitu. V tomto článku se na ni podíváme podrobněji.

Pro první přiblížení bychom mohli High Order Messaging (dále jen HOM) přeložit jako nepřímé zasílání zpráv. Za normálních okolností při objektovém programování objektu posíláme zprávu, jejímž prostřednictvím si vyžádáme nějakou akci: pošleme-li objektu, který reprezentuje pole prvků, zprávu count, objekt vrátí počet prvků; pošleme-li objektu okno zprávu close, okno se zavře. HOM může fungovat v libovolném objektovém jazyce, který podporuje polymorfismus a umožňuje, abychom se na zprávu dívali jako na objekt; v tomto článku použiji jako příklad Objective C. V něm se pro odesílání zpráv používá konstrukce [příjemce zpráva], takže výše popsané triviální příklady by v programovacím jazyce vypadaly takto:

```
int n=[pole count];
[okno close];
```

Princip HOM spočívá v tom, že zprávu nezasíláme objektu přímo; namísto toho "mezi objekt a zprávu" vložíme další příkaz, který určí, co přesně se s následující zprávou má stát. Jedním z nejjednodušších příkladů může být opožděné odeslání zprávy - dejme tomu, že chceme, aby se okno zavřelo za deset sekund. Vhodným řešením ve stylu HOM je použít speciální high-order zprávu, jež objektu řekne: "Následující zprávu máš dostat za deset sekund." Odpovídající zápis v programovacím jazyce by vypadal přibližně takto:

```
[[okno afterDelay:10] close];
```

Samozřejmě že zde nejde o nějakou speciální vlastnost objektu okno nebo zprávy close; HOM je zcela obecný systém, který díky objektové dědičnosti a díky polymorfismu může pracovat s libovolným objektem a s libovolnou zprávou - včetně zpráv s argumenty:

```
[[okno afterDelay:15] setTitle:@"Uplynulo 15 sekund"];
[[server afterDelay:30] login:@"Guest" password:@""];
```

První příklad po uplynutí patnácti sekund změní titulek okna. Ve druhém zřejmě komunikujeme s nějakým serverem, který vyžaduje přihlášení uživatele - v našem příkladu jsme si vyžádali, aby se aplikace po třiceti sekundách připojila jako uživatel Guest s prázdným heslem (jistěže HOM umožňuje také takto naplánovanou akci zrušit, pokud se v průběhu dané doby uživatel přihlásí jinak; to si ale ukážeme až později).

Služby HOM nejsou omezeny jen na opožděné odeslání; naopak, skvěle se hodí na jakékoli komplexnější operace, které v tradičním API musíme rozepisovat do složitých příkazů. Snad nejlépe se projeví při práci s poli, kde ušetří spoustu příkazů for.

HOM a pole

Chceme-li nějak zpracovat každý prvek v daném poli, musíme na to v klasickém API použít příkaz cyklu; nejčastěji jde o tradiční příkaz for. Dejme tomu, že máme více oken uložených v poli a chceme všechna zavřít; v tradiční podobě bychom museli napsat něco jako

```
// klasický příkaz cyklu
int i;
for (i=0;i<[poleOken count];i++)
[[poleOken objectAtIndex:i] close];
```

Máme-li k dispozici bohaté objektové knihovny umožňující práci s tzv. iterátory, jsme na tom trochu lépe, protože se můžeme zbavit indexování a tím i otázek typu "Je pole indexováno od nuly, nebo od jedničky?", nebo "Jsou v poli díry - indexy, na kterých není žádný prvek -, nebo ne?", či dokonce "Jsou vůbec prvky v poli řazeny sekvenčně, nebo existuje důvod je zpracovávat třeba odzadu?". V Objective C s knihovnami Cocoa by to mohlo vypadat nějak takto:

```
// cyklus s iterátorem NSEnumerator *en;
NSWindow *o;
for (en=[poleOken objectEnumerator];o=[en nextObject];)
[o close];
```

Ani iterátory zdaleka neřeší vše - pořadí zpracování prvků už může být pro různá pole různé (jinými slovy objektové, protože polymorfní), stále jsme se nezbavili příkazu cyklu. Co kdyby bylo pro některé pole výhodné provést příkazy nad všemi jeho prvky najednou, v rámci různých threadů? Takovou možnost tady nemáme, tu přinese až HOM.

