Builder.cz - http://www.builder.cz
Autor: Radim Dostál
Rubrika: C/C++
Datum vydání: 13.07. 2001
Url: http://www.builder.cz/art/cpp/cpp_dynidentif.html

Dynamická identifikace typů v C++

Dynamická identifikace typů v C++

Pod pojmem "Dynamická identifikace typů" rozumíme zjišťování typů proměnných, nebo objektů v době běhu programu. Identifikaci typů zajišťuje operátor `typeid`. Než se ale budeme zabývat tímto operátorem, podívejme se nejprve na třídu `type_info`. Instance této třídy slouží k uchování informace o typech.

Třída type_info

Třída je deklarována v hlavičkovém souboru `typeinfo`. Deklarace se nachází v prostoru jmen `std`. Instance této třídy v sobě mají informace o typech v době běhu programu. Třída má dvě veřejné metody: `const char *name() const` a `bool before(const type_info& arg) const`. Obě metody jsou konstantní, tedy nijak nemění vnitřní stav (hodnoty atributů) objektu. Metoda `name` vrací konstantní řetězec udávající název typu. Setkal jsem se s takovými překladači, které vytvářely programy, ve kterých metoda `name` sice vracela název typu, ale nikoliv "pěkně" čitelný pro programátora. Jednalo se asi o nějakou vnitřní reprezentaci typu, se kterou nejspíše pracuje linker. Dnes u "solidních" překladačů by se to stát snad už nemělo. Metoda `before` zjistí, zda parametr má, či nemá být umístěn před daným objektem při řazení typů. Jedná se vlastně o obdobu operátoru `<`. Vedle těchto metod jsou pro třídu `type_info` přetíženy operátory `==` a `!=`. Všechny veřejné metody jsou konstantní a operátor `=` i kopírovací konstruktor nelze použít. Oba jsou deklarovány jako soukromé metody. Neexistuje způsob, jak by mohl programátor změnit (korektní cestou) obsah objektu.

Operátor typeid

Operátor `typeid` vrací konstantní instanci třídy `type_info`. Jeho argumentem může být název typu, nebo výraz. V prvním případě bude vrácená instance třídy `type_info` udávat zadaný typ, ve druhém případě bude udávat typ návratové hodnoty výrazu. Uveďme jednoduchý příklad:

```
#include <iostream>
#include <typeinfo>

using namespace std;

class Trida
```

```
{
        private:
                int A,B;
        public:
                void nastav(int a, int b);
};

void Trida::nastav(int a, int b)
{
        A = a;
        B = b;
}

Trida *funkce()
{
        cout << "Volani" << endl;
        return new Trida;
}

int main()
{
        Trida objekt;
        const char *nazev = typeid(objekt).name();
       /* Jen reference */
        const type_info &t1 = typeid(objekt), &t2 = typeid(Trida);
        cout << "Nazev typu objekt: " << nazev << endl;
        cout << "char < int == " << typeid(char).before(typeid(int)) << endl;
        if (t2 == t1) /* Nebo if ( typeid(objekt) == typeid(Trida) ) */
        {
                cout << "OK" << endl;
        }
        cout << typeid(funkce).name() << endl; /* Typ ukazatel na funkci */
        cout << typeid(funkce()).name() << endl; /* Návratová hodnota funkce */
        cout << typeid(*funkce()).name() << endl;
        return 0;
}
```

V tomto jednoduchém programu jsem předvedl jak pracovat s operátorem `typeid` a s instancemi třídy `type_info`. Zajímavé jsou především poslední tři výpisy. V prvním z nich zjišťuji typ identifikátoru `funkce`, což je ve skutečnosti ukazatel na funkci bez parametrů vracející ukazatel na třídu `Třída`. V předposledním výpisu zjišťuji návratovou hodnotu funkce. K zavolání funkce nedojde. V mém příkladě k žádné DYNAMICKÉ IDENTIFIKACI NEDOŠLO. Všechny identifikace šlo vyhodnotit již v době překladu a také to při překladu překladač udělal. Chceme-li identifikovat typ nějaké instance v době běhu programu, musí se jednat o instanci polymorfní třídy. Tedy třída musí mít alespoň jednu metodu volanou pozdní vazbou ("virtuální" metodu). Tato metoda v ní samozřejmě nemusí být deklarovaná, třída ji muže i zdědit. Viz moje články Časná versus pozdní vazba - úvod do polymorfismu v C++ a Polymorfismus - dokončení. Upravme v našem příkladě deklaraci metody nastav takto: `virtual void nastav(int a, int b);`.

