Druhá zpráva o HOM

High Order Messaging (dále jen HOM) je objektová technologie, která umožní podstatným způsobem zjednodušit kód a zvýšit jeho flexibilitu. Před časem jsme se seznámili s jejími základy (viz Chip 12/02 nebo www.ocs.cz/Text/HOM.html). Dnes se seznámíme se základními principy, na nichž je technologie HOM založena.

HOM bychom mohli volně přeložit jako "nepřímé zasílání zpráv". Při objektovém programování objektu běžně posíláme zprávy, jejichž prostřednictvím si vyžádáme nějaké akce (Java využívá jiné terminologie - místo zasílání zpráv hovoří o volání metod; my však budeme využívat standardní objektové terminologie a budeme mluvit o zprávách).

ZÁKLADY

Princip HOM spočívá v tom, že zprávu nezasíláme přímo. Místo toho "mezi objekt a zprávu" vložíme další příkaz, který určí, co se se zprávou má stát. Jedním z nejjednodušších příkladů může být opožděné odeslání zprávy: například chceme, aby se okno zavřelo za deset sekund. Řešením HOM je použít speciální "high-order" zprávu, jež objektu řekne: "Následující zprávu máš dostat za deset sekund." Zápis v Objective C by vypadal takto:

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

Možností využití služeb HOM je řada. Velmi vhodné jsou pro práci s poli, kde ušetří spoustu příkazů for: stačí zavést HOM zprávu each, která poli řekne: "Následující zprávu předej každému ze svých prvků." Takto třeba zavřeme všechna okna v poli:

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

A takto uložíme do samostatného pole jejich titulky nebo vybereme jen ta okna, jež jsou právě viditelná:

```
NSArray *titulky=[[poleOken collect] title]; NSArray *vo=[[poleOken select] isVisible];
```

Principem funkce HOM je využití tzv. zástupného objektu. Vraťme se k příkladu se zavíráním oken. Zajistíme, aby libovolný objekt po přijetí zprávy afterDelay: vytvořil a vrátil nový objekt třídy řekněme DelayedPerformer. Ten bude obsahovat pouze odkaz na původního příjemce zprávy a v našem případě také časový interval, určující dobu, na kterou se má zpracování zprávy odložit. Objekt třídy Delayed Performer přímo nezpracovává žádnou zprávu. Místo toho ve chvíli, kdy jakoukoli zprávu dostane, udělá dvě věci (viz obrázek):

zprávu i její případné argumenty si zapamatuje (nyní tedy objekt obsahuje trojici údajů: původního příjemce zprávy, časový interval a zprávu);

spustí časovač (s dříve uloženým časovým intervalem), po jehož uplynutí předá uloženému objektu uloženou zprávu. U polí zpráva each vytvoří a vrátí nový zástupný objekt, třeba třídy Trampoline, který obsahuje jen odkaz na pole, jež bylo původním příjemcem zprávy. Jakmile objekt třídy Trampoline dostane nějakou zprávu, rozešle ji všem objektům z pole, na něž má v sobě odkaz.

Složitější je to u zpráv, jež vracejí nějakou hodnotu - collect, select apod. Zpráva collect nejenže rozešle zprávu všem objektům z pole, ale navíc výsledek každého odeslání uloží do nového pole, jež pak vrátí. Zpráva select prozkoumá výsledek odeslání zprávy a do nového pole uloží ty z původních objektů, pro něž zpráva vrátila pravdivou či nenulovou hodnotu. Obdobně lze konstruovat i další, složitější HOM zprávy.

Ani řetězení zpráv není žádný technický problém: nejjednodušší je, když zástupné objekty místo jedné zprávy (již si pamatují pro odeslání cílovému objektu) mají pro zprávy celé pole. Do něj se zprávy ukládají a při vlastním zpracování se odešlou všechny "najednou" (první zpráva z pole se pošle cílovému objektu, výsledku se ihned pošle zpráva druhá, výsledku se pošle třetí atd.).

Potom je jednoduchá i implementace speciálních zpráv begin a end, jež umožňují řetězení zpráv (i těch, jež s HOM nemají nic společného). Zpráva begin v zástupném objektu zvedne pomocný čítač,

zpráva end jej opět sníží. Zástupný objekt využívá čítač při rozhodování, zda zprávy ukládat, nebo rovnou zpracovat: jde-li o HOM zprávu nebo je-li čítač nenulový, zpráva se uloží.

Implementace služeb popsaných v minulém textu o službách HOM už by měla být vcelku zřejmá: třeba zpráva safely prostě vytvoří zástupný objekt, který svému cílovému objektu pošle zprávu v rámci zabezpečeného výjimkového bloku (try... pro Javu, NS_DURING... NS_HANDLER pro Objective C). Zpráva inNewThread vytvoří zástupný objekt, který svému cílovému objektu pošle zprávu v samostatném threadu.