S využitím HOM totiž můžeme snadno do působnosti pole přenést kompletní cyklus: stačí zavést high-order zprávu each, která poli řekne: "Následující zprávu předej beze změny každému ze svých prvků." V programovacím jazyce by to pak vypadalo asi takhle:

```
[[poleOken each] close];
```

Hlavní výhoda, kterou jsme získali, je flexibilita: teprve nyní programujeme skutečně objektově, protože máme k dispozici skutečný polymorfismus. Většina polí však bude zprávu each interpretovat klasicky, tj. tak, že následující zprávu (zde close) postupně rozešle všem svým prvkům, od prvního do posledního. Je-li k tomu ale důvod, libovolné konkrétní pole může toto chování změnit. Pole, jež dovoluje prázdné indexy, bude zprávu rozesílat pouze těm prvkům, jež v něm skutečně jsou - to by s klasickým příkazem cyklu vyžadovalo další příkaz if. Pole může procházet své prvky v libovolném pořadí, podle potřeby. Stejně dobře pole může - dává-li to pro něj smysl - namísto cyklu vytvořit pro každý ze svých prvků samostatný thread a zprávu close mu odeslat v něm, takže se de facto všechna okna zavřou najednou.

Kromě flexibility jsme tak získali i na jednoduchosti a přehlednosti - poslední příklad je mnohem přehlednější než kterýkoli z příkazů cyklu; nepotřebujeme žádné pomocné proměnné... Takhle nějak by asi objektové programování mělo vypadat. Nejde jen o jednoduchost; důležitější je, že říkáme přesně to, co chceme: "Chceme každému prvku pole poslat zprávu close." Není pravda, že chceme použít programový cyklus, to je nám přece jedno! V angličtině se tomuto nesmírně důležitému principu říká intention-revealing: měli bychom programovat tak, aby z kódu bylo jasné, čeho jsme chtěli docílit. Není naopak důležité, jakou cestou.

Pro práci s poli může HOM nabídnout více a bohatších zpráv než jen each. Dejme tomu, že původní příkaz cyklu neměl jen rozeslat nějaký příkaz, ale že měl naopak od každého z prvků pole něco zjistit a vytvořit seznam těchto informací - třeba chceme titulky všech oken z pole:

```
// klasický příkaz cyklu NSMutableArray *titulky=[NSMutableArray array];
int i;
for (i=0;i<[poleOken count];i++)
[titulky addObject:[[poleOken objectAtIndex:i] title]];
```

V cyklu každému oknu v poli pošleme zprávu title, na niž okno reaguje vrácením svého titulku; ty hromadíme v poli titulky. HOM nám ale může nabídnout jednodušší a přehlednější variantu, založenou na high-order zprávě collect. Ta funguje podobně jako each, navíc však hodnoty vrácené všemi prvky soustředí do nového pole, jež se stane její návratovou hodnotou:

```
NSArray *titulky=[[poleOken collect] title];
```

Zprávy HOM můžeme řetězit jako každé jiné. Chtěli bychom, aby získané titulky oken byly velkými písmeny? Nic snazšího - na získané pole titulků přece high-order zpráva collect funguje stejně dobře jako na původní pole oken:

```
NSArray *titulky=[[[[poleOken collect] title] collect] uppercaseString];
```

Další možnosti, které dává HOM nad poli, jsou zprávy select a reject, jež vybírají prvky z pole v závislosti na tom, zda na nějakou jinou zprávu prvek reaguje kladně či záporně. Zpráva select tedy říká: "Následující zprávu pošli každému ze svých prvků; vrať v poli ty, jež na zprávu reagovaly hodnotou YES." Podobně je to se zprávou reject, která ovšem sbírá objekty, jež vrátily NO. Následující příkaz například vybere jen viditelná okna:

NSArray *vo=[[poleOken select] isVisible];

Složitější je to v případě, kdy bychom chtěli vybírat podle složitějšího kritéria. Dejme tomu, že chceme vybrat všechna okna, jejichž titulek začíná písmenem H. Máme k dispozici již známou zprávu title i zprávu hasPrefix:, jež ověří, zda textový řetězec má daný prefix; jednoduše je složit za sebe ovšem nemůžeme:

NSArray *xo=[[[poleOken select] title] hasPrefix:@"H"]; // špatně

Toto samozřejmě fungovat nebude: high-order zpráva select spolu s následující zprávou title prostě vybere všechna okna, jež mají nějaký titulek (tj. zašleme-li jim zprávu title, dostaneme nenulovou hodnotu, což je v jazyce Objective C "pravda"). Poli obsahujícímu seznam těchto oken bychom zaslali zprávu hasPrefix: - a to ovšem nechceme.

