Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 19.02. 2001
Url: http://www.builder.cz/art/cpp/cpp_virtual.html

Časná versus pozdní vazba - úvod do polymorfismu v C++

Všechny metody, které jsem ve svých článcích doposud používal byly volány tak zvanou časnou vazbou. Tedy překladač v době překladu přesně věděl jaký podprogram (metoda) bude kdy vyvolán. Například v mém minulém článku "Jednoduchá dědičnost v C++" jsem v jednom ukázkovém programu uvedl řádek `v1->nastavKola(4);`. Pro překladač je jednoznačné, který podprogram se má vyvolat. Metoda `Vozidlo::nastavKola(int)` bude vyvolána i když v1 bude instance třídy Vozidlo, nebo instance třídy Nákladní vozidlo. V takovém případě říkáme, že metoda je volána časnou vazbou. Pojem "časná vazba" asi proto, že překladač zná adresu podprogramu na který má předat řízení včas (V době překladu.) Může ale nastat případ (nastává velmi často), kdy budeme potřebovat, aby metoda pro podtřídu vykonávala jinou činnost, než stejná metoda u nadtřídy. Tedy vlastně budeme potřebovat přepsat tělo metody v podtřídě. Jednoduše metodu přepsat sice jde, ale není to dobré. Uvedu příklad: (Metody `void casna()` jsou volány časnou vazbou.)

```
#include <iostream.h>

class Nadtrida
{
 public:
   void casna() { cout << "Metoda tridy Nadtrida volana casnou vazbou" << endl; }
};

class Podtrida : public Nadtrida
{
 public:
   void casna() { cout << "Metoda tridy Podtrida volana casnou vazbou" << endl; }
};

int main(void)
{
    Nadtrida *n = new Nadtrida;
    Podtrida *p = new Podtrida;
    n->casna();
    p->casna();
/* Zatím vypadá vše OK. Ale zkusme dál. */
    Nadtrida *problem = new Podtrida; /* Na místo předka dám potomka - to je OK */
    problem->casna(); /* A tady je problém! */
    return 0;
```

```
}
```

Po spuštění zjistíte, že řádek `problem->casna();` vyvolá metodu `void Nadtrida::casna()` což jsme nechtěli, protože ukazatel problém je sice ukazatel typu Nadtřída, ale ve skutečnosti ukazuje na instanci třídy Podtřída. Na místo, kde byl očekáván předek byl dosazen potomek - v OOP běžná a často používaná konstrukce.

Podívejme se, jak překladač postupoval při překladu řádku `problem->casna();` . Nejprve zjistil typ ukazatele problém. Ukazatel je deklarován jako `Nadtrida *problem`. Zjistil si, jestli třída Nadtřída (Případně některý její předek - v našem případě žádný není.) má definovanou metodu `void casna();`. Protože má, rozhodl již v době překladu o tom, že se bude volat `void Nadtrida::casna();`. Kdyby neměla, nahlásil by chybu. Fakt, že ukazatel problém může ukazovat na potomka nyní překladač nezajímalo. Proto abychom na řádku `problem->casna();` volali metodu `void casna()` podle toho, na jakou instanci ukazatel problém ukazuje, musíme použít pozdní vazbu.

Klíčové slovo `virtual`

Klíčové slovo `virtual` před deklarací metody překladači přikazuje použít tak zvanou pozdní vazbu při volání dané metody. Zkuste do mého příkladu mezi veřejné metody třídy Nadtřída vepsat řádek `virtual void pozdni();` a mezi veřejné metody třídy Podtřída vepište řádek `virtual void pozdni();`. Potom do zdrojového textu vepište těla těchto metod:

```
void Nadtrida::pozdni() /* Tady se už virtual nepíše, jen v deklaraci. */
{
   cout << "Metoda tridy Nadtrida volana pozdni vazbou" << endl;
}
void Podtrida::pozdni()
{
   cout << "Metoda tridy Podtrida volana pozdni vazbou" << endl;
}
```

A na konec funkce main (ale samozřejmě před řádek `return 0;`)dopište řádky:

```
   n->pozdni();
   p->pozdni();
   problem->pozdni(); /* Je skutecne volana void Podtrida::pozdni() */
```

O tom která metoda (`void Nadtrida::pozdni();` , nebo `void Podtrida::pozdni();`) ve skutečnosti bude volána se rozhoduje až při běhu programu podle toho, na jakou instanci je volána, ne v době kompilace.