Speciální technika však je zapotřebí pro implementaci automatické synchronizace mezi thready na základě vrácené hodnoty. Připomeňme si, o co jde: služba, jež spouští nový thread, může vrátit tzv. zástupný objekt fault. Ten pouze udržuje vazbu na thread a čeká, až mu kdokoli pošle jakoukoli zprávu. Potom zástupný objekt vyvolá službu wait (počká, až thread skončí), nahradí sám sebe skutečným výsledkem threadu a nakonec ještě tomuto výsledku pošle zprávu, již sám dostal. V programovacím jazyce s využitím HOM je to jednoduché:

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

Služba future spustí nový thread a v něm předá objektu database zprávu complicated Search stejně, jako by to udělala zpráva inNewThread. Zpráva future však navíc vytvoří speciální zástupný HOM objekt řekněme třídy ArrayFault a ten vrátí místo požadovaného pole. Objekt ArrayFault obsahuje odkaz na thread, v němž probíhá zpracování; thread zase obsahuje odkaz na ArrayFault. Jestliže objekt ArrayFault dostane libovolnou zprávu, zapamatuje si ji a využije systémových služeb pro řízení threadů k tomu, aby počkal na ukončení threadu, na nějž je vázán. Pak zprávu pošle sám sobě. (Zní to divně? Čtěte dál!)

Jakmile thread skončí, poslední instrukce threadu uloží výsledek do paměti na místo objektu ArrayFault (objekt ArrayFault zanikne, neboť jeho data jsou přepsána daty objektu, jenž je výsledkem práce threadu). To znamená, že pokud zástupný objekt ArrayFault poslal zprávu "sám sobě" (tj. na svou vlastní adresu), ve skutečnosti ji poslal objektu, jenž je výsledkem threadu. To proto, že ukončení threadu objekt ArrayFault v paměti tímto výsledkem nahradilo.

JAZYKOVÁ PODPORA

Jestliže mají služby HOM fungovat s libovolným příjemcem zpráv, je třeba, aby zprávy typu afterDelay či each byly k dispozici pro libovolný objekt. To znamená, že chceme-li implementovat HOM, musíme mít možnost přidávat nové metody ke kořenové třídě (aby byly k dispozici kterémukoli objektu kterékoli třídy). V dobře navržených objektových jazycích to není problém (Objective C k tomu má tzv. kategorie). V jazycích typu Javy, jež kategorie ani obdobnou službu nemají, to však znamená mít možnost zasáhnout přímo do zdrojových kódů kořenové třídy.

Nutností je, aby objektový programovací jazyk podporoval polymorfismus, jinými slovy, aby bylo možné objektu zaslat libovolnou zprávu s tím, že konkrétní zpracování této zprávy závisí na objektu samotném. To je důsledek toho, že zástupný objekt (Delayed Performer) musí být schopen přijmout jakoukoli zprávu, neboť ji sám nezpracovává.

U jazyků typu Javy nebo C++ zde opět narazíme - polymorfismus podporují pouze zčásti, jen v rámci dědické hierarchie tříd (a v Javě také rozhraní). Vytvořit objekt, jenž přijímá libovolnou zprávu, v nich možné není (v Javě to možné je, pokud využijeme služby java.lang. reflect; problém však je v tom, že nemůžeme využívat standardní syntaxi pro předávání zpráv a zápis programu se komplikuje).

Další podmínkou pro implementaci HOM je, aby bylo možné zprávu vzít a jako objekt uložit do proměnné (a odeslat později). Zde jde spíše o služby standardních knihoven; v Javě lze opět za cenu nepohodlí využít služby java.lang.reflect. Standardní knihovny Objective C obdobné služby přímo nepodporují. Objective C se však se standardními knihovnami už téměř nevyužívá, běžně se používají bohatší knihovny založené na standardu OpenStep (např. Cocoa). Tam je k dispozici třída NSInvocation, určená k ukládání zpráv a jejich argumentů.

Poslední "trik", který musí programovací jazyk pro implementaci HOM nabízet, je zpracování obecné zprávy. To úzce souvisí s polymorfismem i s ukládáním zpráv: má-li objekt třídy DelayedPerformer být schopen přijmout (a uložit a později předat) jakoukoli zprávu, musíme mít v jazyce nebo ve standardních knihovnách podporu pro implementaci "neznámé zprávy".

V Objective C (s knihovnami a runtime založenými na standardu OpenStep) taková podpora je. Standardně tam platí, že pokud objekt dostane zprávu, pro niž není v jeho třídě ani v žádné z rodičovských tříd definována metoda, runtime zprávu i její argumenty automaticky "zabalí" do objektu třídy NSInvocation a ten cílovému objektu předá jako argument zprávy forwardInvocation:.

Žádná ze současných verzí Javy nic podobného standardně neobsahuje. Jelikož ale Java přímo nepodporuje ani polymorfismus, musíme se při implementaci HOM stejně obrátit na vhodný preprocesor a v rámci jeho maker implementovat vlastní konvenci obdobnou zprávě forwardInvocation: z Objective C.

Zástupný objekt typu fault je nahrazen jiným objektem, takže moduly, jež s objektem pracují, vůbec "nepostřehnou změnu". To vyžaduje zásah do objektového systému na poměrně nízké úrovni. V Objective C je to možné díky jeho "céčkovému" dědictví a díky tomu, že objektová struktura je veřejně zdokumentována (objekty obsahují odkaz na svou třídu a odkaz je na nízké úrovni přístupný). Bohužel, v Javě, pokud mi je známo, nic podobného není možné.

TO JE ZATÍM VŠE...

Do tohoto článku se už více informací o HOM nevejde. Příště si však ukážeme konkrétní implementaci HOM a seznámíme se i s některými praktickými příklady použití. Více informací o HOM včetně zdrojových textů (anglicky) lze najít na adrese www.metaobject.com.