Jednou z možností by bylo využít jinou high-order zprávu, jež se pro daný problém hodí lépe. V Objective C s knihovnami Cocoa, jež umožňují k atributům objektů přistupovat podle jména, by to mohlo vypadat třeba takto:

NSArray *xo=[[poleOken selectWhereValueForKey:@"title"] hasPrefix:@"H"];

kde high-order zpráva selectWhereValueForKey: spustí poměrně komplikovaný mechanismus, který by se dal popsat slovy: "Zjisti od každého ze svých prvků atribut se zadaným jménem" - zde "title" "a hodnotě tohoto atributu pošli následující zprávu. Vrať v poli ty prvky, jejichž atribut na zprávu reagoval hodnotou YES."

Ačkoli konkrétní implementace HOM může obsahovat ještě dlouhou řadu dalších high-order zpráv řešících většinu běžných problémů, někdy to přece jen nestačí a je zapotřebí zprávy tak či onak řetězit. Ukažme si, jak na to.

Řetězení high-order zpráv

Připomeňme si příklad z minulého odstavce, v němž jsme v jediném výrazu použili zřetězení čtyř různých zpráv; dvě z nich byly highorder, dvě normální:

NSArray *titulky=[[[[poleOken collect] title] collect] uppercaseString];

To samozřejmě není žádný problém. Vzhledem k tomu, že zprávy se střídají, rozpadá se vlastně výraz na dvě části, z nichž v každé je použita pouze jedna high-order zpráva. První zpráva collect spolu se zprávou title vrátí normální pole objektů, nad kterým pracuje druhá zpráva collect se zprávou uppercaseString.

Řetězení zpráv je možné i jinak - existuje řada situací, kdy je žádoucí řetězit high-order zprávy bezprostředně za sebe. Připomeňme si první dvě zprávy systému HOM, s nimiž jsme se seznámili: afterDelay: a each - co když máme pole oken a chceme je všechna zavřít za minutu? Na první pohled se nabízí následující řešení:

[[[poleOken afterDelay:60] each] close]; [[[poleOken each] afterDelay:60] close];

Obě varianty jsou prakticky rovnocenné, první je malinko efektivnější - znamená "Za minutu pošli všem prvkům poleOken zprávu close", takže v implementaci stačí jediný časovač. Druhá varianta říká "Každému prvku poleOken pošli za minutu zprávu close", takže časovačů je zapotřebí tolik, kolik je v poli oken.

Ovšem... je to vůbec správné? Můžeme high-order zprávy tímto způsobem řetězit? V nejjednodušším případě ne - tak, jak jsme si systém HOM dosud popsali, by to nefungovalo. Například v prvém řádku by high-order zpráva afterDelay: zajistila, že za minutu poleOken dostane zprávu each. Nikde však není psáno, že by se odeslání zprávy close také o minutu odsunulo, takže zprávě each by vlastně chyběl "argument" (tj. ta zpráva, jíž je třeba rozeslat všem prvkům pole).

Naštěstí existuje velmi jednoduché rozšíření pravidel, podle nichž se high-order zprávy zpracovávají. Stačí do souboru "pravidel HOM" přidat podmínku, že jakákoli skupina bezprostředně po sobě následujících high-order zpráv se vždy zpracovává jako jediný celek. Je-li tomu tak, oba výše uvedené příklady budou bez problémů fungovat: zprávy afterDelay: a each spolu vlastně vytvoří jednu složenou high-order zprávu, jejímž "argumentem" je následující zpráva close.