Chování programu se nyní změní. Řádek `cout << typeid(funkce()).name() << endl;` se bude chovat stejně. Žádám vlastně o identifikaci ukazatele na třídu `Třída`. V době překladu nemůže být pochyb o tom, že se bude jednat o tento ukazatel. Jestliže ale tento ukazatel dereferencuji, již budu žádat o identifikaci typu instance polymorfního typu. Zde je situace jiná. Výraz musí být vyhodnocen (Funkce se zavolá.) a poté jej operátor `typeid` identifikuje pomocí tabulky virtuálních metod. Vše se provede v době, kdy program běží, nikoliv v době, kdy je kompilován. V TVM tedy nejsou jen adresy metod volaných pozdní vazbou, ale i informace o typu. Tento příklad neukazuje nejlépe rozdíly mezi identifikací typů v době kompilace a v době běhu programu. Snažil jsem se jen poukázat na fakt, že v tomto případě bude výraz vyhodnocen. Může se jednat o velmi častý "zdroj" chyb, protože výraz může mít nějaký "vedlejší efekt". Například může změnit globální proměnné, atd... V našem příkladě nastane jiný problém, že ve funkci bude vytvořena instance, která nebude nikdy zlikvidována. Je důležité nezapomenout, že v případě identifikace polymorfního typu vlastně dojde k vyhodnocení výrazu. Nyní vytvořme příklad, který lépe ukáže rozdíl mezi statickou a dynamickou identifikací.

```
#include <iostream>
#include <typeinfo>

using namespace std;

class NadTrida
{
        private:
                int Atribut;
        public:
                virtual void nastav(int a);
};

class PodTrida : public NadTrida
{};

void NadTrida::nastav(int a)
{
        Atribut = a;
}

int main()
{
        NadTrida *a = new NadTrida;
        NadTrida *b = new PodTrida; /* Ukazatel b "ukazuje" na PodTridu */
        if (typeid(*b) == typeid(NadTrida))
        {
                cout << "Typ identifikovan v dobe prekladu." << endl;
        }
        else
        {
                cout << "Typ identifikovan pri behu programu." << endl;
        }
        cout << "Ukazatel " << typeid(a).name() <<
```

```
          " se odkazuje na " << typeid(*a).name() << endl;
         cout << "Ukazatel " << typeid(b).name() <<
          " se odkazuje na " << typeid(*b).name() << endl;
         return 0;
}
```

Výraz `typeid(b)` bude vyhodnocen při překladu. Překladač jasně vidí, že `b` je deklarován jako ukazatel. Nemůže si ale být jistý, že tento ukazatel ukazuje na objekt typu `Nadtřída`. `Nadtřída` je totiž polymorfní typ. Odebereme-li v deklaraci metod třídy `Nadtřída` klíčové slovo `virtual` (rozhodně si to zkuste), bude překladač předpokládat, že ukazatel na třídu `Nadtřída` bude ukazovat na instanci třídy `Nadtřída`. Což ale není pravda. V toto případě překladač identifikuje `*b` jako instanci třídy `Nadtřída` a k žádné dynamické identifikaci nedojde. Vše bude rozpoznáno "staticky" při překladu programu.

Když dynamická identifikace selže

Nelze-li určit typ objektu, vyvrhne operátor `typeid` výjimku třídy `bad_typeid`. Třída `bad_typeid` je potomkem třídy `exception`. Viz moje předchozí tři články. Bezpečné zjištění typu by tedy vypadalo následovně: (Třídy jsou deklarovány v předchozím příkladu.)

```
int main()
{
         NadTrida *p = NULL;
         try
         {
                  cout << "Typ:" << typeid(*p).name() << endl;
         }
         catch (std::bad_typeid &e)
         {
                  /* Nějaké ošetření výjimky. Identifikace typu se nepovedla. */
                  cerr << "Odchycena vyjimka" << endl;
         }
         return 0;
}
```

V některých starších překladačích, které nevyhovují normě se může třída `bad_typeid` jmenovat `Bad_typeid` a také nemusí být potomkem třídy `exception`. Jak vidíme z příkladu, výjimka může být vyvržená například v případě, že jako argument operátoru `typeid` je `NULL`, který má být dereferencován.

Závěrem k identifikaci typů

Jak je vidět, C++ má poměrně jednoduchou identifikaci typů. Je proto chybný postup přidávat do tříd atribut, který nějakým způsobem udává typ, jak mnoho programátorů chybně dělá. Buď má instance tabulku virtuálních metod, kde je tato informace již uložená, nebo dojde k identifikaci v době

překladu, potom je takový atribut stejně zbytečný. Při použití operátoru `typeid` si musíme jasně uvědomit, jestli chceme identifikovat polymorfní typ, nebo cokoliv jiného. Polymorfní typ bude identifikován až v době, kdy program poběží, a hlavně nesmíme zapomínat, že v tomto případě dojde k vyhodnocení argumentu (výrazu).

Tolik tedy k dynamické identifikaci typů. Příště se podíváme na téma, které s identifikací typů úzce souvisí. Příští článek bude o dynamickém přetypování v C++.