Inteligentny wskaźnik `std::weak_ptr`

dr inż. Ireneusz Szcześniak

jesień 2017 roku

Współdzielenie bez uwiązania

Współdzielenie bez uwiązania polega na możliwości ewentualnego użycia współdzielonego obiektu, ale bez zachowania gwarancji, że obiekt będzie ciągle istniał.

Mając obiekt współdzielony, chcemy jedynie mieć możliwość ewentualnego odwołania się do niego, jeżeli istnieje. Jeżeli obiekt już nie istnieje, to chcemy jedynie mieć możliwość sprawdzenia tego.

Przykład użycia: fabryka obiektów współdzielonych

- Jest fabryka (czyli funkcja zwracająca obiekty) wracająca obiekty współdzielone z użyciem `shared_ptr<A>`.
- Jeżeli obiekt nie istnieje, to fabryka go tworzy.
- Obiekt współdzielony jest niszczony, jeżeli nie jest już potrzebny.
- Fabryka nie współdzieli obiektu (nie uwiązuje go), ale chce mieć możliwość ponownego zwrócenia go, jeżeli on jeszcze istnieje i jeżeli będzie potrzebny.
- ROZWIĄZANIE: fabryka powinna pamiętać zwrócone obiekty współdzielone, ale bez uwiązywania ich, żeby mogły być zniszczone, jak nie będą już potrzebne.

`std::weak_ptr`

- `#include <memory>`
- Klasa C++11 implementująca współdzielenie bez uwiązania.
- Obiekty tej klasy można kopiować i przenosić, ale nie ma to szczególnego znaczenia, jak przy `unique_ptr` czy `shared_ptr`.
- Nierozerwalnie związany z `shared_ptr`.
- Obiekt `weak_ptr` tworzony jest na podstawie obiektu `shared_ptr`.
- Nie niszczy obiektu, do którego się odwołuje.
- Zarządzane obiekty nie wiedzą, że są zarządzane.

Użycie `weak_ptr`

- Mając: `shared_ptr<A> sp;`
- Deklaracja: `weak_ptr<A> wp;`
- Tworzenie: `weak_ptr<A> wp(sp);`
- Tworzenie: `wp = sp;`
- Kopiowanie obiektów: `weak_ptr<A> wp2(wp);`
- Kopiowanie obiektów: `wp2 = wp;`
- Przenoszenie obiektów: `weak_ptr<A> wp2(move(wp));`
- Przenoszenie obiektów: `wp2 = move(wp);`
- Czy obiekt został już zniszczony? `wp.expired();`
- Ale jak pozyskać wskaźnik na obiekt współdzielony?

Pozyskanie obiektu współdzielonego

PROBLEM: Jak bezpiecznie użyć (bez hazadru), skoro `weak_ptr` nie daje nam gwarancji, że obiekt nie zostanie zniszczony? Taką gwarancję daje nam `shared_ptr`.

Przecież bezpośrednio po uzyskaniu surowego wskaźnika obiekt może być zniszczony.

ROZWIĄZANIE: tworzenie `shared_ptr` na podstawie `weak_ptr`.

Robimy to tak (może rzucić wyjątek):

```
shared_ptr<A> sp(wp);
```

Albo tak (obiekt sp może być `nullptr`):

```
shared_ptr<A> sp = wp.lock();
```

Jak to działa?

- Grupa obiektów `shared_ptr` i `weak_ptr` współdzielą jedną strukturę zarządzającą, alokowaną dynamicznie.
- Każdy obiekt `shared_ptr` i `weak_ptr` posiada wskaźnik na strukturę zarządzającą.
- Częścią struktury zarządzającej jest licznik odwołań tylko obiektów `shared_ptr`.
- Inną częścią struktury zarządzającej jest licznik odwołań obiektów `shared_ptr` i `weak_ptr`.
- Kiedy licznik odwołań `shared_ptr` wyniesie 0, obiekt zarządzany jest niszczony.
- Kiedy licznik odwołań `shared_ptr` i `weak_ptr` wyniesie 0, struktura zarządzająca jest niszczona.

Użycie współbieżne

- Klasa jest bezpieczna w programowaniu współbieżnym.
- Pozyskany obiekt klasy `shared_ptr` możemy swobodnie używać, bo jest on wyłączną kopią wątku.

Wydajność

- Obiekt `weak_ptr` zajmuje dwa razy więcej pamięci niż surowy wskaźnik, bo zawiera dwa pola:
  - wskaźnik na zarządzany obiekt,
  - wskaźnik na strukturę zarządającą.
- Po co wskaźnik na zarządzany obiekt, skoro i tak użytkownik nie ma do niego dostępu? Bo potrzebny jest przy tworzeniu obiektu `shared_ptr`.
- Wskaźnik na zarządzany obiekt mógłby być częścią struktury zarządzającej, ale wtedy dla obiektów `shared_ptr` odwołanie do obiektu zarządzanego byłoby wolniejsze (bo byłoby dodatkowo pośrednie).

Podsumowanie

- Klasa `weak_ptr` implementuje współdzielenie bez uwiązania.
- Bez uwiązania, czyli bez gwarancji, że obiekt, do którego się odwołujemy, będzie istniał.
- Główne zadanie: pozwala użyć obiekt współdzielony, jeżeli jeszcze istnieje.
- Nie niszczy obiektu, do którego się odwołuje.

Dziękuję za uwagę.