Jak je možné, že to funguje?

Jak jsem ve svých předchozích článcích naznačil metody s časnou vazbou jsou překládány jako "obyčejné" céčkovské funkce, kde je přidán jako 1. parametr `this` - implicitní parametr, který ukazuje na instanci, pro kterou je metoda vyvolána. Rozdíl mezi "obyčejnou" funkcí a metodou volanou časnou vazbou je jen v tom, že překladač u metody kontroluje, zda je volána skutečně na správný objekt. Metody volané pozdní vazbou (virtuální metody) se překládají trochu jinak. Každá instance, která má alespoň jednu virtuální metodu má v sobě navíc ukazatel na tak zvanou tabulku virtuálních metod (TVM, někdy jsem také viděl zkratku VMT). Tento ukazatel je pro programátora nepřístupný (Alespoň ne korektní cestou.) a ukazuje na tabulku, ve které jsou uloženy adresy virtuálních metod. O tom, že v instanci je jeden ukazatel navíc se můžete lehce přesvědčit pomocí `sizeof`. Řádek `problem->casna();` bude přeložen ne jako jednoduché zavolání funkce, ale jako vyvolání 1. virtuální metody (metody, jejíž adresa je ve TVM na 1. místě) která je ve TVM pro instanci na kterou ukazatel problém ukazuje. O správnou inicializaci ukazatele na TVM se postará překladač, který při vytváření každé instance (před zavoláním samotného konstruktoru) přidá kód pro inicializaci ukazatele na TVM. Programátor, který chce používat virtuální metody nemusí vlastně vůbec pojmy jako TVM znát, prostě stačí, že to funguje jak má. Uvedl jsem je jen proto, aby byly jasné některé omezení:

- Metoda označená jako `virtual` nemůže být současně označená jako `inline`. Což je vlastně logické uvědomíte-li si význam slova `inline` a mechanismus pozdní vazby pomocí TVM.
- Konstruktor nemůže být virtuální. Volat konstruktor pozdní vazbou nemá žádný smysl a vlastně ani není jasné, jak by se měl lišit od konstruktoru volaného časnou vazbou. Vytvářím-li instanci, musí mě být jasné jaké třídy bude. Ukazatel na TVM je inicializován před vyvoláním konstruktoru, tedy v těle konstruktoru lze použít virtuální metody instance, kterou vytvářím.
- Instance musí být korektně inicializována. Jednou jsem napsal, že pro dynamickou inicializaci instancí máte používat operátor `new` a ne nějaké céčkovské funkce `malloc` a podobné. Dosud v případě že instance měla implicitní konstruktor to nevadilo. Nyní to již vadí i v tomto případě. Funkce `malloc` nevolá konstruktor a ani neinicializuje ukazatel na TVM, takže vyvolání jakékoliv virtuální metody se bude chovat hodně , ale opravdu hodně zvláštně. Ani Vám nedoporučuji to zkoušet, když tak jen na Vaší vlastní zodpovědnost. Funkce `malloc` a podobné patří do jazyka C, do C++ patří operátor `new`.

Konstruktor virtuální být nemůže, ale destruktor virtuální být může a dokonce by i měl být. Alespoň v případě, že třída má jiné virtuální metody. V mém předchozím článku jsem upozornil na jedno zvláštní vyvolání "nesprávného" destruktoru. Problém byl v tom, že destruktor byl překládán časnou vazbou.

Někteří uživatelé Borland C++ Builderu si mohou všimnout jedné zajímavosti, která jako-by vyvrací mé tvrzení. Například třída TForm z knihovny VCL má konstruktor deklarovaný s klíčovým slovem `virtual`. Nejedná se v žádném případě o konstruktor volaný pozdní vazbou. Konstruktor volaný pozdní vazbou není ani v ANSI C++, ani nemůže být žádným rozšířením jazyka C++. Klíčové slovo virtual u konstruktorů některých tříd z VCL má jiný význam než pozdní vazba, a nemá s ANSI C++ nic společného.

Tolik pro vysvětlení rozdílu mezi časnou a pozdní vazbou, příště dokončím téma polymorfismu. Poté v dalším článku se vrátím zpět k dědičnosti, tentokráte vícenásobné, což je téma na 2 - 3 články.

------ http://www.builder.cz ------