Podobným způsobem můžeme sestavovat i složitější řetězy highorder zpráv, včetně poněkud absurdního, ale plně funkčního

```
[[[okno afterDelay:1] afterDelay:2] close];
```

Stojí za to si uvědomit zcela zásadní rozdíl mezi tímto řetězením, jež je zcela v pořádku, a nekorektním případem z minulého odstavce. Pokud bychom zapsali

```
NSArray *xo=[[[poleOken select] title] hasPrefix:@"H"]; // špatně
```

k žádnému spojení zpráv do jediného celku nedojde, protože zde používáme jen jedinou high-order zprávu select; zbývající dvě zprávy jsou standardní, a proto se nespojí. To je jasné, ale... Bylo by přece jen šikovné, kdybychom je v případě potřeby takto spojit mohli v principu to přece musí jít; můžeme-li vzájemně spojovat high-order zprávy, není důvod, proč by to nemělo jít i s ostatními.

Ukazuje se, že tomu skutečně tak je. Dobrá implementace HOM může nabízet speciální high-order zprávy begin a end, jejichž jediným účelem je označit blok zpráv, jež se mají spojit a provést jako jeden celek, vlastně jako jedna jediná složená zpráva. Máme-li tyto zprávy k dispozici, můžeme minulý příklad přepsat korektním způsobem:

```
NSArray *xo=[[[[[poleOken select] begin] title] hasPrefix:@"H"] end];
```

Tentokrát bude vše fungovat bez nejmenších obtíží. High-order zpráva select narazí na zprávu begin, která říká: "Ber vše, co za mnou následuje, až po zprávu end, jako jediný celek." Proto zpráva select pošle každému prvku z pole obě zprávy - title i hasPrefix: - a až podle výsledku druhé z nich rozhodne, zda okno zařadí do výsledného pole, nebo ne.

Přístup k řídicímu objektu

Samozřejmě že "skromně v pozadí" implementace HOM stojí řada pomocných tříd a objektů, jež zajišťují všechny odpovídající služby. My se na ně podrobněji podíváme později, až si budeme vysvětlovat princip, na němž je implementace HOM založena; prozatím se soustředíme na základní programátorské rozhraní.

V některých případech je však vhodné zajistit přístup k řídicímu objektu i na této úrovni. Jde o high-order zprávy typu afterDelay: nebo inNewThread (viz níže). V jejich případě totiž může být někdy žádoucí se jaksi vrátit k odeslání zprávy a nějak jej dodatečně modifikovat - například zrušit požadavek na opožděné odeslání zprávy dříve, než k němu došlo, nebo počkat na ukončení threadu. Naštěstí systém HOM nějaký řídicí objekt stejně obsahuje, takže je to velice snadné. Stačí použít proměnnou, do které uložíme výsledek odeslání patřičné high-order zprávy. Připomeňme jeden z prvních příkladů, jímž bylo automatické přihlášení k serveru po uplynutí zadaného času; zde bychom samozřejmě v praxi chtěli zrušit automatické přihlášení tehdy, když se uživatel v daném časovém intervalu přihlásí sám. Celý kód samozřejmě bez grafického uživatelského rozhraní - by pak mohl vypadat asi nějak takto:

```
id autLogin; // tato metoda se volá na začátku (zajistí GUI) -(void)prepareLogin {
[autLogin=[server afterDelay:30] login:@"Guest" password:@""];
}
// tato metoda se volá pokud se uživatel přihlásí (zajistí GUI) -(void)loginWithName:name
password:password {
[autLogin cancel]; // zruší naplánované automatické přihlášení [server login:name
password:password];
}
```

Máme-li k dispozici přístup k řídicímu objektu, můžeme pohodlně zavést další high-order zprávu spojenou s časem. Zpráva repeatWithDelay: říká: "Následující zprávu pošli každých N sekund." Chceme-li pravidelné zasílání zprávy ukončit, využijeme opět řídicí objekt a zprávu cancel.

Služby systému HOM nejsou ani zdaleka omezeny jen na to, co jsme si dosud ukázali - značné množství nejrůznějších tradičních programových struktur je možné nahradit high-order zprávami. Ukažme si několik dalších typických příkladů využití HOM.

