Inteligentny wskaźnik `std::shared_ptr`

dr inż. Ireneusz Szcześniak

jesień 2017 roku

Współdzielenie obiektów

- Współdzielenie obiektu jest wymagane, gdy obiekt:
  - jest dynamicznie tworzony,
  - używany jest, być może współbieżnie, przez różne części programu,
  - nie wiadomo, kiedy i gdzie dokładnie będzie zniszczony.
- Nie wiadomo, gdzie dokładnie obiekt będzie zniszczony, bo:
  - przetwarzanie obiektu może zależeć od jego danych,
  - różne współbieżne części programu mogą w różnym czasie zakończyć działanie na obiekcie.

`std::shared_ptr`

- `#include <memory>`
- Klasa C++11 implementująca współdzielenie dynamicznego obiektu.
- Przeciwstawność `std::unique_ptr`.
- Obiekty tej klasy można kopiować i przenosić.
- Kopiując obiekt `std::shared_ptr`, tworzymy grupę obiektów zarządzających, które współdzielą jeden obiekt dynamiczny.
- Grupa obiektów `shared_ptr` jest właścicielem zarządzanego obiektu.
- Grupa posiada jeden licznik odwołań do zarządzanego obiektu.
- Zarządzane obiekty nie wiedzą, że są zarządzane.
- Użycie tak wydajne czasowo, jak surowego wskaźnika.
- Zajmuje dwa razy więcej pamięci, niż surowy wskaźnik.

Użycie

- Deklaracja: `shared_ptr<A> a;`
- Oddanie pod zarządzanie: `shared_ptr<A> sp(new A());`
- Oddanie pod zarządzanie: `sp.reset(new A());`
- Kopiowanie obiektów: `shared_ptr<A> sp2(sp);`
- Kopiowanie obiektów: `sp2 = sp;`
- Przenoszenie obiektów: `shared_ptr<A> sp2(move(sp));`
- Przenoszenie obiektów: `sp2 = move(sp);`
- Niszczenie: tego nie robimy, to odbywa się automatycznie.

Jak to działa?

- Grupa obiektów `shared_ptr` współdzieli jedną strukturę zarządzającą, alokowaną dynamicznie.
- Każdy obiekt `shared_ptr` posiada wskaźnik na strukturę zarządzającą.
- Częścią struktury zarządzającej jest licznik odwołań.
- Licznik odwołań to inaczej wielkość grupy.
- Przy kopiowaniu obiektów `shared_ptr`, licznik odwołań jest inkrementowany.
- Przy niszczeniu obiektów `shared_ptr`, licznik odwołań jest dekrementowany.
- Kiedy licznik odwołań wyniesie 0, obiekt zarządzany jest niszczony.

Z unique_ptr<T> do shared_ptr<T>

Można tak? Można.

```
unique_ptr<A> up(new A("A1"));
shared_ptr<A> sp(up.release());
```

Ale lepiej tak:

```
unique_ptr<A> up(new A("A1"));
shared_ptr<A> sp(move(up));
```

Można, bo `shared_ptr<T>` ma konstruktor, który przyjmuje referencję typu rvalue do obiektu typu `unique_ptr<T>`.

Dzięki temu możemy tworzyć `shared_ptr<T>` z obiektów tymczasowych typu `unique_ptr<T>`, np. zwracanych jako wynik wywołania funkcji.

Użycie współbieżne

- Klasa częściowo jest bezpieczna w programowaniu współbieżnym.
- Jeżeli wątek ma swoją kopię `shared_ptr`, to może jej swobodnie używać bez synchronizacji, włącznie z jej przenoszeniem i tworzeniem kopii.
- Jeżeli wątki używają tego samego obiektu `shared_ptr`, to muszą synchronizować działania, jeżeli obiekt jest zmieniany.

Wydajność

- Obiekt `shared_ptr` zajmuje dwa razy więcej pamięci niż surowy wskaźnik, bo zawiera dwa pola:
  - wskaźnik na zarządzany obiekt,
  - wskaźnik na strukturę zarządającą.
- Dodatkowo jest dynamicznie alokowana pamięć na strukturę zarządzającą.
- Wskaźnik na zarządzany obiekt mógłby być częścią struktury zarządzającej, ale wtedy odwołanie do obiektu zarządzanego byłoby wolniejsze.

std::make_shared

Zamiast pisać typ A dwa razy w sposobie na piechotę:

```
shared_ptr<A> sp(new A("A1"));
```

Możemy napisać typ A tylko raz używając funkcji `make_shared`:

```
auto sp = make_shared<A>("A1");
```

Obiekt stworzony przez `make_shared` jest przenoszony, a nie kopiowany.

ZALETA #1: tworzenie obiektu zarządzanego i zarządzającego odbywa się w jedyn kroku, co jest bezpieczne pod względem obsługi wyjątków.

ZALETA #2: Funkcja `std::make_shared` tworzy obiekt zarządzany i zarządzający używając jednej alokacji pamięci, przez co jest szybsza niż osobne tworzenie tych obiektów.

Podsumowanie

- Klasa `shared_ptr<T>` pozwala na łatwe współdzielenie obiektów, które były stworzone dynamicznie.
- Główne zadanie: zniszczyć obiekt zarządzany, kiedy nie jest już potrzebny.
- Obiekty klasy `shared_ptr<T>` są dwa razy większe niż surowy wskaźnik.
- Można łatwo tworzyć obiekty `shared_ptr<T>` z `unique_ptr<T>`.

Dziękuję za uwagę.