Умные указатели

13 июля 2017 г.

Почему плохи обычные указатели?

- Непонятно, на что именно они указывают: на один объект или на массив.
- По объявлению непонятно, единственный ли указатель указывает на объект.
- Нужно удалить через delete или delete[]?
- Нужно обеспечить уничтожение данных ровно один раз.
- Трудно проверить, не является ли указатель висячим (не указывает ли он на nullptr).

Проблема

```
#include <fstream>
#include <iostream>

void f(){
    int* px = new int;
    px = 10;
    std::ifstream filein("file.txt");
    // …
    filein.close();
    delete px;
}
```

До вызова оператора delete дело может не дойти

Умные (интеллектуальные) указатели

- обеспечивают ту же функциональность, что и обычные указатели
- упрощают работу с памятью
- позволяют избежать многих ошибок

auto_ptr (C++98)

shared_ptr (C++11)

unique_ptr (C++11)

weak_ptr (C++11)

Умный указатель: реализация

- устроен как шаблонный класс
- есть конструктор, который принимает указатель
- хранит адрес объекта, созданного через new
- когда истекает время жизни, деструктор использует операцию delete
- перегружены операторы * и ->

```
#include <memory>

std::auto_ptr<double> px1(new double);
std::unique_ptr<std::string> px2(new std::string);
std::shared_ptr<int> px3(new int);
```

Умный указатель: пример

```
#include <memory>

std::shared_ptr<double>  px1(new double);     // OK

double* px = new double;
*px = 9.99;
std::shared_ptr<double> px2(px);     // OK
std:cout << *px2 << std::endl;           // 9.99
std::shared_ptr<double> px3 = px;   // Error

std::shared_ptr<double> px4;
px4 = std::shared_ptr<double>(px);   // OK
px4 = px;    // Error


double y = 10.1;
std::shared_ptr<double> py(&y);  // Error! Объект на стеке
```

std::auto_ptr

- при присваивании реализовывает семантику перемещения, а не копирования
- при присваивании умного указателя А другому умному указателю Б объект А теряет владение объектом и начинает указывать на nullptr
- нельзя использовать в контейнерах STL
- удален из стандарта C++17

auto_ptr (пример 1)

```
#include <memory>


void f() {
   std::auto_ptr<double>  px(new double(10.1));
   std::auto_ptr<double>  new_px;

   new_px = px;

   std::cout << *px << std::endl;     // Error
   std::cout << *new_px << std::endl;     // OK
}
```

auto_ptr (пример 2)

```
void f() {
    std::auto_ptr<std::string> arr[4] = {
            std::auto_ptr<std::string>(new std::string("One")),
            std::auto_ptr<std::string>(new std::string("Two")),
            std::auto_ptr<std::string>(new std::string("Three")),
            std::auto_ptr<std::string>(new std::string("Ten"))
    };
    for (int i = 0; i < 4; ++i) { std::cout << *(arr[i]) << " "; }
    std::cout << std::endl;

    std::auto_ptr<std::string> elem = arr[0];

    for (int i = 0; i < 4; ++i) { std::cout << *arr[i] << " "; }
    std::cout << std::endl;
}
```

std::shared_ptr

- несколько указателей могут владеть одним объектом
- реализован подсчет ссылок (учитывается, сколько умных указателей ссылается на объект)
- при присваивании число ссылок увеличивается на 1, а при удалении указателя – уменьшается
- сам объект удаляется только тогда, когда число ссылок становится равно 0
- можно использовать в контейнерах STL

shared_ptr (пример 1)

```
#include <memory>


void f() {
   std::shared_ptr<double>  px(new double(10.1));
   std::shared_ptr<double>  new_px;

   new_px = px;

   std::cout << *px << std::endl;     // OK
   std::cout << *new_px << std::endl;     // OK
}
```

shared_ptr (пример 2)

```
void f() {
    std::shared_ptr<std::string> arr[4] = {
            std::shared_ptr<std::string>(new std::string("One")),
            std::shared_ptr<std::string>(new std::string("Two")),
            std::shared_ptr<std::string>(new std::string("Three")),
            std::shared_ptr<std::string>(new std::string("Ten"))
    };
    for (int i = 0; i < 4; ++i) { std::cout << *(arr[i]) << " "; }
    std::cout << std::endl;

    std::shared_ptr<std::string> elem = arr[0];

    for (int i = 0; i < 4; ++i) { std::cout << *arr[i] << " "; }
    std::cout << std::endl;
}
```

std::unique_ptr

- два указателя не могут владеть одним объектом
- обычно явное присваивание одного умного указателя другому запрещено (ошибка при компиляции)
- разрешено присваивание умного указателя другому, если это временный объект
- можно осуществить присваивание с помощью функции std::move()
- можно использовать для массивов

unique_ptr (пример 1)

```
#include <memory>


void f() {
   std::unique_ptr<double>  px(new double(10.1));
   std::unique_ptr<double>  new_px;

   new_px = px;    // Error

   std::cout << *px << std::endl;
   std::cout << *new_px << std::endl;
}
```

unique_ptr (пример 2)

```
#include <memory>

std::unique_ptr<std::string> make_unique(std::string* str){
        return std::unique_ptr<std::string>(str);
}

void f() {
   std::string* str = new std::string("To be, or not to be");
   std::unique_ptr<std::string> ptr;
   ptr = make_unique(str);      // OK

   std::cout << *ptr << std::endl;
   std::cout << *str << std::endl;
}
```

unique_ptr (пример 3)

```
void f() {
    std::unique_ptr<std::string> arr[4] = {
            std::unique_ptr<std::string>(new std::string("One")),
            std::unique_ptr<std::string>(new std::string("Two")),
            std::unique_ptr<std::string>(new std::string("Three")),
            std::unique_ptr<std::string>(new std::string("Ten"))
    };
    for (int i = 0; i < 4; ++i) { std::cout << *(arr[i]) << " "; }
    std::cout << std::endl;

    std::unique_ptr<std::string> elem = std::move(arr[0]);
    arr[0] = std::unique_ptr<std::string>(new std::string("Zero"));

    for (int i = 0; i < 4; ++i) { std::cout << *arr[i] << " "; }
    std::cout << std::endl;
}
```

unique_ptr (пример 4)

```
void f() {
    std::unique_ptr<int[ ]> arr =
                                std::unique_ptr<int[ ]>(new int[10]);
    for (int i = 0; i < 10; ++i) { arr[i] = i; }

    for (int i = 0; i < 10; ++i) { std::cout << arr[i] << " "; }
    std::cout << std::endl;
}
```

unique_ptr: методы

unique_ptr up();	конструктор по умолчанию, **up** содержит nullptr
unique_ptr up(p);	конструктор с параметром, **up** содержит **p**
unique_ptr up(up1);	**up** содержит **up1**, **up1** содержит nullptr
p = up.get();	получение обычного указателя из умного
p = up.release();	получение обычного указателя из умного, **up** содержит nullptr
up.reset(p);	удаление старого указателя из **up** и запись нового указателя **p**

Задание

Попробуйте реализовать шаблон класса auto_ptr для умного указателя. В классе должны быть следующие методы:

- несколько конструкторов
- деструктор
- оператор присваивания =
- операторы * и –>
- метод get()
- метод reset().

Вопросы?