Hlídání přístupových práv

Ve složitějších programových systémech často platí, že řada akcí (reprezentovaných v objektovém prostředí zprávami) je povolena pouze některým uživatelům, kteří disponují patřičnými právy. V tradičním programování bychom museli pokaždé použít příkaz if; elegantnější ale je připravit high-order zprávu ifAllowedForCurrentUser, a kteroukoli z "citlivých" zpráv posílat jejím prostřednictvím:

```
id account=[[database ifAllowedForCurrentUser] accountForName:client];
[[database ifAllowedForCurrentUser] removeClient:client];
```

Zpráva ifAllowedForCurrentUser zjistí, který uživatel je momentálně přihlášen, a ověří, zda je pro něj požadovaná akce (v našem příkladu získání nebo smazání údajů o klientovi) přípustná. Pokud ano, zprávu předá; ne-li, generuje výjimku.

Ošetření výjimek

I pro zpracování výjimek se mohou občas hodit služby HOM. Poměrně často narazíme na situaci, v níž výjimku bezprostředně nezpracováváme - jen ji uložíme, aby její uloženou hodnotu mohly využít rutiny grafického uživatelského rozhraní ve chvíli, kdy budou uživateli hlásit, že došlo k nějaké chybě, a proto jím vyžádaná akce nebyla provedena. V tradičním programování obvykle odpovídající kód vypadá nějak takto:

```
lastError=nil;
NS_DURING // začátek chráněného bloku [sqlServer commit];
NS_HANDLER // došlo k výjimce lastError=localException;
NS_ENDHANDLER
```

Jestliže tedy při zpracování commitu došlo k výjimce, vyvolá se chybový kód, který tuto výjimku uloží do proměnné lastError.

Grafické uživatelské rozhraní jen zkontroluje obsah této proměnné. Je-li nulový, akce byla provedena; jinak ohlásí uživateli chybu, jejíž příčinou je lastError.

Máme-li k dispozici HOM, můžeme tento příklad přepsat daleko pohodlněji:

```
lastError=[[sqlServer safely] commit];
```

High-order zpráva safely totiž říká: "Následující zprávu pošli uvnitř chráněného bloku; dojde-li přitom k výjimce, vrať ji. Pokud zpráva proběhne bez problémů, vrať nil."

Paralelní zpracování

O paralelním zpracování už jsme se zmínili s tím, že zprávy typu each nebo collect mohou v rámci polymorfismu pro některá pole (a některé zprávy) rozesílat zprávy v různých threadech. HOM ovšem nabízí řadu dalších možností. Nejjednodušší jsou varianty již známých zpráv, jež paralelní zpracování v každém případě vynutí:

```
[[poleOken concurrentEach] close];
NSArray *titulky=[[poleOken concurrentCollect] title];
NSArray *vo=[[poleOken concurrentSelect] isVisible];
```

Tyto varianty high-order zpráv vždy spustí tolik threadů, kolik je v poli objektů, a v každém z nich pošlou jednu zprávu - ty jsou tedy všechny zpracovány najednou. Další velmi šikovná high-order zpráva umožní přímo spustit další thread:

```
[[sheet inNewThread] recalculate];
```

Předpokládáme-li, že sheet reprezentuje tabulku v spreadsheetu, je rozumné vyžádat si její přepočítání v samostatném threadu - a to je přesně to, co high-order zpráva inNewThread zajistí.

Chceme-li s hlavním threadem počkat, než skončí zpracování vedlejšího threadu, můžeme použít přístup k řídicímu objektu velmi podobným způsobem, jakým jsme si to ukázali s časovačem:

```
[thread=[sheet inNewThread] recalculate];
...
[thread wait];
```

Jde-li nám ovšem o získání výsledku, na němž samostatný thread pracuje, nabízí objektové prostředí a HOM mnohem elegantnější řešení. Je totiž možné zařídit, aby high-order služba vrátila tzv. zástupný objekt. Ten nedělá nic jiného, než že udržuje vazbu na thread a čeká, až mu kdokoli pošle jakoukoli zprávu. Jakmile se to stane, zástupný objekt nejprve vyvolá službu wait (počká, až thread svou

práci dokončí), pak nahradí sám sebe výsledkem threadu - a nakonec ještě tomuto skutečnému výsledku pošle zprávu, již sám dostal! (Vidíme, že vzhledem k nepřímému odeslání zprávy vlastně jde opět o další službu HOM.) Na první přečtení to zní složitě, ale v programovacím jazyce s využitím HOM je to nesmírně jednoduché:

```
NSArray *seznam=[[database future] complicatedSearch];
...
printf("%d prvků",[seznam count]);
```

High-order služba future spustí nový thread a v něm předá objektu database zprávu complicatedSearch stejně, jako by to udělala high-order zpráva inNewThread. Zpráva future však navíc vytvoří speciální zástupný HOM objekt a ten vrátí místo požadovaného pole.

V době, kdy běží libovolný kód označený v našem příkladu třemi tečkami, zároveň probíhá "komplikované vyhledávání" v databázi, a proměnná seznam neobsahuje pole (ale zástupný objekt). Jakmile ovšem pošleme zástupnému objektu první zprávu - v našem případě zprávu count -, vše se změní. Systém HOM se postará o to, aby hlavní thread nejprve počkal, než prohledávání databáze skončí; pak se seznam nahradí skutečným výsledkem (tedy polem nalezených údajů) a ten dostane zprávu count. Příkaz printf tedy skutečně vypíše správný počet nalezených údajů.

Je téměř zbytečné dodávat, že pokud náhodou thread, v němž probíhá služba complicatedSearch, skončí dříve než kód označený třemi tečkami, systém HOM se postará o to, aby se zástupný objekt seznam ihned nahradil skutečným výsledkem. Až pak dojde na zprávu count v příkazu printf, bude již seznam obsahovat skutečné výsledné pole a vše bude v pořádku.

Posílání zpráv tomu, kdo jim rozumí

V objektovém prostředí je nesmírně výhodným a velmi často užívaným trikem odeslání zprávy nějakému objektu jen tehdy, když objekt dané zprávě rozumí.

Vzhledem k tomu, že uživatelé jazyků typu C++ (a do jisté míry i Javy), jež nabízejí jen nedokonalou podporu objektového programování, nemají s tímto systémem zkušenosti, stojí za to uvést několik příkladů:

Máme okno a jeho řídicí objekt. Okno řídicí objekt informuje o změnách velikosti, pokud řídicí objekt rozumí odpovídající zprávě.

Máme tabulku a objekt, jenž dodává data, která tabulka zobrazuje.

Pokud řídicí objekt rozumí zprávě pro změnu dat v dané řádce a daném sloupci, tabulka tuto změnu automaticky umožní - jestliže tomu tak není, tabulka bude "read-only".

Máme pole objektů reprezentujících aktivní prvky uživatelského rozhraní. Uživatel si vyžádal službu Copy. Vyhledáme proto v poli objekt, který rozumí zprávě copy, a pošleme mu ji. Tak je automaticky zajištěno, že kopírujeme skutečně obsah textového pole, a ne třeba tlačítka.

Ve všech příkladech jde vždy o to samé - na místě řídicího objektu může stát zcela libovolný objekt, v jehož rámci implementujeme zpracování jen těch zpráv, které pro nás mají v dané aplikaci smysl.

Při rozhodování, které zprávy jakému objektu poslat, nám HOM opět pomůže uspořit nějaké "ify" a "fory" proti tradičnímu programování:

```
[[delegate ifPossible] resizedFrom:oldSize to:newSize];
[[views firstThatResponds] copy];
```

Příklady jen ilustrují výše uvedený text. V prvním posíláme objektu delegate (řídicí objekt) zprávu resizedFrom:to:, která jej informuje o změně velikosti - ovšem pouze v případě, že řídicí objekt této zprávě rozumí; není-li tomu tak, jde o prázdnou akci. Ve druhém případě vyhledáme v poli views (jež obsahuje prvky uživatelského rozhraní) první objekt, jenž rozumí zprávě copy, a zprávu mu pošleme.

To je zatím vše...

Do dnešního článku se již více informací o systému HOM nevejde. Bude-li však k tomu příležitost, řekneme si v budoucnosti o této nesmírně zajímavé technologii více. Vysvětlíme si princip, na kterém HOM funguje, ukážeme si příklad jeho konkrétní implementace a praktické příklady jeho použití.

Ondřej Čada

Umíte-li anglicky, můžete více informací o HOM včetně zdrojových textů najít také na adrese http://www.metaobject.com.